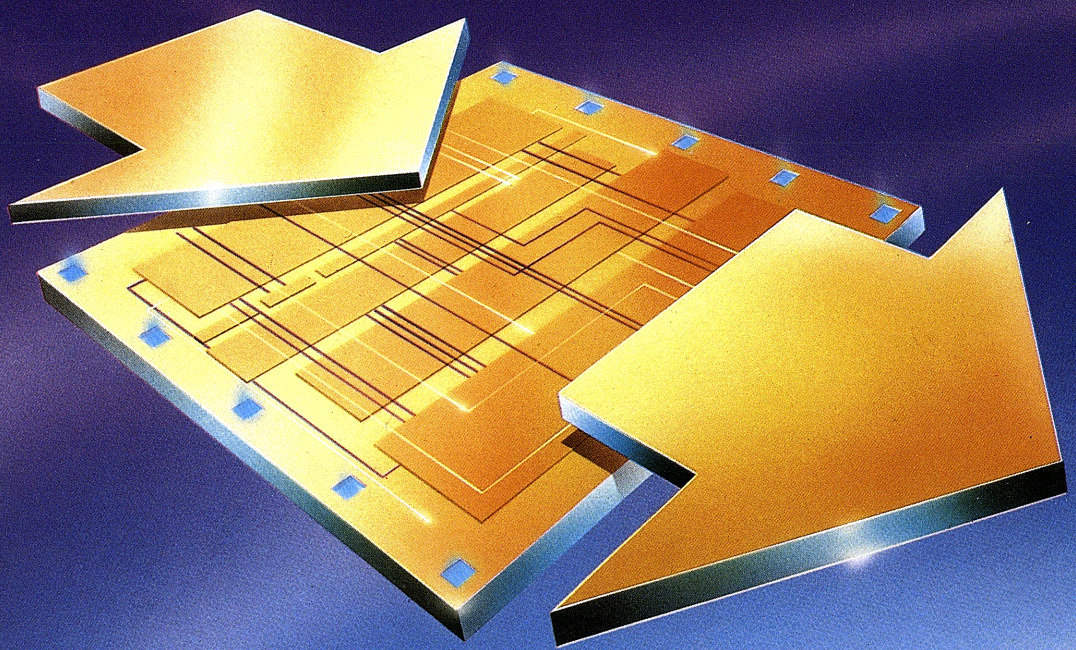
SINGLE-CHIP 8-BIT MICROCONTROLLERS

USER MANUAL 1988



PHILIPS

SINGLE-CHIP 8-BIT MICROCONTROLLERS USER MANUAL 1988

CONTENTS

		page
1.	INTRODUCTION	1-1
2.	THE MAB8048/80C49 MICROCONTROLLER FAMILY.	2-1
3.	THE MAB8051/C51/52 MICROCONTROLLER FAMILY	3-1
	3.1 SC80/83C451	3.1-1
	3.2 PCB80/83C552	3.2-1
	3.3 PCB80/83C652	3.3-1
4.	THE MAB84X1 MICROCONTROLLER FAMILY	4-1
5.	THE MAB8422/42 MICROCONTROLLER FAMILY	5-1
6.	THE PCF84CXX MICROCONTROLLER FAMILY	6-1
	6.1 PCF84C12	6.1-1
	6.2 PCF84C430	6.2-1
	6.3 PCF84C85	6.3-1
7.	THE MAB84X1/PCF84CXX serial I/O	7-1
8.	THE I^2C-BUS SPECIFICATION	8-1
9.	THE MAB8048/MAB84X1/PCF84CXX INSTRUCTION SET	9-1
10.	SOFTWARE EXAMPLES FOR THE MAB8048/MAB84X1/PCF84CXX FAMILY	10-1
11.	THE D^2B SPECIFICATION	11-1
12.	MICROCONTROLLER TEST AND DEVELOPMENT SYSTEMS	12-1
13.	APPLICATION NOTES	13-1
14.	I^2C-BUS SUPPORT CHIP SET	14-1
15.	ORDER ENTRY FORMS	15-1

For the electrical and timing characteristics of all microcontrollers described herein, please refer to current data handbook IC14.

1. Introduction

CONTENTS – INTRODUCTION

AN INTRODUCTION TO MICROCONTROLLERS

		page
1.0	APPLICATIONS OF MICROCONTROLLERS	1-4
2.0	MICROPROCESSORS AND MICROCONTROLLERS	1-4
3.0	THE BASIC MICROCONTROLLER SYSTEM	1-5
4.0	RAMs, ROMs AND PROMs	1-6
5.0	PORTS AND BUSES	1-7
6.0	CENTRAL PROCESSING UNIT (CPU)	1-8
7.0	REGISTERS	1-10
7.1	Accumulator	1-10
7.2	Working registers	1-10
7.3	Program Counter and Stack	1-10
7.4	Program Status Word	1-11
7.5	Instruction Register	1-11
8.0	MEMORY EXPANSION	1-12
9.0	INPUT/OUTPUT PORTS	1-13
9.1	Parallel Input/Output	1-13
9.2	Serial Input/Output	1-13
10.0	COMPUTER TIMING	1-14
10.1	Timer/Counter	1-14
11.0	MICROCONTROLLER PROGRAMMING	1-15

AN INTRODUCTION TO MICROCONTROLLERS

For those not familiar with microcontrollers, this section contains background information which will prove useful when reading other manual sections. If you are already versed in computer concepts and techniques, this material can be ignored.

1.0 MICROPROCESSORS AND MICROCONTROLLERS

To avoid confusion between microcontrollers and microprocessors, a definition of the two may prove helpful:-

A microcontroller is a single LSI (Large Scale Integration) chip capable of implementing arithmetic and control functions. The microprocessor itself is only one of the functional units which form a working computer. The other major units are I/O (Input/Output) interfaces and data and program memory. Input/Output interfaces provide a means of communicating with external sources, while data and program memory permit the storage of information and instructions. All units are joined by connections called buses, simply paths for signals having a common function.

A microcontroller is a complete system, including a CPU (Central Processing Unit), memory and I/O interfaces. The advantages of using single-chip microcontrollers including high reliability, because few external connections are necessary, short development time and fast assembly.

2.0 APPLICATIONS OF MICROCONTROLLERS

What was once the domain of passive and programmable logic in control systems is now the territory of the microcontroller. Low-cost and increasing flexibility has led to microcontrollers seeing widespread use in everyday control applications. Today's microcontroller requires little peripheral equipment and is found in everyday devices such as cars, washing machines,telephony equipment and televisions etc. However, it is in the realm of industrial and domestic control applications that the microcontrollers and microprocessors role has seriously expanded. The data processing capability of these devices merits only a small production market when compared with their control possibilities.

When considering the design of a microcontroller system inportant factors come under two broad headings:

Technical parameters

What will the system control?
What are the data I/O requirements?
What are the software requirements?
What operating frequency will the microcontroller have?
Which microcontroller is best for the application?

Cost

Will the system be cost effective?
Can it be done cheaper with conventional logic?

After these design considerations have been resolved and trade-offs weighed up, and there is always some trade-off! serious design work can begin.

The potential of the microcontroller is now universally recognized. However, what is not so widely appreciated is that these devices, which combine versatility with simplicity, can also be used by those with limited knowledge of electronics but who wish to design or improve existing control applications.

3.0 THE BASIC MICROCOMPUTER SYSTEM

A typical digital computer system consists of:

Central Processing Unit (CPU)
Program Memory (Read Only Memory)
Data Memory (Random Access Memory)
Input/Output Ports (I/O)

Some advanced products may also contain:

A/D and DAC hardware
I^2C compatibility

Instructions for control olperation are stored in Program Memory. Data itself, the group of operands to be processed, is stored in the Data Memory. Each instruction from Program Memory is 'read' by the CPU in a logical sequence to perform various processing functions. Fig. 1 shows a block diagram of a microcontroller.

4.0 RAMs, ROMs, AND PROMs

A microcontroller cannot operate until a program is present. A program must therefore exist, ready for use, before the microcontroller is switched on.

This type of memory is called ROM - Read Only Memory - the name implying that stored information cannot be changed by writing actions.

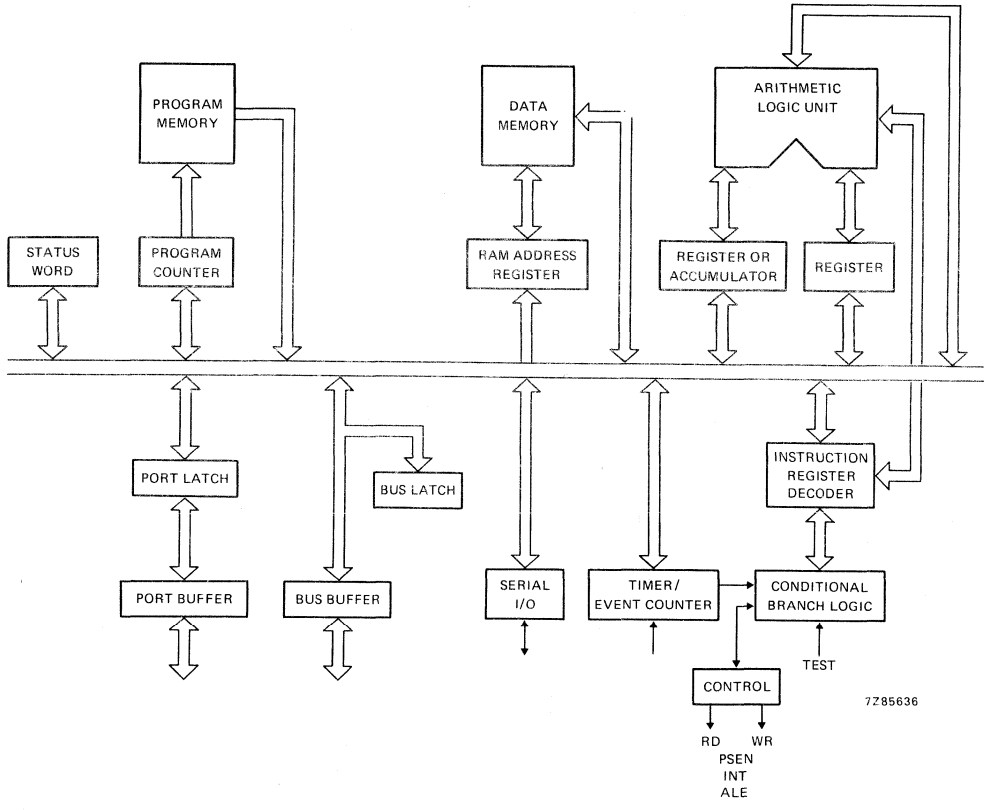


Fig. 1 Block diagram of a microcontroller.

There are three main types of ROM:-

Mask programmed - programmed during manufacture
Fusible link ROM - programmed by the user, ONCE. This memory consists of an array of tracks (or fuses) which can be selectively 'blown' forming a binary pattern.
Erasable programmable ROM - EPROM and E^2PROM

The 'fusible link' type ROM is better known as a PROM (Programmable Read Only Memory) and is mainly used for prototypes or short production runs. However, once a program is entered i.e. the PROM is 'blown', the process cannot be reversed

Often some sort of semi-permanent memory is required, where the program may be changed at any time. The data in this memory (like the fusible link) must be independent of the supply voltage. The Erasable Programmable Read Only memory (EPROM) was developed to fulfill this need. Data stored in an EPROM is erased by the shining of an ultra-violet light source on the memory chip surface. More often now, an EPROM may be incorporated on-chip and 'blown' by the user according to the application.

Microcontrollers cannot live on ROM alone, some memory is needed to store operational results and to temporarily store quantities to be used by the Central Processor. This memory must be written to as well as read from, so the read/write memory used in a microcontroller system is called RAM - (Random Access Memory).

Becoming more available today, is the non-volatile memory or EEPROM sometimes written E^2PROM - Electrically eraseable PROM. Using this type of memory for either ROM or RAM enables data placed in memory to be electrically erased at a later date.

5.0 PORTS AND BUSES

Each unit within a microcontroller system is linked by a bus(es). These paths carry addresses and data enabling binary data to be rapidly accessed by the CPU. If memory is not large enough for a particular application memory space may be expanded.

Ports enable a computer to have access to the 'outside world'. Input Ports enable the computer to receive information from sources such as switches or sensors, in order to monitor external events. Output Ports are used to transmit results to devices beyond the system, such as LEDs, relays, motors or even other microcontrollers.

Input/Output may be represented on hardware/port basis or as memory mapped I/O. On a port basis an instruction to the relevant port is executed before information can be transferred. When memory mapped, information is read or written in the same way as read/write actions to a memory.

6.0 CENTRAL PROCESSING UNIT (CPU)

A CPU is the nerve centre of any digital computer system, since it coordinates and controls the activities of the other units and performs all arithmetic and logical processes to the applied data. It references memory and I/O ports when executing instructions and recognizes and responds to external control signals. A typical CPU consists of interconnected functional units:-

Registers
Arithmetic/Logic Unit (ALU)
Control Circuits (e.g. Instruction Decoder)
Interrupt logic

Registers are temporary storage units within the CPU. Some registers, such as the program counter and instruction register, have dedicated uses. Others, such as the accumulator, are for more general use.

The Arithmetic/Logic Unit, as its name implies, performs arithmetic and logical operations on the binary data. It contains an Adder which is able to combine the contents of two registers. Using only this Adder, a capable programmer can write routines to subtract, multiply and divide, giving the machine full arithmetic capability. In practice however, most ALUs are provided with built-in functions, including Boolean logic operations and shift capabilities. Flag bits are contained in the ALU to specify certain conditions that arise in the course of arithmetic and logical manipulations. It is possible to program 'jumps' which are conditionally dependent on the status of one or more of these flags.

The control circuitry is the primary functional unit within a CPU. Using clock signals, it maintains the proper processing event sequence. After an instruction is fetched and decoded, the control circuitry issues signals to external and internal units to initiate proper processing actions.

Often the control circuitry will be called upon to respond to external signals, for example, an interrupt. An interrupt request causes the control circuitry to temporarily interrupt main program execution, jump to a special 'interrupt service routine' then automatically return to the main program. Interrupts are not always external in nature and can be caused by internal events such as a timer interrupt or a serial I/O interrupt.

Interrupt are required to use the processing power of the microcontroller to best effect. Often the computer is working with external devices operating at slower speeds than itself. Printers, displays and disk stores are typical examples of such devices. To avoid the microcontroller 'idling' whilst the Peripheral performs the task required, the main program is continued until an interrupt is received which halts the execution of the main program to allow transference of the next set of data.

Another method of servicing peripheral equipment is by polling. Polling involves testing the input ports and/or the interrupt pin in rotation for the presence of a signal. This service rotation is controlled by software. When a signal is detected the program will hand over control to a subroutine or interrupt service routine. This routine will usually be dedicated to a particular device and operation.

Interrupts are also important when the functions carried out by the microcontroller depend on external events. An interrupt in this case could indicate an over-limit value in a control situation or a failure of one of the peripherals. The microcontroller is then able to perform the relevant interrupt routine to deal with these events.

Several interrupting devices may share the same microcontroller. In this case each is assigned a different priority level so that if interrupts occur simultaneously, those considered most important are treated first.

7.0 REGISTERS

Each register in a computer has specific properties for use during arithmetic and/or logical operations. The following are the most important:-

7.1 Accumulator

The accumulator is a register in which arithmetic results are created. It contains the operand upon which the operation takes place and in some case this operand will be replaced by the result. Often a CPU will include a number of additional general purpose registers to store operands or intermediate data. These registers eliminate the need to constantly move intermediate results between memory and the accumulator.

7.2 Working Registers

The first eight locations (0 - 7) of the data memory array are usually assigned as 8-bit working registers and are directly addressable by several instructions. The remaining memory locations are usually indirectly addressable through the RAM Pointer Register within the register array. Since the working registers are more easily addressed, they are usually used to store frequentlyaccessed intermediate results.

7.3 Program Counter and Stack

As the CPU references memory contents when carrying out processing actions, it must know which location contains the next instruction. Each memory location is numbered to distinguish it from any other. This distinguishing number is called an Address . The register which holds the address of the next instriction in a processing sequence is called the Program Counter. The processor updates the counter each time it fetches an instruction. The Program Counter therefore is always pointing to the next instruction.

Instructions are stored in numerically adjacent addresses, with the lower byte containing the op-code or instruction to be executed and the higher byte containing the operand or data. The only time a programmer avoids sequential addressing is when an instruction is placed in one memory section to jump to another, or when an interrupt instruction (INT) forces a jump.

A jump instruction is a departure from the normal sequence of program steps. The next instruction in the sequence may be stored in another memory location. During the execution of a jump, the processor replaces the contents of the Program Counter with the address to which the jump is made.

An important type of program jump involves the 'calling' of a subroutine by the main program. A subroutine is a sequence of instructions which perform a specific and frequent process. By 'calling' the subroutine the programmer can avoid unecessary repetition of this set of instructions in the main program. A subroutine may be considered as a program within a program (see section 12).

There are two main reasons for using a subroutine:-

1) Certain routines are of a general nature and are common to many programs. Arithmetic functions such as square roots, sines or cosines are good examples of common subroutines.

2) Certain sections of a particular program may be required at several points in the main program. Storage space is saved by making these sections into subroutines.

When the program 'calls' a subroutine, the processor 'stores' the current contents pf the Program Counter in order to resume execution of the program at the pointer immediately following the 'call' instruction. The reserved data memory area which stores this Program Counter value is called the Stack.

The Program Counter Stack is implemented using registers in Data Memory. Those to be used are determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW). Each time Program Counter contents is stored in the stack, the Stack Pointer is incremented and poinds to the next location in anticipation of another CALL.

Subroutines are often 'nested', that is, the subroutine contains a structure similar to itself. Entry to a lower level subroutine can be achieved via any of the high level subroutines. The maximum depth of nesting is determined by the depth of the stack. If overflow occurs, the deepest address stored will be overwritten and lost since the stack pointer overflows from 111 to 000.

The end of a subroutine, which requires a return instruction (RET) causes the last stored program counter value in the Stack to be transferred to the Program Counter.

7.4 Program Status Word

The Program Status word (PSW) is an 8-bit word defining system status. Although the actual bit allocation of PSW varies for different microcontroller systems. In general the following information is stored:

- Stack Pointer Bits
- Working Register Bank Switch Bit
- Flag bits
- Auxiliary Carry bit
- Carry flag

7.5 Instruction Register

An instruction is fetched in two distinct operations. First, the address in the Program Counter is transmitted by the processor, via the address bus, to the program memory. Secondly, the addressed byte is then returned to the processor from the memory. This byte is then stored in the Instruction Register and is used to direct activities during the remainder of the instruction execution.

The 8-bits stored in the instruction register are decoded and interpretedas one of 256 possible instructions, more than adequate for most processors.

Every computer has a word length characteristic to that type of machine. A word is the basic unit of data in a computer memory that can be processed as an entity. Word length is usually determined by the size of internal storage registers and interconnecting buses. An 8-bit parallel processor, for example, is most efficient in dealing with 8-bit binary numbers, or numbers that are integral multiples of 8-bits. An 8-bit word is generally known as a byte .

8.0 MEMORY EXPANSION

Two possibilities exist for memory expansion when using a microcontroller family:

- Versions with more on-chip ROM and RAM may be available which are upwardly compatible and able to replace the original chip used.

- Memory may be expanded by adding more ROM or RAM external to the microcontroller, up to a limiting value dependant upon the type of controller used. Although connected externally, this additional memory is still treated as part of the computer system and is therefore linked to the rest of the system by the bus

An expansion of program memory means that external instruction fetch cycles have to be implemented. For these cycles, signals ALE (Address Latch Enable) and PSEN (Program Stored Enable) are used to control the fetching of the instruction. ALE indicates the time at which an address to external memory is valid. Usually the trailing edge of ALE is used to latch the address externally. PSEN indicates than an external instruction fetch is in progress and serves to enable the external memory device.

For the expansion of data memory, a read or write cycle occurs as follows:- The address is placed onto the bus. ALE indicates that the address is valid. A read (RD) or write (WR) pulse indicates the type of data memory access in progress and data is transferred in or out, over the bus.

9.0 INPUT/OUTPUT PORTS

Input and Output operations are similar in nature to memory read/write operations with the exception that an I/O port is addressed instead of a memory location. The CPU issues the appropriate input or output control signal, transmits the address and either receives the data being input or sends the data to be output.

9.1 Parallel Input/Output

Parallel output devices are often used with microcontrollers. The parallel transmission of data gives output devices the advantage of high speed, with the complete byte being transferred simultaneously.

Special input/output devices are available for handling parallel data. These are known as Parallel Interface Adapters (PIA) or Parallel Input/Output (PIO). The use of parallel transmission does, however, have the disadvantage of greater hardware costs, especially if peripheral devices are positioned at locations distant from the computer system.

9.2 Serial Input/Output

Serial data communication between devices greatly reduces the number of connections required, leading to simpler circuit layouts and economies in connector size. Obviously, transferring one bit at a time makes serial I/O slower.

A serial I/O interface can be used to eliminate the heavy processing load imposed upon a normal computer performing serial data transfer. Whereas a normal computer must regularly monitor the serial dat bus, the serial I/O detects, receives and converts the data stream to parallel without interrupting the execution of the current program.

10.0 COMPUTER TIMING

The synchronization of functions in a computer is fundamental to its correct operation. Instructions must be fetched, acted upon and fetched continously until the program is complete. A precise timing reference is therefore required. A free running clock furnishes just such a reference for all processing actions. This clock is generated by an on-chip oscillator with XTAL or LC combination providing the feedback and phase-shift required for oscillation.

The use of a clock as reference means that each instruction carried out by the computer takes a specified number of instruction cycles. The clock defines the state times of the microcontroller. the state times are then divided into machine cycles. In figure 2 below, an MAB8048 instruction cycle is shown as consisting of two machine cycles each of five states. A machine cycle is of two parts; a fetch phase and an execute phase.

During the fetch phase, the address of the instruction to be executed is taken from the program counter (see 7.3) and placed on the address bus . A memory location is located by this address, and the instruction/data therein is placed in the instruction register via the data bus . The program counter is then incremented and will point to the next location.

During the execute phase, the instruction is acted upon by the controller in the manner chosen by the programmer. Usually this instruction (unless it is an implied instruction, single byte) will direct the controller to obtain data from memory which will be contained in, or pointed to, by the next byte.

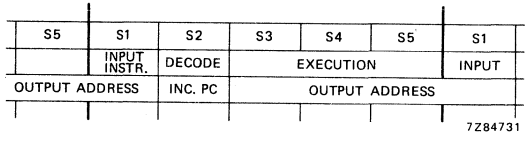


Fig. 2 An 8048 instruction cycle

10.1 Timer/Counter

A timer/counter aids the user in counting external and internal events and generating accurate time delays without placing a burden on the processor. In both modes, counter operation is the same, the only difference is the source to the counter.

11.0 MICROCOMPUTER PROGRAMMING

In general, what we have been dealing with up to now is the hardware of a microcontroller system. We shall now look at the software - the instructions which enable the hardware to do something useful.

To carry out the functions required of it, each instruction in program memory must be in a form that the processor understands. For most microcontrollers, instructions are made up of eight binary 1s and 0s. This is intelligable to the controller, but a list of such instructions neither helps the reader to understand the program nor the programmer to correct mistakes. One answer to writing a large number of 1s and 0s is to use a hexadecimal number base. Hexadecimal is a means of writing numbers to base 16, instead of, in the case of binary, base 2 and is very easy to convert to and from binary. If you consider the binary number:-

1 1 0 1 0 0 1 1

It looks sligthly daunting to convert, but if you split it up into two groups of four and each part looked up in the conversion table below, it can be seen

1 1 0 1 0 0 1 1

D 3

that the number converts easily to the hexadecimal number D3.

Decimal	Hexadecimal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Although hexadecimal (or 'hex' for short) reduces the possibility of error when writing programs it does not make them easier to understand. The use of Assembly Languages overcomes this disadvantage by allowing the programmer to use alphanumeric symbols to represent machine code. For example, adding the contents of register 1 to the accumulator becomes ADD A,R1 instead of 'hex' 69.

Assembly languages differ with the type of microcontroller used. But for each type, there is a specification of commands, their actions, the registers they use and the flag bits they affect. This specification is called the Instruction Set for the microcontroller and is a condensed summary of all actions that are possible with that particular microcontroller.

The example program given below demonstrates the use of Assembly Language and how it enables programs to be both understandable and concise. The purpose of the program is to set Register 1 to zero, read a value from Port 1 into the Accumulator, add the contents of Register 1 to the Accumulator and store the results in Register 1 again. The procedure is repeated until the value read from Port 1 = 0, then the sum of all values read, stored in Register 1, is output to Port 2. The binary and hexadecimal codes for this program are also included to give a direct comparison.

Step Number	Assembly Language		hex	binary
0		MOV R1,#0	B9	1011 1001
1			00	0000 0000
2	START	IN A,P1	09	0000 1001
3		JZ STOP	C6	1100 0110
4			09	0000 1001
5		ADD A,R1	69	0110 1001
6		MOV R1,A	A9	1010 1001
7		JMP START	04	0000 0100
8			02	0000 0010
9	STOP	MOV A,R1	69	0110 1001
A		OUTL P2,A	3A	0011 1010

Note that two consecutive bytes of memory are used by the program when transferring data or specifying a branch address.

Example: When transferring data to memory the first byte is the instruction or opcode, and the second byte the data. In todays microcontroller the address is very often 16 bits wide, hence it is not uncommon to find instructions which are three bytes long.

The first instruction, MOV R1,#0, can be explained as: Clear register 1 using the value 0. Move (MOV) instructions for the MAB8400 and MAB8048 families are always structured with the destination first and the source second. The hash sign '#' indicates that the source is 'immediate' data (contained in the next byte of program memory).

The input instruction IN A,P1 is similar to a MOV instruction in that it indicates that the contents of Port 1 are to be transferred to the accumulator.To the left of the input instruction is an address label separated from the instruction by a colon. A label allows a program to be written independent of its final situation in program memory, since the jump instruction at the end of the program can refer to this label rather than a specific address. For more program examples see the separate software examples section of this manual.

Elsewhere in this manual is an expanded instruction set which explains the mnemonics and the binary machine codes they represent. All the instructions used so far will also be found in that section.

Once a program has been written in Assembly Langauge, the problem exists of how to translate it into binary code. The answer is to use Assemblers. Assemblers are computer programs that interpret typed instructions and data and convert them into correct binary codes. When a program has been succesfully assembled, the first stage of software development can be considered complete.

Although programs may be correctly assembled, contain no syntax error and be structurally sound, they may still present problems when executed. The reason for such behaviour is that some hidden logic 'bugs' may remain and show up only when the hardware and software are integrated and operate in 'real-time'. In order to iron-out such bugs, the system must be run and checked at normal 'real-time' speeds. This is known as 'real-time debugging'.

When using real-time debugging, emulation techniques are required. Instead of the microcontroller being 'simulated' by a much larger computer when tests are carried out, an identical microcontroller is used within the test system itself. The emulation programmer must remember to tailor the program so that no address higher than the highest hardware address in ROM or RAM is used.

In-circuit emulation and debugging are the prime functions of a Microcontroller Development System. By using the benefits offered by such a system, access can be gained to all internal functions of the microcontroller in the prototype system and software can be tested in the hardware environment for which it was written. More detailed information on Software Development Support can be found in the Test and Development section of this manual.

2. The MAB8048/80C49 microcontroller family

CONTENTS – THE MAB8048/80C49 MICROCONTROLLER FAMILY

		page
1.0	DESCRIPTION	2-5
2.0	FEATURES	2-6
3.0	FUNCTIONAL DESCRIPTION	2-12
3.1	Program memory (ROM)	2-12
3.2	Data memory (RAM)	2-14
3.3	Input/Output	2-14
3.3.1	Ports 1 and 2	2-14
3.3.2	BUS	2-16
3.4	Interrupt	2-16
3.4.1	Interrupt timing	2-17
3.5	Timer/event counter	2-17
3.5.1	Counter operation	2-17
3.5.2	Event counter operation	2-19
3.5.3	Timer operation	2-19
3.6	Program status word	2-20
3.7	Program counter and stack	2-21
3.8	External access mode	2-22
3.9	Oscillator and clock	2-22
3.9.1	State counter	2-22
3.9.2	Cycle counter	2-22
3.10	Central processing unit	2-23
3.11	Test (T0, T1) and $\overline{INT}$ inputs	2-24
3.12	RESET input	2-24
3.13	Power-down mode	2-25
3.13.1	Power-down sequence	2-25
3.14	Idle mode (PCB80C49, PCB80C39)	2-26
3.15	Single step input ($\overline{SS}$)	2-26
3.15.1	Single step timing	2-27
3.16	Instruction set	2-27

		page
4.0	EXPANDED 8048/80C49 FAMILY SYSTEM	2-32
4.1	Expansion of program memory	2-32
4.1.1	Instruction fetch cycle (external)	2-32
4.1.2	Extended program memory addressing (beyond 2 K bytes)	2-33
4.1.2.1	Program memory bank switching	2-33
4.1.2.2	Interrupt routines	2-33
4.1.3	Restoring I/O port information	2-34
4.1.4	Expansion example	2-34
4.2	Expansion of data memory	2-34
4.2.1	Read/write cycle	2-34
4.2.2	Addressing external data memory	2-35
4.2.3	Data memory expansion example	2-35
4.3	Expansion of input/output	2-35
4.3.1	I/O expander device	2-36
4.4	Memory bank switching	2-37
4.5	Control signal summary	2-37

1.0 DESCRIPTION

Within this section, the characteristics of the 8048/80C49 family of single-chip microcontrollers are described in detail. The 8048/80C49 family consists of:

MAB8048H - single-chip microcontroller
MAB8035HL - ROMless version of MAB8048H

MAB8049H - single-chip microcontroller
MAB8039HL - ROMless version of MAB8049H

MAB8050H - single-chip microcontroller
MAB8040HL - ROMless version of MAB8050H

PCB80C49 - MAB8049 n-well CMOS version
PCB80C39 - MAB8039 n-well CMOS version

Unless otherwise stated, details apply to all versions. This description uses the MAB8048H as the representative n-MOS product within the family, the PCB80C49 will be used as the representative n-well CMOS product.

The devices which make up the 8048/80C49 family are control processors as well as arithmetic processors. The 8048/80C49 family instruction set allows the user to set and reset individual I/O lines directly, as well as test individual bits within the accumulator. A large variety of branch and table look-up instructions enable implementation of standard logic functions. Code efficiency is high, over 70% of the instructions are single byte, all others are two byte.

An on-chip 8-bit counter is provided that counts either machine cycles ÷32, or external events. The counter can be programmed to cause an interrupt to the processor. Program and data memories can be expanded using standard devices (see section 4). Input/output capabilities can be expanded using standard devices or a specialized I/O expander.

2.0 FEATURES

MAB8048H/MAB8035HL
MAB8049H/MAB8039HL
MAB8050H/MAB8040HL

- 8-bit CPU
- MAB8048H: 1 K x 8 ROM, 64 x 8 RAM, 27 I/O lines
- MAB8049H: 2 K x 8 ROM, 128 x 8 RAM, 27 I/O lines
- MAB8050H: 4 K x 8 ROM, 256 x 8 RAM, 27 I/O lines
- Internal counter/timer
- Internal oscillator
- Single-level interrupts: external and counter/timer
- 17 internal registers: accumulator, 16 addressable registers
- Over 90 instructions: 70% single-byte
- All instructions: 1 or 2 cycles (1,36 usec instruction cycle @f_{osc}= 11 MHz)
- Easily expandable memory and I/O
- TTL compatible inputs and outputs
- Single 5 V supply
- Reduced power comsumption

PCB80C49/PCB80C39

As above, except:

- Pin and function compatible with:-
  MAB8049H/MAB8039HL
- Maintains operation during AC power line interruptions
- Power consumption selections
- Exit Idle mode with an external or internal interrupt signal
- Battery operation

PACKAGE OUTLINES

All versions : 40-lead DIL; plastic (SOT-129).
MAB8048H/35HLT : 40-lead mini-pack; plastic (VSO-40; SOT-158).
MAB80XXH/HLWP : 44-lead PLCC; plastic, leaded chip-carrier (SOT-187).
PCF80C39/C49WP : 44-lead PLCC; plastic, leaded chip-carrier (SOT-187A).

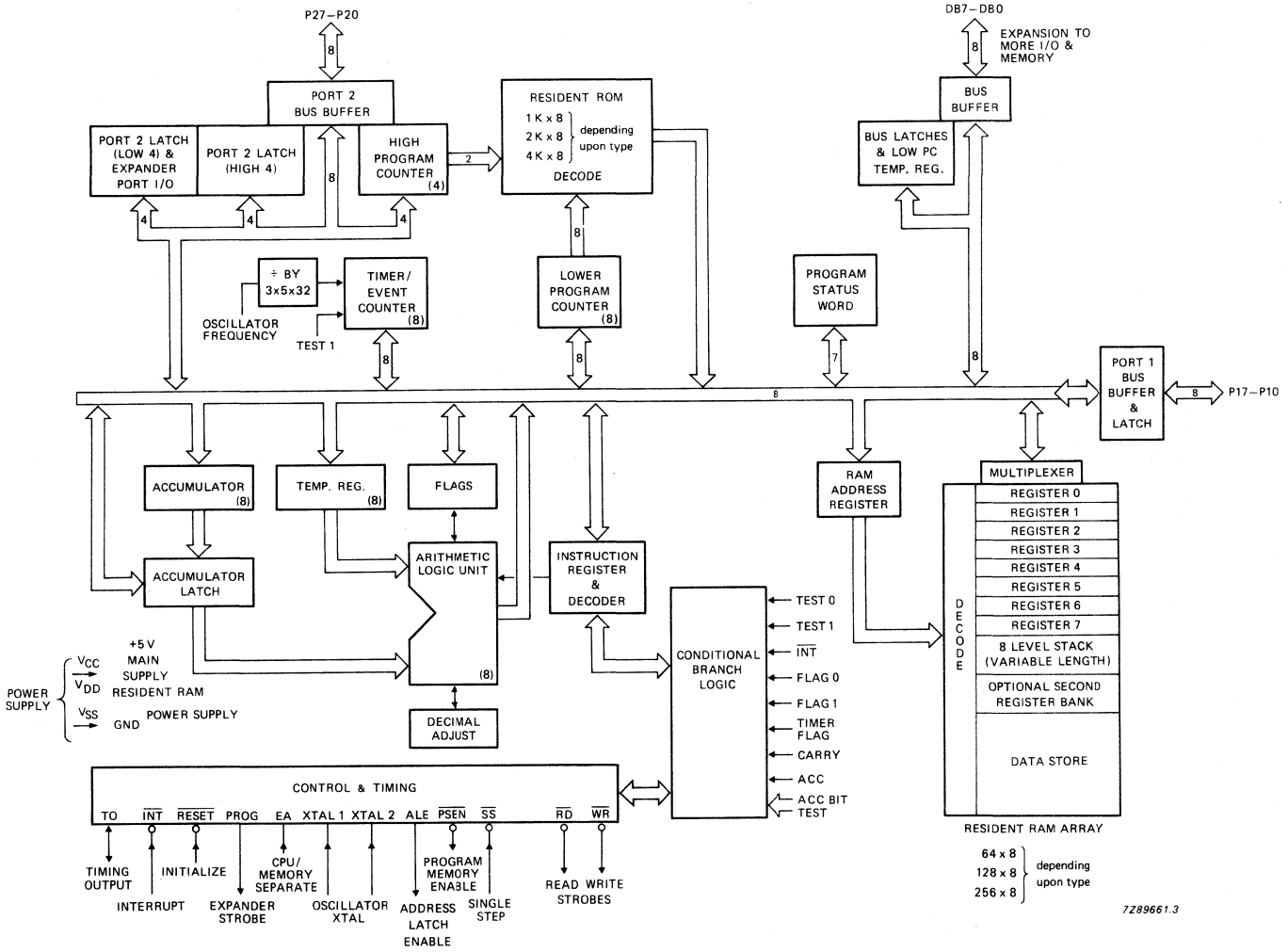


Fig. 2.1 Block diagram of the 8048/C49 family

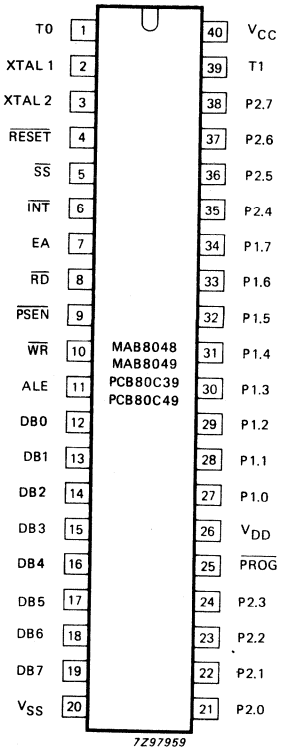
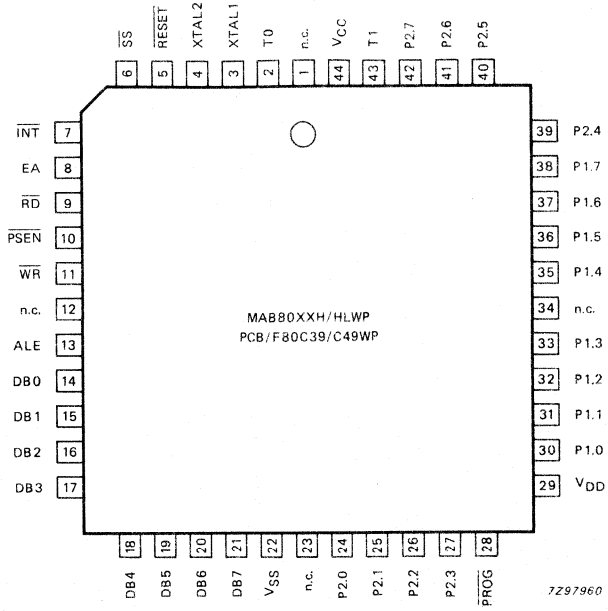


Fig. 2.2 Pinning diagram; for suffix 'P' DIL versions.



Where : n.c. = not connected.

Fig. 2.3 Pinning diagram for MAB/PCF80/80CXXH/HLWP.

12-19	DB0-DB7	Data Bus: bidirectional I/O port which write or read using the $\overline{RD}$ and $\overline{WR}$ strobes. This port can also be statically latched. It contains the 8 lower order address bits during external memory access and receives the addressed instruction under control of $\overline{PSEN}$. $\overline{PSEN}$, ALE, $\overline{RD}$ and $\overline{WR}$ determine whether the access is an instruction fetch or a read/write access to external RAM.
27-34	P10-P17	Port 1: 8-bit quasi-bidirectional I/O port (note 1).
21-24	P20-P27	Port 2: 8-bit quasi-bidirectional I/O port (note 1).
35-38		P20-P23 contain the 4 higher order address bits during an access of external program memory.
25	$\overline{PROG}$	Output strobe: active LOW for 8243 I/O expander.
1	T0	Test 0: test input pin sensed using the JT0 and JNT0 instructions. Clock: clock output pin when designated by the ENT0 CLK instruction.
39	T1	Test 1: test input pin sensed using the JT1 and JNT1 instructions. Can be designated as the timer/counter input by the STRT CNT instruction.
6	$\overline{INT}$	Interrupt: interrupt input pin, which causes an interrupt in the current program, provided that the external interrupt is enabled. Can also be used as an input, testable using the JNI instruction. Interrupt is disabled during and after $\overline{RESET}$.
4	$\overline{RESET}$	Reset: active LOW input used to initialize the controller. During program verification the address is latched by a '0' to '1' transition on $\overline{RESET}$ and the data at the addressed location is output on BUS (note 2).
11	ALE	Address latch enable: occurs each cycle and is used for timing and sampling. During external program or data memory access, ALE is used to strobe the address information multiplexed on the DB0 to DB7 outputs.
8	$\overline{RD}$	Read BUS: active LOW strobe used to gate data onto BUS lines when reading from an external source.

10	$\overline{WR}$	Write BUS: active LOW strobe used to write data from BUS lines to an external designation.
7	EA	External access input: when HIGH, forces instruction fetch from external memory.
9	$\overline{PSEN}$	Program store enable: active LOW strobe that occurs only during a fetch from external memory.
5	$\overline{SS}$	Single step: active LOW input used with ALE to cause the microcontroller to execute a single instruction.
2	XTAL 1	Crystal inputs: inputs for a crystal, LC-network or an external timing signal to determine the internal oscillator frequency (note 2).
3	XTAL 2	
20	V_{SS}	Ground: circuit earth potential.
40	V_{CC}	Power supply: + 5 V main power supply pin.
26	V_{DD}	Power supply: + 5 V RAM standby power supply; low power standby pin.

Notes

1. Each port line can be designated as an input or an output. A line is designated as an input by first writing a logic 1 to the line. $\overline{RESET}$ sets all lines to logic 1.
2. Non-standard TTL V_{IH}.

3.0 FUNCTIONAL DESCRIPTION

3.1 Program memory (ROM)

The resident program memory comprises 1024, 2048 or 4096 bytes of ROM, the contents of which are addressed by the program counter; the MAB8035HL/MAB8039HL/MAB8040HL have no resident program memory. The total addressing capability is 4096 bytes. Program code is totally compatible between the various versions.

The program memory address space is divided into two 2048-byte banks MB0 and MB1 (Fig. 3.1). Further, the program memory is divided into pages of 256 bytes. This latter division applies only for conditional branches. To access the upper 2 K of program memory in the 8050H, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2 K boundary.

There are three locations in program memory of special importance. These locations contain the first instruction to be executed after one of three events.

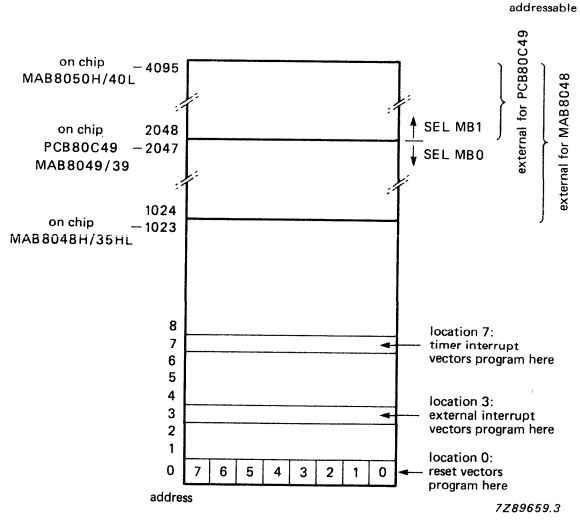


Fig. 3.1 Program memory map.

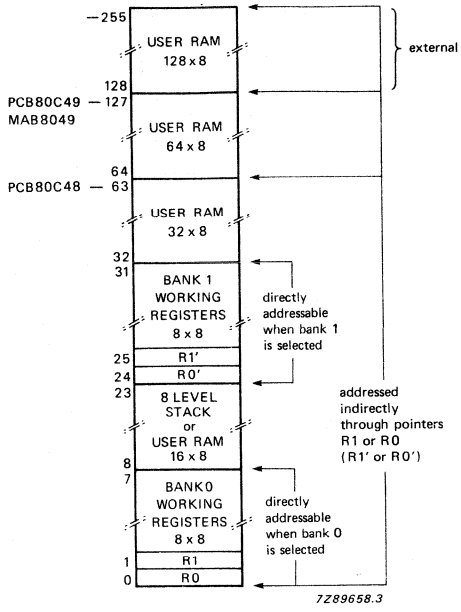


Fig. 3.2 Data memory map.

The three locations and their contents are:

location 0 - activating the $\overline{\text{RESET}}$ line of the processor causes the first instruction to be fetched from location 0.

location 3 - activating the Interrupt line ($\overline{\text{INT}}$) of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

location 7 - a timer/counter interrupt resulting from timer/counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first byte of an external interrupt service subroutine is stored in location 3, and the first byte of a timer/counter service routine is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "look-up" tables.

3.2 Data memory (RAM)

Resident memory is organised as 64, 128, or 256 bytes in the MAB8048H, MAB8049H and MAB8050H respectively. All locations are indirectly addressable by two RAM Pointer Registers at locations 0 and 1. The first eight locations of RAM (0 to 7) are designated as working registers and are directly addressable by several instructions.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24-31 are designated as the directly addressable working registers in place of locations 0-7. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines, this allows the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note, that if this second bank is not used, locations 24-31 are still addressable as general pupose RAM. Since the two RAM Pointer Registers R0 and R1 are a part of the working register array, bank switches effectively create two more pointer registers (R0' and R1') which can be used with R0 and R1 to access up to four separate working areas in RAM.

RAM locations 8 to 23 are designated as the stack. Two locations (bytes) are used per CALL, allowing up to eight levels of subroutine nesting. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting in less than 8, unused stack registers may be used as general purpose RAM Locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

If additional RAM is required, up to 256 bytes may be externally added and directly addressed using the MOVX instructions. If more RAM (greater than 256 bytes total) is required, an I/O line can be used to select one bank (256 bytes) of external memory at a time.

3.3 Input/Output

The MAB8048H family has 27 I/O lines. These lines are arranged as three 8-bit ports, which serve as either inputs, outputs or bidirectional ports, and three 'test' inputs which can alter program sequences when tested by conditional jump instructions.

3.3.1 Ports 1 and 2

Ports 1 and 2 are each 8-bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports, these lines are non-latching, i.e. inputs must be present until read by an input instruction. Inputs are fully TTL-compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, an output, or both even though outputs are statically latched. The circuit configuration for the MAB8048 is shown in figure 3.3. Each line is continuously pulled up to +5 V through a relatively high resistance (50 kΩ). This pull-up is sufficient to provide the source current for a TTL HIGH level, yet can be pulled LOW by a standard TTL gate, thus allowing the same pin to be used for both input and output.

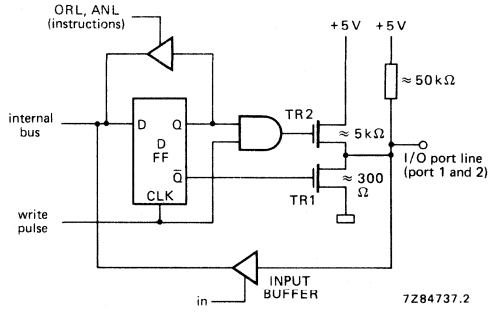


Fig. 3.3 Quasi-bidirectional port structure of MAB8048 family.

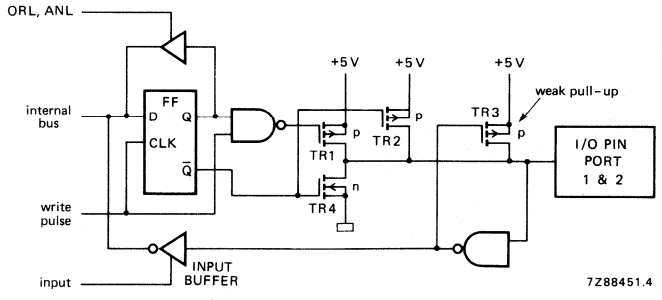


Fig. 3.4 Quasi-bidirectional port structure of PCB80C49 family.

To provide fast switching times for the MAB8048, during 0 to 1 transitions a relatively low-impedance transistor (5 kΩ) is switched on momentarily for 1/5 of a machine cycle (by the write pulse) whenever a '1' is written to the line. When a '0' is written, a low-impedance transistor (3000) pulls the output low and provides TTL current sinking capability. (Fig. 3.3)

The circuit configuration for the PCB80C49 is shown in figure 3.4. Each line has a unique high-impedance pull-up transistor TR3, this is turned on when the line is pulled above 2 V by an external source or by writing a logic 1 to the port. This pull-up is sufficient to provide the source current for a TTL HIGH level, yet can be pulled LOW by a standard TTL gate, thus allowing the pin to be used for both input and output. When a logic 1 is written to a line a second high impedance transistor TR2, pulls the line up to 5 V. To provide fast switching during a '0' to '1' transition, a relativly low-impedance transistor TR1(approx. 750 Ω) is switched on for 1/5 of a machine cycle whenever a 1 is written to the line. When a '0' is written to the line, a low-impedance (approx. 250 Ω) transistor TR4, pulls the output low and provides TTL current sinking capability.

Since the pull-down transistor TR4 is a low impedance device, a 1 must be written to any line which is to be used as an input. RESET initializes all the port lines to the high-impedance '1' state *. This structure allows input and output on the same pin and also allows a mixture of input lines and output lines on the same port.

3.3.2 BUS

BUS is also a true bidirectional 8-bit port with associated input and output strobes. If the bidirectional feature is not needed, BUS can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed.

As a static port, data is written and latched using the OUTL instruction and input using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding $\overline{RD}$ and $\overline{WR}$ output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port, the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the $\overline{WR}$ output line and output data is valid at the trailing edge of $\overline{WR}$. A read of the port generates a pulse on the $\overline{RD}$ output line and input data must be valid at the trailing edge of $\overline{RD}$. When not being written or read, the BUS lines are high impedance.

3.4 Interrupt

An interrupt may be generated by either an external input ($\overline{INT}$, pin 6) or the overflow of the internal counter, when enabled. In either case, the processor completes execution of the present instruction and then does a CALL to the interrupt service routine. After service, a RETR instruction restores the machine to the state it was prior to the interrupt. The external interrupt has priority over an internal interrupt.

* It is important to note that any output operation on a port (ORL, ANL or OUTL) will activate the low-impedance device on all pins of the port. If the result leaves the port pin in the high impedance 1 state, the low-impedance pull-up device will always turn on momentarily. This specifically applies to configurations that have inputs and outputs mixed together on the same port.

An interrupt sequence is initiated by applying a LOW level pulse to the $\overline{INT}$ pin. Interrupt is level triggered and active LOW to allow "wired-ORing" of several interrupt sources at the input pin. The interrupt line is sampled every instruction cycle and when detected causes a "jump to subroutine" at location 3 in program memory, as soon as the current instruction is completed. For 2 cycle instructions, the interrupt line is sampled on the second cycle only. The $\overline{INT}$ signal must be held LOW for at least 3 machine cycles to ensure correct interrupt operation. As in any CALL to subroutine, the Program Counter and Program Status Word are saved in the stack. For a describtion of this operation see section 3.7. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory.

The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR re-enables the interrupt input logic (Fig. 3.5). This occurs at the beginning of the second cycle of the RETR instruction. The sequence also holds true for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (one less than terminal count) and enabling the event counter mode. A '1' to '0' transition on the T1 input will then cause an interrupt vector to location 7.

3.4.1 Interrupt timing

The interrupt may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the user's program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the MAB8048H may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The INT pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled, $\overline{INT}$ may be used as another test input such as T0 and T1.

3.5 Timer/event counter

The MAB8048 contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions (Fig. 3.6). In both modes the counter operation is the same, the only difference being the source of the input to the counter.

3.5.1 Counter operation

The 8-bit binary counter is presetable and readable with two MOV instructions, these transfer the contents of the accumulator to the counter and vice versa. The counter should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once

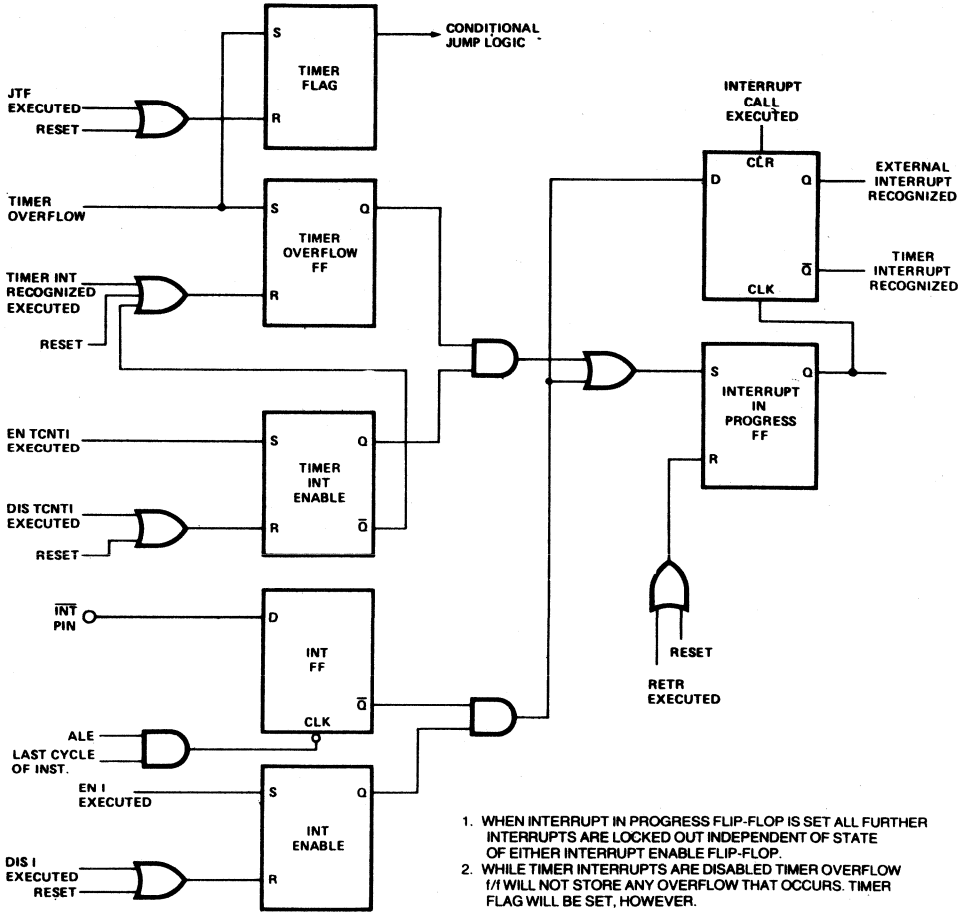


Fig. 3.5 The interrupt logic

started the counter will increment to this maximum count (FF) and overflow to zero, continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNTI and DIS TCNTI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If both timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and will immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TNCTI instruction.

3.5.2 Event counter operation

Execution of a STRT CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3, or in later MAB8048H devices in state 4. Subsequent HIGH to LOW transitions on T1 will cause the counter to increment. The maximum rate at which the counter may be incremented is once per three machine cycles - there is no minimum frequency. T1 must be held LOW for at least 1 machine cycle to ensure it won't be missed. T1 input must remain HIGH for at least 1/5 machine cycle after each transition.

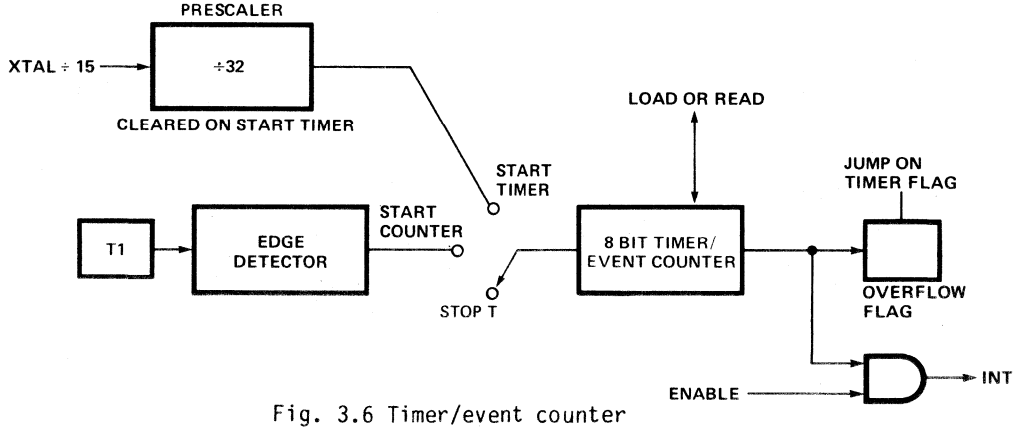


Fig. 3.6 Timer/event counter

3.5.3 Timer operation

Execution of a STRT T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived by passing the basic machine cycle clock ALE through a -32 prescaler. The prescaler is reset during the STRT T instruction.

The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be obtained by acumulating multiple overflows in a register under software control. For time resolution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very short delays or "fine tuning" of larger delays can be easily accomplished in software delay loops.

3.6 Program status word

The program status word (PSW) is an 8-bit word that can be loaded to and from the accumulator. Figure 3.7 shows the information available in the word. The program status word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to the PSW allows for easy restoration of machine status after a power-down sequence.

The upper four bits of PSW are stored in the program counter stack with every call to subroutine or interrupt vector, and are optionally restored upon return with the RETR instruction. The RET instruction does not update the PSW.

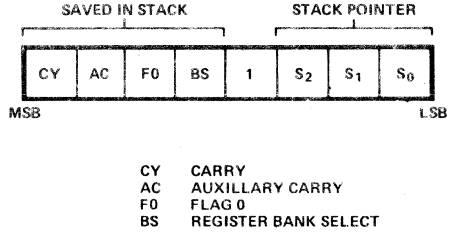


Fig. 3.7 Program status word (PSW).

The PSW bit definitions are as follows:

Bits 0-2 : Stack Pointer bits (S_0, S_1, S_2)

Bit 3 : Not used ('1' level when read)

Bit 4 : Working Register Bank Switch Bit (BS)
0 = Bank 0
1 = Bank 1

Bit 5 : Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.

Bit 6 : Auxilliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.

Bit 7 : Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

3.7 Program counter and stack

The program counter (PC) is a 12-bit counter/register that points to the location from which the next instruction is to be fetched (Fig. 3.8). When EA is '0', the PC can address locations 0 to 1023 of internal program memory. At the 1 K boundary, an automatic switch-over to the external memory is made (for the MAB8049H this occurs at the 2 K boundary, for the MAB8050H there is no automatic switch over to external program memory). When EA = 1, all the program is fetched from external ROM/EPROM. The total address range is 4 K bytes. The program counter is initialized to zero by activating the $\overline{\text{RESET}}$ line.

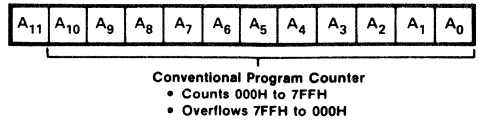


Fig. 3.8 Program counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the program counter stack (Fig. 3.9). The pair to be used is determined by a 3-bit stack pointer which is part of the program status word (PSW). Data RAM locations 8 to 23 are available as stack registers and are used to store the program counter and 4 bits of PSW. The stack pointer, when initialized to 000, points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL.

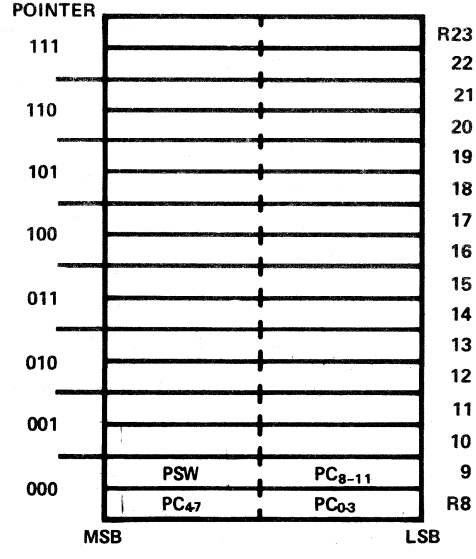


Fig. 3.9 Program counter stack.

Nesting of subroutine within subroutine can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (location 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000; it also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the stack pointer to be decremented and the contents of the resulting register pair to be transferred to the program counter.

3.8 External access mode

Normally the first 1 K (MAB8048H), 2 K (MAB8049H), or 4 K (MAB8050H) bytes of program memory are automatically fetched from internal ROM. The EA input pin however, allows the user to disable internal program memory by forcing all program memory fetches to reference external memory. Chapter 4.1 explains how external program memory is accessed.

The External Access mode is very useful in system test and debug because it allows users to disable their internal applications program and substitute an external program of their choice - a diagnostic routine for instance! Section 4.11 explains how internal program memory can be read externally, independent of the processor. A '1' level on EA initiates the external access mode. For correct operation, $\overline{\text{RESET}}$ should be applied while the EA input is changed.

3.9 Oscillator and clock

The MAB8048H contains its own internal oscillator. A crystal, inductor or external pulse generator determines the oscillator frequency. The output of the oscillator is divided-by-3 and is available at T0 (pin 1) by executing the ENT0 CLK instruction. This CLK signal is divided-by-5 to define a machine cycle. It is available at the ALE pin. For more details see the clock section in the data sheet at the end of this descriptive section.

3.9.1 State counter

The output of the oscillator is divided-by-3 in the State counter (Fig. 3.10) to create a clock that defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENT0 CLK instruction. The output of CLK to T0 is disabled by a Reset.

3.9.2 Cycle counter

CLK is then divided-by-5 in the Cycle counter to provide a clock that defines a machine cycle consisting of 5 machine states. At every third machine state an address valid pulse (Address Latch Enable) is available at the pin ALE. External memory or peripheral address data may be latched on the falling edge of this pulse.

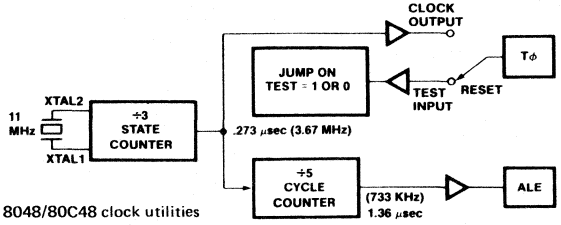
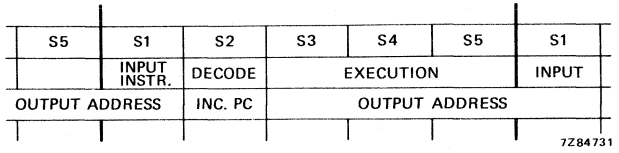
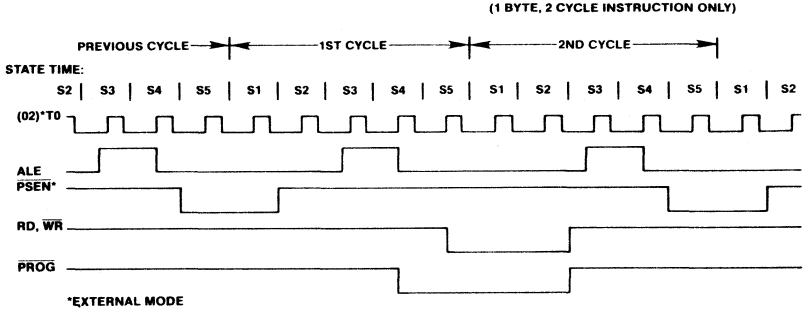


Diagram of 8048/80C48 clock utilities



Instruction cycle



Instruction cycle timing

Fig. 3.10 Clock utilities.

3.10 Central processing unit

The 8048/80C48 family central processing unit can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the Arithmetic Logic Unit (ALU) results in a value represented by more than 9 bits (8 bit value plus overflow), a Carry Flag is set in the program status word.

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the user's program. By using the conditional jump instruction the following conditions can effect a change in the sequence of the program execution.

Device Testable	Jump Conditions (Jump on)	
Accumulator	All zeros	Not all zeros
Accumulator	-	1
Carry Flag	0	1
User Flags (F0.F1)	-	1
Timer Overflow Flag	-	1
Test Inputs (T0,T1)	0	1
Interrupt Input ($\overline{INT}$)	0	-

3.11 Test (T0, T1) and $\overline{INT}$ inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These pins are T0, T1 and $\overline{INT}$ and they allow inputs to cause program branches without having to load an input port into the accumulator. The T0, T1 and $\overline{INT}$ pins have other functions as well.

3.12 $\overline{RESET}$ input

The $\overline{RESET}$ input provides a means of initializing the processor. This Schmitt-trigger input has an internal pull-up resistor which, in combination with an external 1,0 µF capacitor, provides an internal reset pulse of sufficient duration to guarantee all circuitry is reset. If the reset pulse is generated externally, the RESET pin must be held at ground (0,45 V) for at least 10 ms after the power supply has stabilized. Only five machine cycles are required if power is already on and the oscillator has also stabilized. Typical circuitry is shown in figure 3.11.

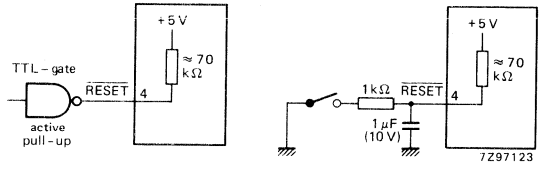


Fig. 3.11 Reset circuitry.

Reset performs the following functions:

1. Sets program counter to zero
2. Sets stack pointer to zero
3. Selects register bank 0
4. Selects memory bank 0
5. Sets Bus to HIGH impedance state (except when EA=5V)
6. Sets Ports 1 and 2 to input modes
7. Disables interrupts (timer and external)
8. Stops timer
9. Clears timer flag
10. Clears F0 and F1
11. Disables clock output from T0

Note: Reset does not disturb the contents of the accumulator or RAM locations

3.13 Power-down mode

Power can be removed from all but the 64 byte data RAM array for low power standby operation. In the power-down mode the contents of the data RAM are be maintained while drawing typically 10% to 15% of the normal operating power. V_{CC} serves as the + 5 V supply pin for the bulk of the circuitry, while the V_{DD} pin supplies only the RAM array. In normal operation, both pins are at + 5 V. In standby, V_{CC} is at ground and only V_{DD} is maintained at + 5 V.

3.13.1 Power-down sequence

A typical power-down sequence occurs as follows:

1. Imminent power supply failure is detected by user-defined circuitry. The signal must be early enough to allow the microcomputer to save all necessay data before V_{CC} falls below normal operating limits.

2. The power fail signal is used to interrupt the processor and vector it to a power fail service routine.

3. The power fail routine saves all important data and machine status in the internal data RAM array. The routine may also initiate a transfer of the backup supply to the V_{DD}pin and indicate to external circuitry that the power fail routine is complete.

4. A $\overline{\text{RESET}}$ is applied to guarantee that data will not be altered as the Power supply falls out of limits. $\overline{\text{RESET}}$ must be held LOW until V_{CC} is at ground level.

Recovery from the power-down mode follows any other power-on sequence with an external capacitor on the $\overline{\text{RESET}}$ input providing the necessary delay. See section 4.10 on $\overline{\text{RESET}}$. Applying a reset to the processor through the $\overline{\text{RESET}}$ pin inhibits any access to the RAM by the processor and guarantees that the RAM cannot be inadvertently altered as power is removed from V_{CC}. A typical power-down sequence occurs as shown in figure 3.12.

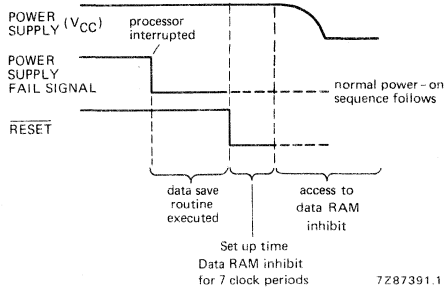


Fig. 3.12 Power-down sequence.

3.14 Idle mode

(PCB80C49, PCB80C39)

The PCB80C49, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the status of the following:

- RAM and register
- Ports 1 and 2
- BUS

To place the PCB80C49 in Idle mode, a command instruction (opcode '01') is executed and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. A reset signal will also take the processor out of Idle mode. During the Idle mode, the standard power-down mode is still maintained.

Added instruction

Mnemonic	Description	Byte	Cycles
IDL	Select Idle operation	1	1

3.15 Single step input ($\overline{SS}$)

This feature provides the user with the debug capability to step through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available simultaneously on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps.

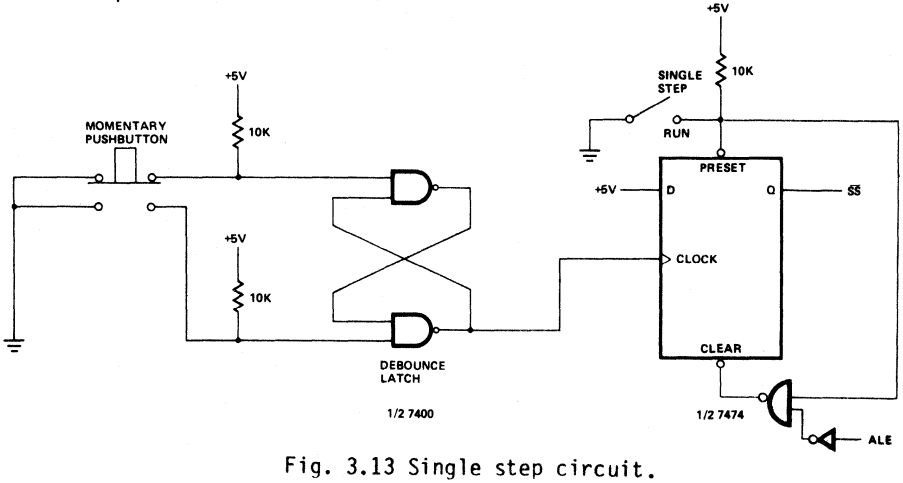


Fig. 3.13 Single step circuit.

3.15.1 Single step timing

The MAB8048/PCB80C49 operates in a single-step mode as follows:

1. The processor is requested to stop by applying a LOW to SS.

2. The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.

3. The processor acknowledges it has entered the stopped state by raising ALE HIGH. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.

4. SS is then raised HIGH to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE LOW.

5. To stop the processor at the next instruction SS must be brought LOW again soon after ALE goes LOW. If SS is left HIGH the processor remains in a "Run" mode.

A diagram for implementing the single-step function is shown in figure 3.13. A D-type flip-flop with preset and clear is used to generate SS. In the run mode SS is held HIGH by keeping the flip-flop preset (preset has precedence over the clear input). To enter single-step, preset is removed allowing ALE to bring SS LOW via the clear input. The processor is now in the stopped state. The next instruction is initiated by clocking a '1' into the flip-flop. This '1' will not appear on SS unless ALE is HIGH removing clear from the flip-flop. In response to SS going HIGH the processor begins an instruction fetch bringing ALE LOW, resetting SS through the clear input and causing the processor to again enter the stopped state.

3.16 Instruction set

For more detailed description of each instruction in the 8048/80C49 family, see the separate Instruction Set section of this Single-Chip Microcomputer Users Manual.

The 8048/80C49 family instruction set includes over 90 one and two-byte instructions (see Table 1). Program code efficiency is high because:

- working registers and program variables are stored in the RAM locations 0 to 63, -these require only a single byte to address

- program memory is divided into pages of 256 bytes, which means that branch destination addresses require only one byte.

The instruction set manipulates and tests bits as well as performing logical and arithmetic operations. A set of MOV instructions operate indirectly upon either RAM or ROM, and permit access to pointers and data tables. Within a 256 byte page, the indirect jump instruction performs a multi-way branch upon the content of the accumulator. The contents of the accumulator point to a location in program memory which contains the jump address. The "decrement register and jump if not zero" instruction economises on a byte every time it is used, when compared to using separate increment and test instructions.

The on-chip counter enables either external events or time to be counted off-line from the main program. The 8048/80C49 family can either test the counter (under program control) or cause its overflow to generate an interrupt. These features are highly desirable for real-time applications. See Table 1 for instruction timing.

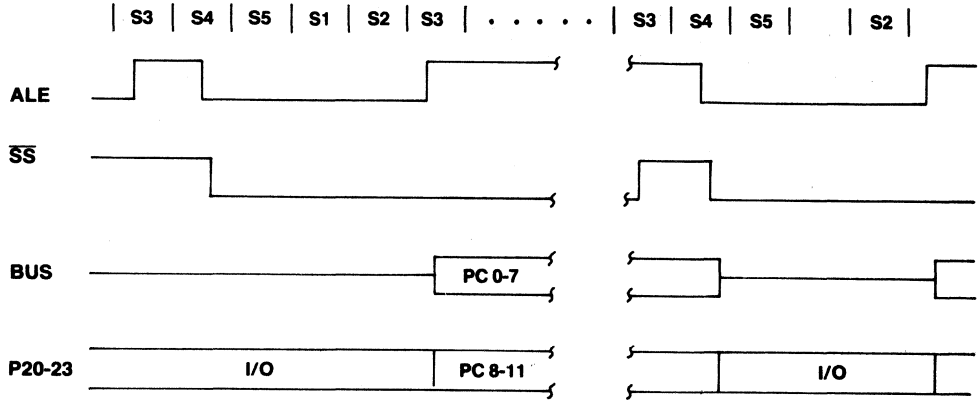


Fig. 3.14 Single step timing.

INSTRUCTION SET SUMMARY

Table 1

emonic	Description	Bytes	Cycle
cumulator			
D A,R	Add register to A	1	1
D A,@R	Add data memory to A	1	1
D A,#data	Add immediate to A	2	2
DC A,R	Add register with carry	1	1
DC A,@R	Add data memory with carry	1	1
DC A,#data	Add immediate with carry	2	2
L A,R	And register to A	1	1
L A,@R	And data memory to A	1	1
L A,#data	And immediate to A	2	2
L A,R	Or register to A	1	1
L A,@R	Or data memory to A	1	1
L A,#data	Or immediate to A	2	2
L A,R	Exclusive Or register to A	1	1
L A,@R	Exclusive Or data memory to A	1	1
L A,#data	Exclusive Or immediate to A	2	2
C A	Increment A	1	1
C A	Decrement A	1	1
R A	Clear A	1	1
L A	Complement A	1	1
A	Decimal Adjust A	1	1
AP A	Swap nibbles of A	1	1
A	Rotate A left	1	1
C A	Rotate A left through carry	1	1
A	Rotate A right	1	1
C A	Rotate A right through carry	1	1
put/Output			
A,P	Input port to A	1	1
TL P,A	Output A to port	1	2
L P,#data	And immediate to port	2	2
L P,#data	Or immediate to port	2	2
S A,BUS	Input BUS to A	1	2
TL BUS,A	Output A to BUS	1	2
L BUS,#data	And immediate to BUS	2	2
L BUS,#data	Or immediate to BUS	2	2
VD A,P	Input Expander port to A	1	2
VD P,A	Output A to Expander port	1	2
LD P,A	And A to Expander port	1	2
LD P,A	Or A to Expander port	1	2
gisters			
C R	Increment register	1	1
C @R	Increment data memory	1	1
C R	Decrement register	1	1

Mnemonic	Description	Bytes	Cycle
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R,addr	Decrement register and jump	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag = 1	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on Accumulator Bit	2	2
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear Flag 1	1	1
CPL F1	Complement Flag 1	1	1
Data Moves			
MOV A,R	Move register to A	1	1
MOV A,@R	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV R,A	Move A to register	1	1
MOV @R,A	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,R	Exchange A and register	1	1
XCH A,@R	Exchange A and data memory	1	1
XCHD A,@R	Exchange nibble of A and register	1	1
MOVX A,@R	Move external data memory to A	1	2
MOVX @R,A	Move A to external data memory	1	2
MOVP A,@A	Move to A from current page	1	2
MOVP3 A,@A	Move to A from Page 3	1	2

Mnemonic	Description	Bytes	Cycle
Timer/Counter			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control			
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENTO CLK	Enable Clock output on T0	1	1
NOP	No Operation	1	1
IDL	Select idle operation		

	CYCLE 1					CYCLE 2				
INSTRUCTION	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,P	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	—	—	READ PORT	—	• —	—
OUTL P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	OUTPUT TO PORT	—	—	—	• —	—
ANL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
INS A, BUS	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	—	—	READ PORT	—	• —	—
OUTL BUS, A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	OUTPUT TO PORT	—	—	—	• —	—
ANL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
MOVX @ R,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	OUTPUT DATA TO RAM	—	—	—	• —	—
MOVX A,@R	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	—	—	READ DATA	—	• —	—
MOVD A,P_i	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	—	—	READ P2 LOWER	—	• —	—
MOVD P_i,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA TO P2 LOWER	—	—	—	• —	—
ANLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	• —	—
ORLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	• —	—
J(CONDITIONAL)	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	SAMPLE CONDITION	*INCREMENT SAMPLE	—	FETCH IMMEDIATE DATA	—	UPDATE PROGRAM COUNTER	• —	—
STRT T STRT CNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	• —	START COUNTER					
STOP TCNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	• —	STOP COUNTER					
ENI	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	• ENABLE INTERRUPT	—					
DIS I	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	• DISABLE INTERRUPT	—					
ENTO CLK	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER		• ENABLE CLOCK	—					

*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.

Table 1 gives the instruction set of the MAB8048H. The following symbols and abbreviations are used.:

Symbol	description
A	accumulator
addr	program memory address
Bb	bit designation (b = 0-7)
RBS	register bank select
C	carry (bit CY)
CNT	event counter
D	mnemonic for 4-bit digit (nibble)
data	8-bit number or expression
I	interrupt
MB	memory bank
MBFF	memory bank flip-flop
P	mnemonic for 'in-page' operation
PC	program counter
Pp	port designation (p = 0,1,2)
PSW	program status word
RB	register bank
Rr	register designation (r = 0-7)
Sn	serial I/O register
SP	stack pointer
T	timer
TF	timer flag
T1	test 1 input
T0	test 0 input
#	immediate data prefix
@	indirect address prefix
(X)	contents of X
((X))	contents of location addressed by X
←	is replaces by
↔	is exchanged with

4.0 EXPANDED 8048/80C49 FAMILY SYSTEM

If the capabilities resident on-chip are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals. The processors can be directly expanded in the following areas:

- Program Memory to 4 K bytes
- Data Memory to 320 bytes (384 bytes with MAB8049H)
- I/O by an unlimited amount

By using bank switching techniques, maximum use of the addressing range is acheived. Bank switching is discussed in a subsequent section. Expansion is accomplished in two ways:

1. Expander I/O - A special I/O expander circuit provides the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple I/O expander chips may be added to this 4-bit bus by allocating the required "chip select" lines.

2. Standard Bus - One port of the MAB8048H/MAB8049H/50H/PCB80C49 is similar to the 8-bit bidirectional data bus of the Intel 8085 microprocessor, allowing interface to the numerous standard memories and peripherals.

Systems can be designed using either or both of these expansion features to fit system capabilities to the application. Expander devices, standard memories and peripherals can be added in virtually any number and combination required.

4.1 Expansion of program memory

Program Memory may be expanded beyond the resident 1 K or 2 K bytes by using the BUS feature of the 8048/80C49 family. All program memory access below addresses 1024 on the MAB8048H and below address 2048 on the MAB8049H/80C49, occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the MAB8048H/PCB80C49 (2048 in the 8049H/80C49), the processor automatically initiates external program memory fetch.

4.1.1 Instruction fetch cycle (external)

This is not included on the MAB8050H because of its 4 K byte program memory.

As shown in figure 4.1, for all instruction fetch from address 1024 (2048) or greater, the following will occur:

- The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.

- Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.

- Program Store Enable ($\overline{PSEN}$) indicates that an external instruction fetch is in progress and serves to enable the external memory device.

- BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

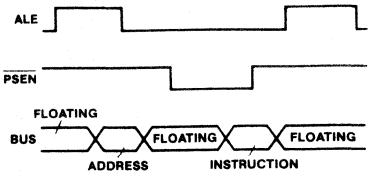


Fig. 4.1 Instruction fetch from external program memory.

All instruction fetches, including those from internal addresses, can be altered to external by activating the EA pin. The ROM-less versions without program memory mode (EA = 5 V).
For more detailed timing information, see the data sheet at the end of the user manual.

4.1.2 Extended program memory addressing (beyond 2 K bytes)

For programs of 2 K bytes or less, the MAB8048H/PCB80C49 family addresses program memory in the conventional manner. Addresses beyond 2047 may be reached by executing a memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2 K range and at the same time prevents the user from inadvertently crossing the 2 K boundary.

4.1.2.1 Program memory bank switching

The switching of 2 K program memory banks is brought about by directly setting or resetting the most significant bit of the program counter (bit 11). Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1 instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch, this occurs during the next branch instruction encounter. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a CALL is executed the user may jump to subroutines across the 2 K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

4.1.2.2 Interrupt routines

Interrupts always vector the program counter to location 3 or 7 in the first 2K bank, and bit 11 of the program counter is held at '0' during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be accommodated entirely in the lower 2 K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip-flop.

4.1.3 Restoring I/O port information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still output during certain portions of each machine cycle. I/O information is always present on Port 2's lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

4.1.4 Expansion example

Figure 4.2 shows the addition of 2 K bytes of program memory using an 2716A 2 K x 8 EPROM, expanding system memory to 4 K bytes. In this case no chip select decoding is required and $\overline{PSEN}$ enables the memory directly through the chip select input. If the system requires only 2 K of program memory, the same configuration can be used with an MAB8035HL substituted for the MAB8048H. The MAB8049H/80C49 would provide 4 K of program memory with the same configuration.

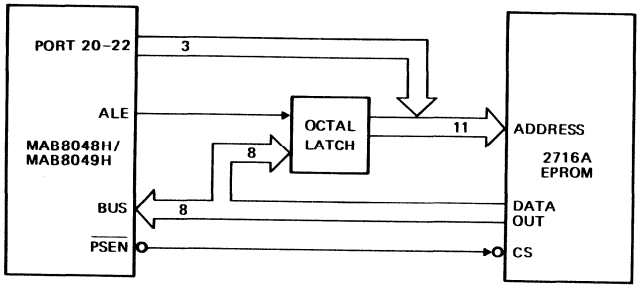


Fig. 4.2 Expanding program memory using standard memory components.

4.2 Expansion of data memory

Data memory is expanded beyond the resident 64 bytes by using the BUS feature of the 8048/80C49 family.

4.2.1 Read/write cycle

All address and data is transferred over the 8 lines of BUS. As shown in figure 4.3, a read or write cycle occurs as follows:

1. The contents of register R0 or R1 are output onto BUS.

2. Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.

3. A read ($\overline{RD}$) or write ($\overline{WR}$) pulse on the corresponding output pins of

   the MAB8048H indicates the type of data memory access in progress. Output data is valid at the trailing edge of $\overline{WR}$ and input data must be valid at the trailing edge of $\overline{RD}$.

4. Data (8 bits) is transferred in or out via BUS.

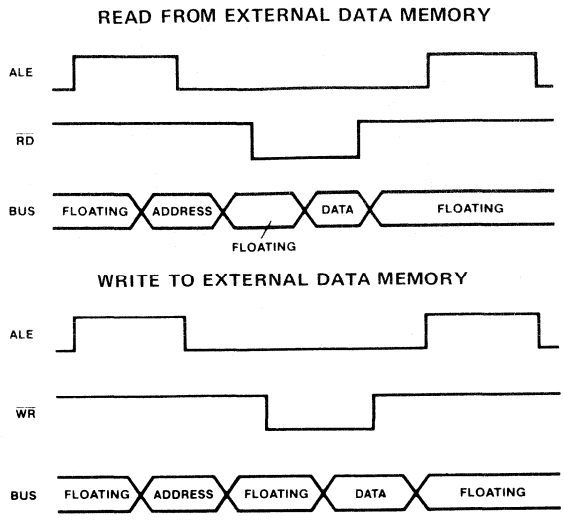


Fig. 4.3 External data memory timing.

4.2.2 Addressing external data memory

External data memory is accessed with two 2-cycle move instructions, MOVX A,@R and MOVX @R,A, these instructions transfer 8 bits of data between the accumulator and the external memory location via an address contained in one of the RAM Pointer Register R0 and R1. This permits an extra 256 RAM bytes to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the MAB8048H.

4.2.3 Data memory expansion example

Figure 4.4 shows how the MAB8048H can be expanded using a 256 x 8 standard memory.

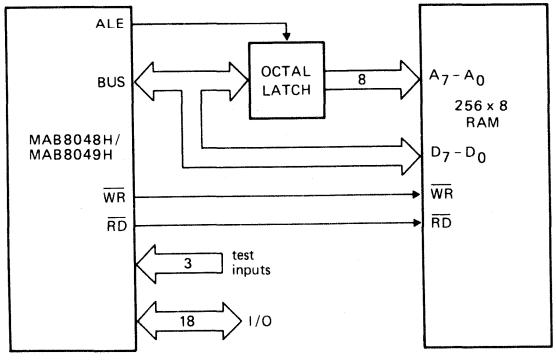


Fig. 4.4 MAB8048H to 256 x 8 standard memories.

4.3 Expansion of input/output

There are four possible modes of I/O expansion with the MAB8048H: one using a special expander; another using standard I/O devices; and a third using the combination memory I/O expander devices. It is also possible to expand using standard TTL devices.

4.3.1 I/O expander device

The most cost-effective means of I/O expansion for small systems is an I/O expander device (8243), this requires only 4 port lines (lower half of Port 2) for communication with the MAB8048H. This I/O expander contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and these are addressed as ports 4 - 7 (see Fig. 4.5). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

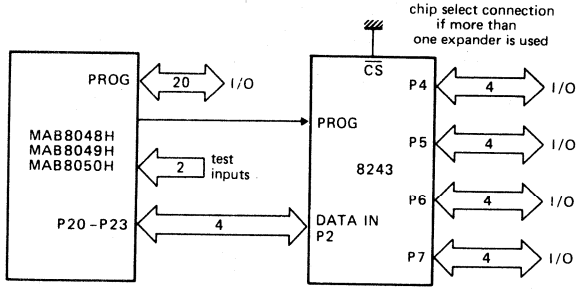


Fig. 4.5 Expander I/O interface.

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. All communication between the MAB8048H and the I/O expander occurs over Port 2 lower (P20-P23) with timing provided by an output pulse in the $\overline{PROG}$ pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing "op-code" and port address, and the second containing the actual 4 bits of data.

II	AA
$\overline{00}$ Read	$\overline{00}$-Port #4
01 Write	01-Port #5
10 OR	10-Port #6
11 AND	11-Port #7

A HIGH to LOW transition of the $\overline{PROG}$ line indicates that address data is present, while a LOW to HIGH transition indicates the presence of working data. Additional I/O expander devices may be added to the four-bit bus and chip select acheived using additional output lines from the MAB8048H.

4.4 Memory bank switching

Certain systems may require more than the 4 K bytes of program memory which are directly addressable by the program counter, or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be accommodated using "bank switching" techniques. Bank Switching is merely the selection of various blocks or "banks" of memory using dedicated output port lines from the processor. In the case of the MAB8048H, program memory is selected in blocks of 4 K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks, is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum. Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line(s) as a bank enable signal. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

4.5 Control signal summary

Table 2 summarizes the instructions which activate the various control outputs of the 8048/80C49 microcomputers. During all other instructions these outputs are driven to the high-impedance state.

Table 2 8048/80C49 control signals

Control Signal	When active
$\overline{RD}$	During MOVX A,@R or INS Bus
WR	During MOVX @R,A or OUTL Bus
ALE	Every machine cycle
PSEN	During fetch or external program memory (instruction or immidiate data)
PROG	During MOVD A,P ANLD P,A MOVD P,A ORLD P,A

3. The MAB8051/C51/52 microcontroller family

CONTENTS – THE MAB8051/C51/52 MICROCONTROLLER FAMILY

		page
1.0	DESCRIPTION	3-6
2.0	FEATURES	3-7
3.0	FUNCTIONAL DESCRIPTION	3-13
3.1	Program memory	3-13
3.2	Data memory	3-13
3.3	Special function registers (SFRs)	3-14
3.4	Oscillator and clock circuit	3-16
3.5	CPU timing	3-16
3.6	Input/Output operation	3-17
3.6.1	Input/Output operation in the MAB8051/8031	3-18
3.6.2	Writing to a port	3-19
3.6.3	Port loading	3-20
3.6.4	Input/Output operation in the PCB80C51/80C31	3-20
3.7	Timer/event counter	3-21
3.7.1	Modes 0 and 1	3-24
3.7.2	Mode 2	3-24
3.7.3	Mode 3	3-24
3.7.4	Timer/event counter control and status register	3-25
3.7.5	Timer 2	3-26
3.8	Serial I/O interface	3-28
3.8.1	Baud rates	3-30
3.8.1.1	Using Timer 1 to generate baud rates	3-31
3.8.1.2	Using Timer 2 to generate baud rates (8052 only)	3-31
3.8.2	Baud rate bit in PCON	3-33
3.8.3	Serial port control register	3-33
3.8.4	Serial port data registers	3-34
3.8.5	Mode 0	3-34
3.8.6	Mode 1	3-38
3.8.7	Modes 2 and 3	3-40
3.9	Accessing external memory	3-43
3.9.1	Accessing external data memory	3-43
3.9.2	Accessing external program memory	3-43
3.9.3	Overlapping program and data memory spaces	3-44
3.9.4	The address latch enable (ALE)	3-45
3.10	Interrupts	3-45
3.10.1	External interrupt logic	3-47
3.10.2	Interrupt control registers	3-49
3.10.3	Single step mode	3-50
3.11	Reset	3-51
3.11.1	The RST/VPD pin of the MAB8051/8031	3-52
3.11.2	The RST pin of the PCB80C51/80C31	3-52
3.11.3	Power down operation in the MAB8051/8031	3-53

page

3.12 The idle and power down modes of the PCB80C51/80C31 3-54
3.12.1 The idle mode 3-55
3.12.2 The power down mode 3-55
3.13 Program verification 3-56
3.14 NMOS and CMOS versions of the on-chip oscillator 3-57
3.14.1 NMOS versions 3-57
3.14.2 CMOS versions 3-58

4.0 MEMORY, ORGANIZATION, ADDRESSING MODES AND DATA MANIPULATION 3-60
4.1 Memory organization 3-60
4.2 Operand addressing 3-64
4.2.1 Register addressing 3-65
4.2.2 Direct addressing 3-66
4.2.3 Register-indirect addressing 3-66
4.2.4 Immediate addressing 3-67
4.2.5 Base-register plus index-register indirect addressing 3-67
4.3 Data manipulation 3-67
4.4 Boolean processor 3-67
4.5 Data transfer operations 3-68
4.6 Logic operations 3-70
4.7 Arithmetic operations 3-71
4.8 Control transfer 3-73

5.0 INSTRUCTION SET OF THE 8051 3-75
5.1 Organization of the instruction set 3-75
5.1.1 Data transfer 3-76
5.1.2 Logic 3-77
5.1.3 Arithmetic 3-77
5.1.4 Control transfer 3-80
5.2 Instruction definitions 3-89

page

6.0 APPLICATION EXAMPLES FOR THE 8051/80C51 FAMILY OF MICROCOMPUTERS 3-147

6.1 8051 programming techniques 3-147

6.1.1 Radix conversion routines 3-147
6.1.2 Multiple-precision arithmetic 3-148
6.1.3 Table look-up sequences 3-149
6.1.4 Saving CPU status during interrupts 3-151
6.1.5 Passing parameters on the stack 3-152
6.1.6 N-way branching 3-153
6.1.7 Computing branch destinations at run time 3-155
6.1.8 In-line-code parameter passing 3-156

6.2 Peripheral interfacing techniques 3-158

6.2.1 I/O port reconfiguration (first method) 3-158
6.2.2 I/O port reconfiguration (second method) 3-159
6.2.3 Software delay timing 3-160
6.2.4 Serial port and timer configuration 3-161
6.2.5 Simple serial I/O drivers 3-161
6.2.6 Transmitting serial port character strings 3-162
6.2.7 Recognizing and processing special cases 3-163
6.2.8 Synchronizing timer overflows 3-164
6.2.9 Reading a timer/counter without disrupting the timing process 3-164

6.3 Connections to peripherals 3-165

1.0 DESCRIPTION

The MAB8051 and PCB80C51 family of 8-bit microcontrollers are manufactured in both NMOS and CMOS respectively. The 8051/80C51 family consists of:

MAB8051 - single-chip microcontroller with 4 K ROM/128 bytes RAM
MAB8031 - ROMless version of MAB8051 with 128 bytes RAM

PCB80C51 - MAB8051 CMOS version of MAB8051
PCB80C31 - MAB8031 CMOS version of MAB8031

MAB8052 - MAB8051 derivative with Timer 2, 8 K ROM/256 byes RAM

MAB8032 - ROMless version of MAB8052 with 256 bytes RAM

The derivative version of the family, the MAB8032/52AH, is identical to the other family members but posseses an extra timer, Timer 2 and larger on-chip RAM/ROM. The operation of Timer 2 is described separately in section 3.7.5.

Throughout this section of the manual, "8051" is used as the generic term for the 8051/80C51 family of microcontrollers, also for brevity the term "8052" is used to refer to the MAB8032/52AH. Unless a specific function of the 8052 is mentioned, the reader may assume that all descriptions of 8051 functions apply equaly to the 8052. Where specific devices are referred to, their type number will be given.

The MAB8051/PCB80C51 is designed as a stand-alone high-performance single-chip microcontroller for real-time applications such as instrumentation, intelligent computer peripherals and industrial control. Hardware features, architectural improvements and new instructions make the MAB8051/PCB80C51 a powerful and cost effective controller.

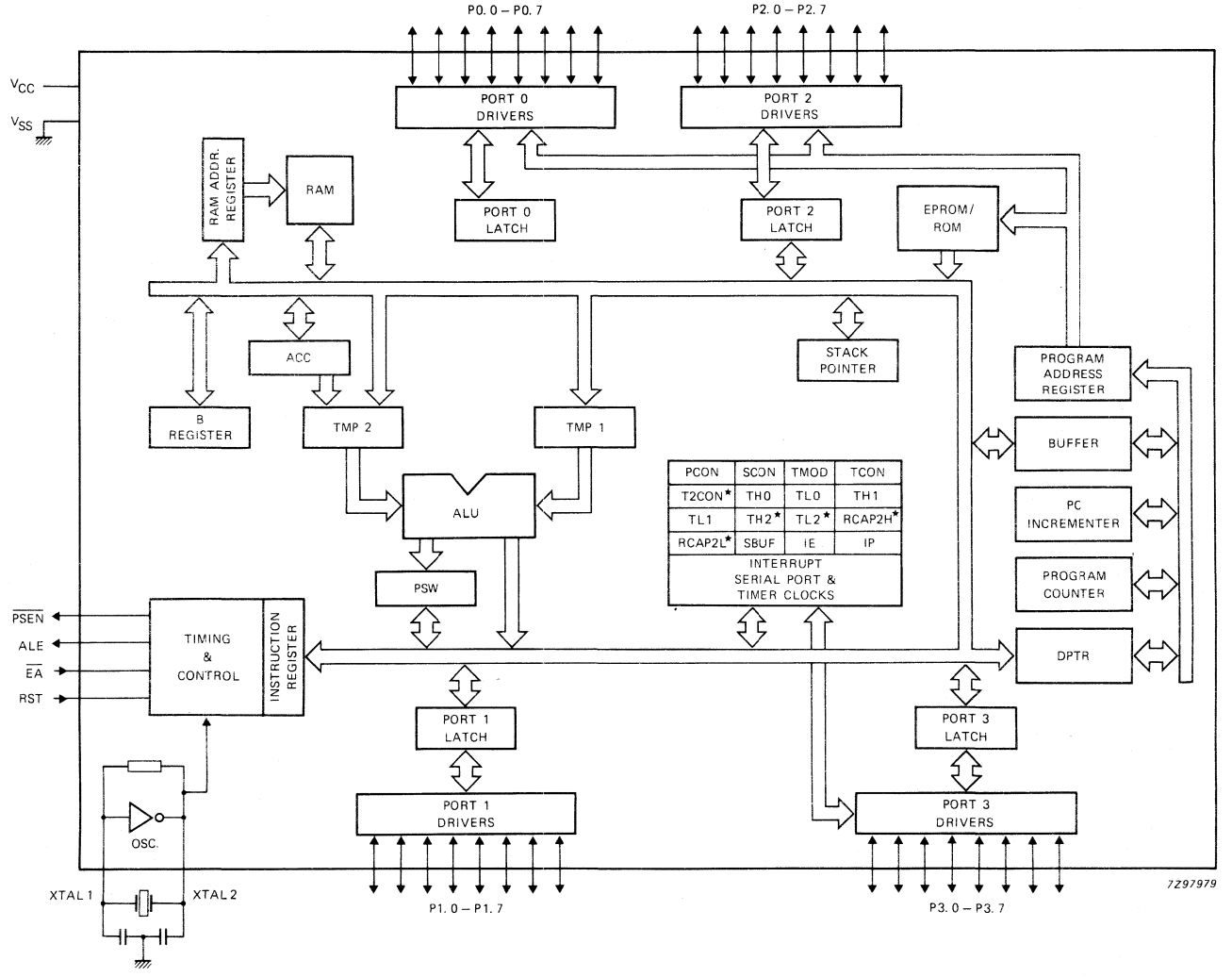
The MAB8031/PCB80C31 is a control-oriented CPU without on-chip program memory. It can address 64K-bytes of external program memory as well as 64K-bytes of externally added data memory. This type of device is most suited to applications requiring the flexibility of external memory which can be easily extended and improved.

The 8051 has 32 I/O lines and a receive-buffered, fully duplex serial I/O. This means as well being able to transmit and receive simultaneously, the device can begin receiving a second byte before a previously received byte has been read from the receive register.

Program and data memories as well as input/output capabilities can be expanded using standard peripherals.

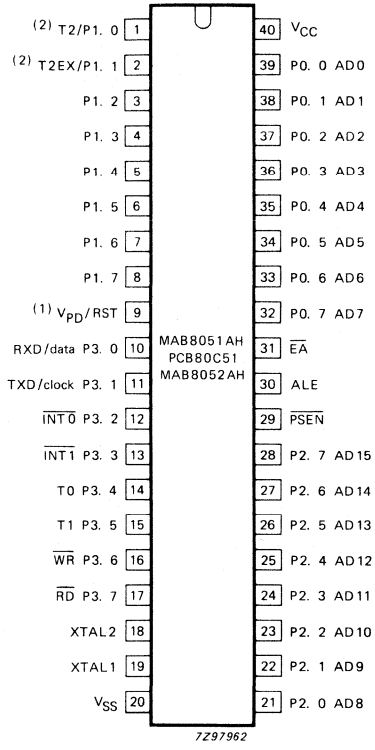
2.0 FEATURES

- 8-bit CPU
- 4K bytes of ROM on MAB8051 and PCB80C51, 8K bytes of ROM on the 8052
- 128 bytes of RAM
- 21 special function registers (26 in the 8052)
- Stack depth limited only by the internal data RAM
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- On-chip oscillator
- Two 16-bit timer/counters (three in the 8052)
- Five interrupt sources with a two priority-level structure (six on the 8052)
- A fully duplex serial I/O
- Bit addressability for Boolean processing
- Reduced power consumption with CMOS versions



* only present in 8052 version

Fig. 2.1 Block diagram of the 8051 family.



(1) NMOS versions only.

(2) applicable to 8052 versions only.

Fig. 2.2 Pinning diagram for the 8051/C51/52 DIL version.

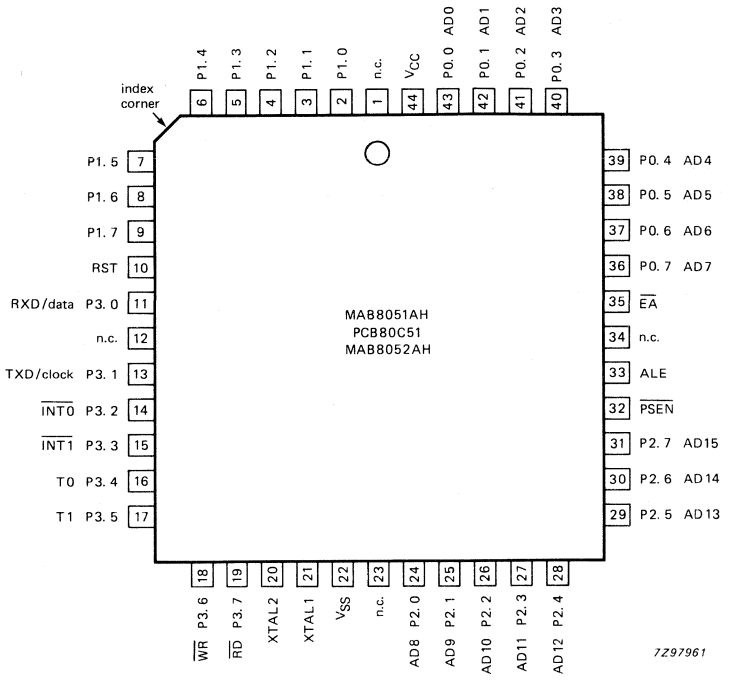


Fig. 2.3 Pinning diagram for the MAB8051/C51/52 PLCC version.

VSS: Circuit ground potential.

VCC: Supply voltage during normal operation and verification.

Port 0: Port 0 is an 8-bit open drain quasi-bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pull-ups). It also outputs instruction bytes during program verification. (External pull-ups are required during program verification).

Port 1: Port 1 is an 8-bit quasi-bidirectional I/O port with internal pull-ups. It receives the low-order address byte during program verification. On the 8052, pins 1 and 2 also supply the alternative functions T2 and T2EX. T2 is the counter trigger input for Timer 2; T2EX the external input to Timer 2. Operation of the alternative functions is determined by the relevant output latch programmed to logic 1.

Port 2: Port 2 is an 8-bit quasi-bidirectional I/O port with internal pull-ups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification.

Port 3: Port 3 is an 8-bit quasi-bidirectional I/O port with internal pull-ups. It serves the functions of various special features of the 8051, as listed below:

Port pin	Alternate function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

RST/VPD: A HIGH level on this pin for two machine cycles while the oscillator is running resets the device. An internal pull-up permits power-on reset using only a capacitor connected to VCC. In the PCB80C51/80C31 this pin is simply termed RST. For the MAB8051 this pin also supplies standby power to the RAM: VPD should be held within its specified limit while VCC drops below its specified limit. When VPD is LOW, the RAM's current is drawn from VCC.

ALE: Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated for this purpose at a constant rate of 1/6th of the oscillator frequency even when the external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external data memory, i.e. the first ALE of the second cycle of a MOVX instruction is missing).

$\overline{PSEN}$: Program Store Enable output is the read strobe to external program memory. $\overline{PSEN}$ is activated twice in every machine cycle during fetches from external program memory (except during a MOVX instruction). $\overline{PSEN}$ is not activated during fetches from internal program memory.

$\overline{EA}$: When $\overline{EA}$ is held HIGH the CPU executes out of internal program memory (unless the program counter exceeds OFFFH, (1FFFH in the 8052)). When $\overline{EA}$ is held LOW the CPU executes only out of external program memory. In the MAB8031, PCB80C31 and MAB8032 versions $\overline{EA}$ must be externally wired LOW.

XTAL1: Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

XTAL2: Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

Re: PCB80C51 and PCB80C31 inc. all 80C51 derivatives

1. At power on

At power on, the voltage at pin 40 (V_{CC}) and pin 9 (RESET) must come up at the same time in order to ensure a correct start up of the microcontroller. If this action is not ensured, it is possible that the microcontroller could act as a thyristor clamped over the power supply.

In general, the voltage at any pin must not be higher than V_{CC}+0,5 V or lower than V_{SS}-0,5 V at any time.

3.0 FUNCTIONAL DESCRIPTION

3.1 Program memory

The program memory address space of the 8051 is 64K-bytes. The lowest 4 K-bytes (8 K in the 8052) of this memory are in the on-chip ROM. The program memory always uses 16-bit addresses.

If the $\overline{EA}$ pin is held HIGH, the 8051 executes instructions from the internal ROM unless the program counter exceeds 0FFFH in the 8051 or 1FFFH in the 8052. Fetches from locations 1000H up to FFFFH in the 8051 (2000H up to FFFFH in the 8052) are directed to the external program memory.

If the $\overline{EA}$ pin is held LOW, the 8051 fetches all instructions from the external program memory. In the specific cases of the MAB8031, PCB80C31 and MAB8032 which have no internal program memory, $\overline{EA}$ must be externally wired LOW to enable the proccessor to fetch the lowest 4 K-bytes from the external program memory. The program memory map is shown in Fig. 3.1.

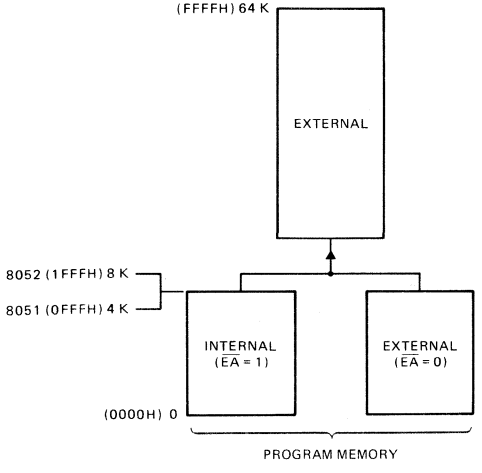


Fig. 3.1 Program memory map.

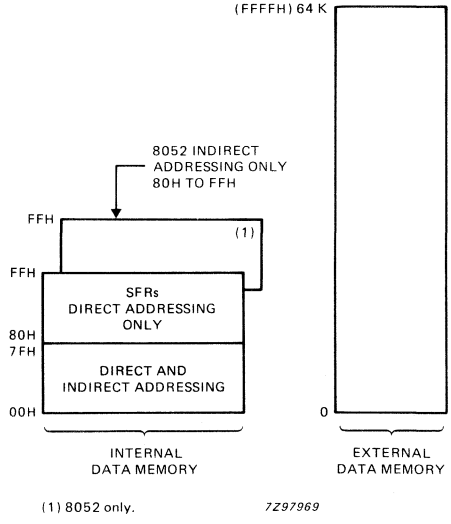


Fig. 3.2 Data memory map.

3.2 Data memory

The data memory of the 8051 comprises 128-bytes of on-chip RAM, 21 special function registers and up to 64 K-bytes of external data memory (see Fig. 3.2).

The data memory of the 8052 comprises 256-bytes of on-chip RAM, 26 special function registers and up to 64 K-bytes of external data memory (see Fig. 3.2).

The external data memory can use either 8-bit or 16-bit addresses. The internal data memory uses 8-bit addresses, providing a 256-location address space. The lower 128 addresses access the on-chip RAM. Various

locations within the upper 128 locations are occupied by the special function registers (SFRs) (see section 3.3).

The lowest 32 bytes of the internal RAM (locations 00 up to 1FH) are divided into four banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the 'working registers' of the CPU and these can be accessed by 3-bit addresses contained in the same byte as the opcode of a register instruction. A large number of the instructions only require a single byte.

The next 16 bytes of the internal RAM (locations 20H up to 2FH) have individually addressable bits. These are designed for use as software flags or for one-bit (Boolean) processing. In addition to the 128 individually addressable bits within the RAM, eleven (twelve in the 8052) of the special function registers also have individually addressable bits.

3.3 Special function registers (SFRs)

The SFRs are as follows:

The registers marked with an * are both byte and bit addressable.

* ACC

ACC is the Accumulator. The mnemonics for accumulator directed instructions refer to the accumulator simply as A, but the register itself is termed ACC.

* B

B, the B register, is used during multiply and divide operations. For other instructions it can be treated as another scratchpad register.

SP

The Stack Pointer, SP, is 8 bits wide. The stack can reside anywhere within the 128 bytes of on-chip RAM. When the 8051 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL instruction, the stack pointer is incremented before data is stored, so the stack would begin at location 08H. This implementation is for compatibility with the 8048 family of microcontrollers. Normally, during initialization the user places the stack within the register array.

DPTR

The Data Pointer (DPTR) is a 16-bit register, consisting of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address as a pointer for the ROM and the external RAM. It may also be used as either a 16-bit register or two separate 8-bit registers.

* P0, P1, P2 and P3.

The four parallel ports provide the 32 I/O lines. Each port consists of a latch (special function registers P0, P1, P2 and P3), an output driver, and an input buffer.

The output drivers of Port 0 and 2, and input buffers of Port 0 are used in accessing the external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2, meanwhile, outputs the high byte of the external memory address.

The output drivers and input buffers of Port 3 are also multi-functional, as given below:

Port pin	Alternate function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	WR (external data memory write strobe)
P3.7	RD (external data memory read strobe)

SBUF

The Serial Data Buffer (SBUF) is actually two registers. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Transmission is initiated by moving a byte to SBUF.) When the data is moved from SBUF, it comes from the receive buffer.

During serial reception the incoming bits are clocked into a separate shift register. When reception of a frame is completed, and various conditions are satisfied, 8 received data bits are transferred from the shift register to the receive buffer. The shift register is then ready to receive a second frame, while the frame already received awaits servicing.

* IP, * IE, TMOD, *TCON, *SCON and PCON.

Interrupt Priority (IP), Interrupt Enable (IE), Timer/Counter Mode (TMOD), Timer/Counter Control (TCON), Serial Control (SCON), and Power Control (PCON) contain the control and status bits for the interrupt system, the timers, and the serial control. They will be described in detail in following chapters of this section.

The other SFRs are:

*	PSW	Program status word
	TH0	Timer/Counter 0 (high byte)
	TL0	Timer/Counter 0 (low byte)
	TH1	Timer/Counter 1 (high byte)
	TL1	Timer/Counter 1 (low byte)

The 8052 contains a further five SFRs dealing with the extra Timer 2:

*	T2CON	Timer/counter 2 control
	RCAP2H	Timer/counter 2 capture register (high byte)
	RCAP2L	Timer/counter 2 capture register (low byte)
	TH2	Timer/counter 2 (high byte)
	TL2	Timer/counter 2 (low byte)

3.4 Oscillator and clock circuit

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which is intended for use as a crystal oscillator, in the Pierce configuration. XTAL2 is also the input to the internal clock generator.

With an external oscillator the chip would be driven through XTAL2 and XTAL1 would be grounded. Since the input to the clock generator is a divide-by-two flip-flop, there are no restrictions placed on the duty cycle of the external oscillator signal. However, minimum HIGH and LOW times must be observed.

The output of the divide-by-two flip-flop provides a two-phase clock signal to the logic. The Phase 1 signal is active during the first half of each clock period, and the Phase 2 signal is active during the second half of each clock period.

3.5 CPU timing

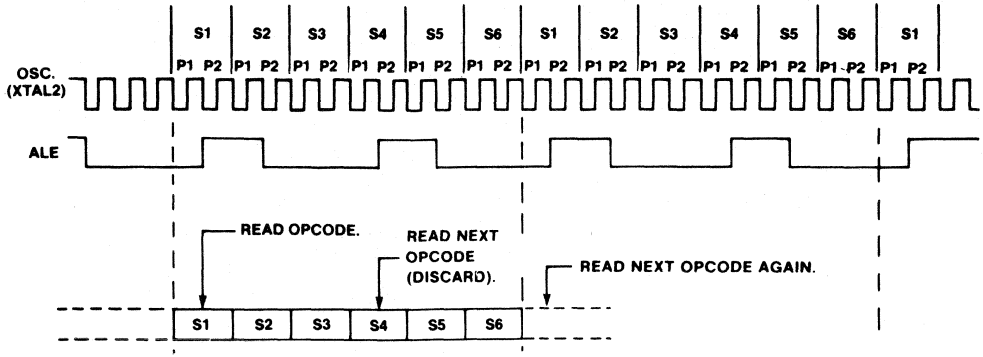
A machine cycle consists of 6 states (12 oscillator periods) and each state is divided into two phases corresponding to the two phases of the clock signal.

The diagrams of Fig. 3.3 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user visable, the XTAL2 and ALE signals are shown for external reference. A machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1 Phase 1) to S6P2 (State 6 Phase 2). Each state has a two oscillator period duration and each phase lasts for one oscillator period. ALE is normally activated twice during each machine cycle, once during S1P2 and S2P1, and again during S4P2 and S5P1.

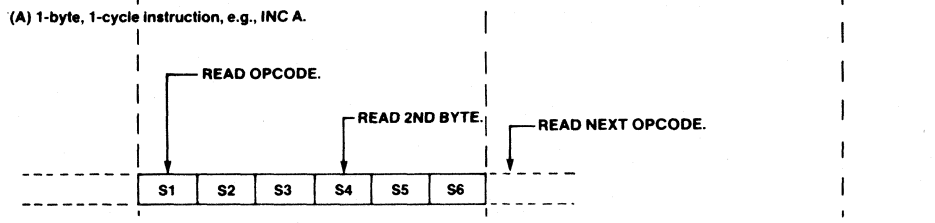
Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the instruction register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one byte instruction, there is still a read at S4, but the byte read (which would be the next opcode) is ignored, and the program counter is not incremented. In any case, execution is complete at the end of S6P2. Fig. 3.3a and b show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction respectively.

Most 8051 instructions are executed in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete, they take 4 cycles.

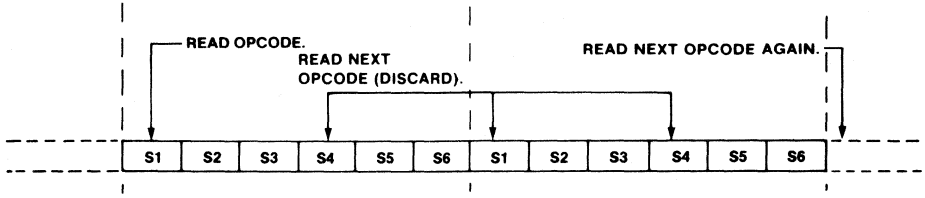
Normally, two code bytes are fetched from program memory during every machine cycle. The only exception to that is when a MOVX instruction is executed. MOVX is a 1-byte, 2-cycle instruction that accesses the external data memory, during which two fetches are skipped while the external data memory is being addressed and strobed. Fig. 3.3c and d show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction respectively.



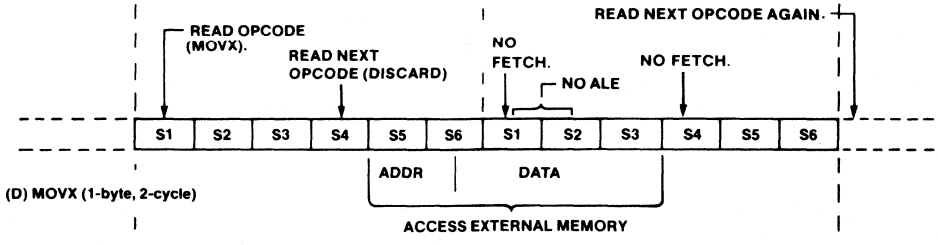
(a) 1-byte, 1-cycle instruction, e.g., INC A.



(b) 2-byte, 1-cycle instruction, e.g., ADD A, #data.



(c) 1-byte, 2-cycle instruction, e.g., INC DPTR.



(d) 1-byte, 2-cycle instruction, MOVX.

Fig. 3.3 Fetch/execute timing.

3.6 Input/Output operation

The 32 I/O lines are divided equally into four ports, all of which are bidirectional. Ports 1,2 and 3 have internal pull-ups, with Port 0 having open-drain outputs. These four ports are termed 'quasi-bidirectional' ports because of the special output circuit structure which allows each line to serve as an input or an output. Fig. 3.4 shows a functional diagram of a typical bit in each of the four ports.

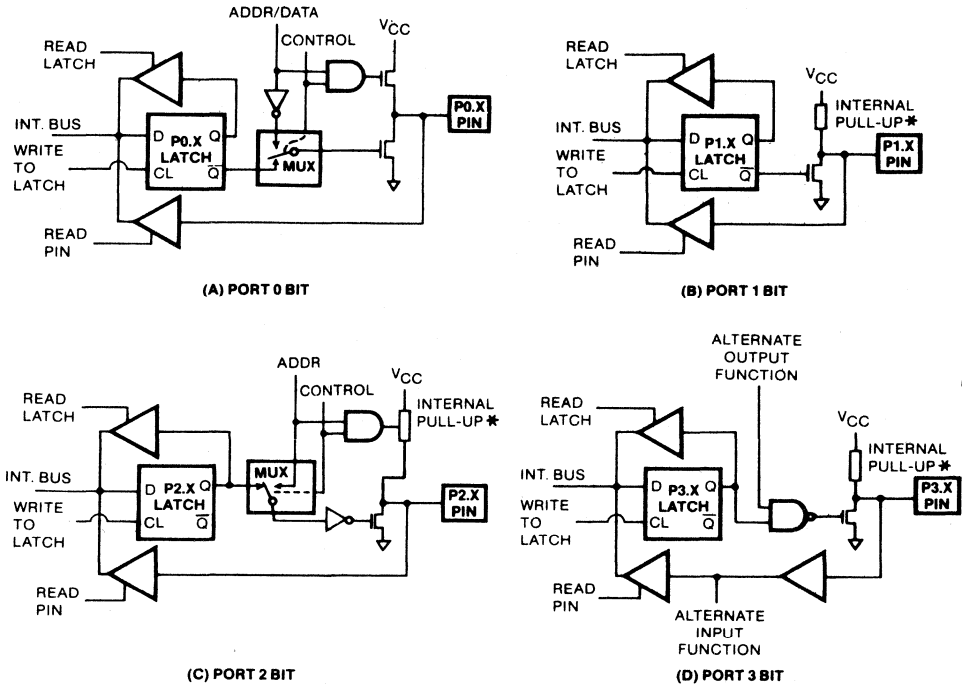


Fig. 3.4 Input/Output port latches and buffers.

3.6.1 Input/Output operation in the MAB8051/8031

Each I/O line can be used independently as an input or as an output. For a line to be used as an input, the port latch must contain a 1, which turns off the output-HIGH driver FET. Then, for Ports 1, 2 and 3, the pin is pulled HIGH by the internal pull-up, but it can be pulled LOW by an external source. In the case of port 0, a 1 in the port latch causes the output pin to float. All port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

As inputs, Ports 1, 2 and 3 can be driven in a normal manner by any TTL or MOS circuit. However they can also be driven by open-collector or open-drain outputs without needing any additional external pull-ups because they have internal pull-ups.

Port 0 differs in not having internal pull-ups. The upper FET in the P0 output driver (see Fig. 3.4a) is turned off except when the port is being used as an ADDR/DATA bus in accesses to external memory. Consequently, P0 lines that are being used as output ports have open-drain outputs. Writing a 1 to a P0 latch results in both output FETs being turned off, so the pin floats, and in this condition it can be used as a high impedance input.

As is evident from Fig. 3.4 there are two ways to read a port: an instruction reads either the latch or the pin. In the 8051, some instructions read the latch and some read the pin.

The instructions that read the latch are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called 'read-modify-write' instructions. The instructions listed below are of read-modify-write type. When the destination operand is a port or a port bit, these instuctions read the latch rather than the pin:

```
ANL           (logical AND, e.g., ANL P1,A)
ORL           (logical OR, e.g., ORL P2,A)
XRL           (logical exclusive-OR, e.g., XRL P3,A)
JBC           (jump if bit=1 and clear bit, e.g., JBC P1,1,LABEL)
CPL           (complement bit, e.g., CPL P3.0)
INC           (increment, e.g., INC P2)
DEC           (decrement, e.g., DEC P2)
DJNZ          (decrement and jump if not zero, e.g., DJNZ P3,LABEL)
MOV PX.Y,C    (move carry bit to bit Y of Port X)
CLR PX.Y      (clear bit Y of Port X)
SET PX.Y      (set bit Y of Port X)
```

It is not obvious that the last three instructions in this list are read-modify-write instructions. What they do is read the port byte, all 8 bits, modify the addressed bit, and then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor and a 1 could be written to it in order to turn the transistor on. If the CPU now reads the same port bit at the pin, instead of the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch will give the correct value of 1. It is to avoid this type of problem that the 8051 directs read-modify-write instructions to the port latch rather than the port pin.

It is evident from Fig. 3.4d that each line in port 3 is able to perform an alternate function, not related to the port function. The bit latch must contain a 1, otherwise the port pin will be stuck at 0 regardless of which alternate function is trying to operate on it. For the alternate functions of the lines of Port 3 see 3.3.

3.6.2 Writing to a port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the last cycle in the instruction. However, port latches are in fact sampled by their output buffers only during the first phase of any clock period. (During the second phase the output buffer holds the value it saw during the previous first phase.) Consequently, the new value of the port latch will not be read until the first phase of the following machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase transition speed. The extra pull-up can source about 100 times more current than the normal pull-up.

It should be noted that the internal pull-ups are FETs and not linear resistors. The pull-up arrangement is shown in Fig. 3.5. The fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. If the port pin is connected to ground, this transistor will allow about 0,25 mA (typical) to exit the pin.

In parallel with the fixed pull-up is an enhancement-mode transistor which is activated during the first machine cycle whenever the port bit makes a 0-to-1 transition. During this interval, if the port pin is connected to ground, this extra transistor will allow an additional 30 mA (typical) to exit the pin.

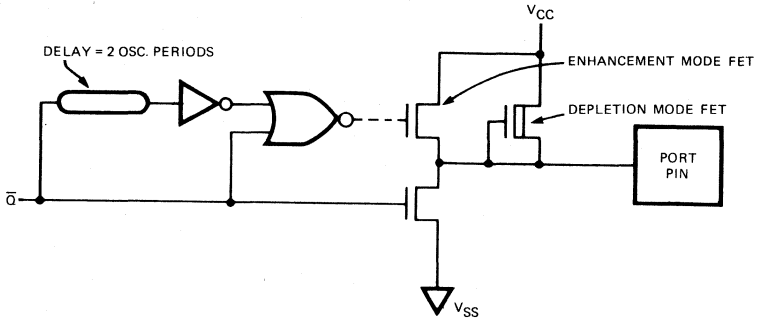


Fig. 3.5 The arrangement of the internal pull-ups for the MAB8051/8031

3.6.3 Port loading

The output buffers of Ports 1, 2, and 3 can drive four LS TTL inputs. The output buffers of Port 0 can each drive eight LS TTL inputs.

Ports 1, 2, and 3 can drive any MOS input without the need for external pull-ups. Port 0 requires external pull-ups to drive MOS inputs, except when it is being used as an ADDRESS/DATA bus, where it requires no such pull-ups.

3.6.4 Input/Output operation in the PCB80C51/80C31

The ports of the PCB80C51/80C31 versions have a similar drive capability to the MAB8051/8031. The output buffers of Ports 1, 2 and 3 are implemented as shown in Fig. 3.6.

Note: When a logic 1 is applied to the gate of an n-channel FET (nFET), the FET is turned on and is turned off when a logical 0 is written to its gate electrode. This is the opposite case to the p-channel FET (pFET).

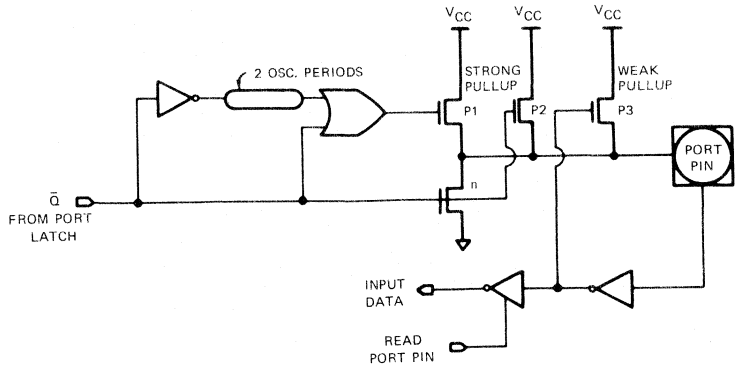


Fig. 3.6 I/O buffers in the PCB80C51/80C31 (Ports 1, 2, and 3).

When the port latch of Port 1, 2 or 3 contains a 0 (Q = 1), all pFETs in Fig. 3.6 are off and the nFET is on. After the port latch makes a 0-to-1 transition, the nFET is turned off; the pFET directly above it (a strong pull-up) is turned on, but only for two oscillator periods. While this pFET is on, it turns on pFET 3 (a weak pull-up) through the inverter. This inverter and pFET form a latch which holds the 1; pFET 2 is also on.

In addition, port 2 uses the strong pull-ups whenever it outputs addresses during accesses to the external program or data memory. In this application, any address bit which is a 1 will have a strong pull-up turned on for the entire duration of the external memory access.

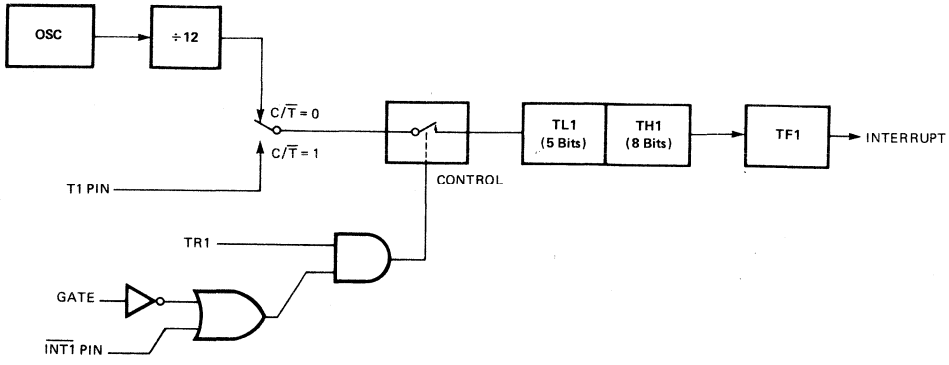
3.7 Timer/event counters

The 8051 has two 16-bit registers, Timer 0 and Timer 1, which can be used as timers or event counters, the 8052 has a third; Timer 2 which is described separately in section 3.7.5. In this chapter there follows a description of Timers 0 and 1. For each timer/event counter there is a control bit in SFR TMOD that selects the timer/event counter function to be either 'timer' or 'counter'.

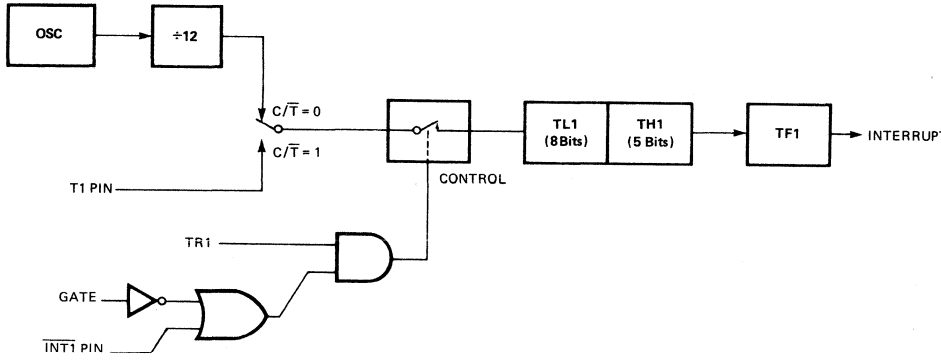
In the 'timer' function the register is incremented every machine cycle and therefore it can be thought of as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the 'counter' function the register is incremented in response to a 1-to-0 transition at its respective external input pin, T0 or T1. In this function the external input is sampled during S5P2 of every machine cycle. When the samples show a HIGH in one cycle and a LOW in the next, the counter is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition is detected. Since it takes two machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions placed on the duty cycle of the external input signal, but to ensure that the given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the 'timer' or 'counter' selection, each timer/event counter has four operating modes from which to select. These modes are functionally illustrated in Fig. 3.7. Operating modes 0, 1, and 2 are the same in both timer/event counters, but in mode 3 they differ.

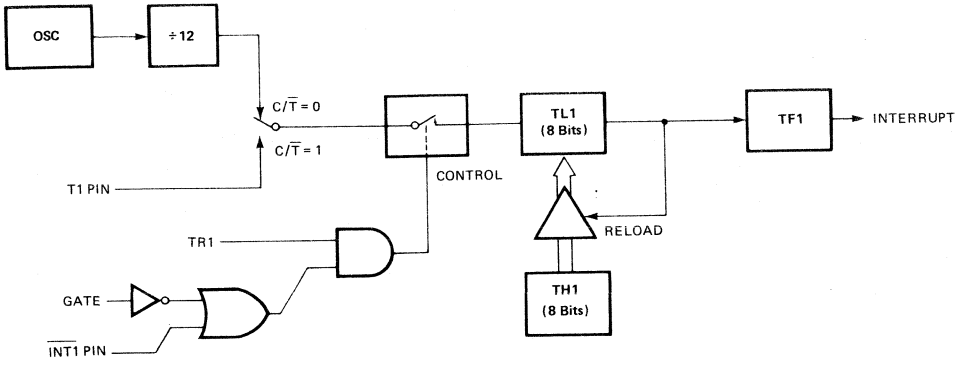


(a) Timer 1 mode 0: 13-bit counter.

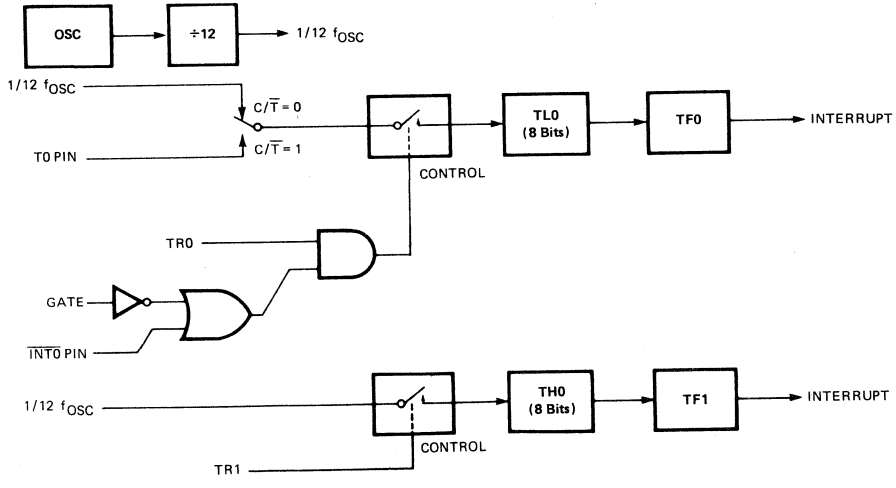


(b) Timer 1 mode 1: 16-bit counter.

Fig. 3.7 Timer/counter modes.



(c) Timer 1 mode 2: auto reload.



(d) Timer 0 in mode 3: split into two 8-bit counters.

Fig. 3.7 (continued).

3.7.1 Modes 0 and 1

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Fig. 3.7a shows the mode 0 operation as it applies to Timer 1.

In this mode the Timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag, TF1. The counter input is enabled to the Timer when TR1 = 1 and either GATE = 0 or $\overline{INT1}$ = 1. (Setting GATE = 1 allows the Timer to be controlled by external input $\overline{INT1}$, to facilitate pulse width measurements.) TR1 is a control bit in SFR TCON, and GATE is a control bit in SFR TMOD.

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 substituting TR0, TF0, and $\overline{INT0}$ for the corresponding Timer 1 signals in Fig. 3.7a. There are two different GATE bits, one for Timer 0 (TMOD.3) and one for Timer 1 (TMOD.7)

Mode 1 is the same as mode 0, except that the Timer is being run with all 16 bits (see Fig. 3.7b).

3.7.2 Mode 2

Mode 2 configures the Timer register as an 8-bit counter with automatic reload. Overflow from TL1 not only sets TF1 but also reloads TL1 with the contents of TH1, which is preset by software to any desired one-byte value. The reload leaves TH1 unchanged. Mode 2 operation is the same for both Timers.

3.7.3 Mode 3

If Timer 1 is put into mode 3, it holds its count. This has the same effect as setting TR1 = 0.

If Timer 0 is put into mode 3, TL0 and TH0 become two separate counters. The logic for mode 3 on Timer 0 is shown in Fig. 3.7d. It should be noted that TL0 is using all of the Timer 0 control bits: $C/\overline{T}$, GATE, TR0, $\overline{INT0}$, and TF0. TH0 is locked into a timer function (counting machine cycles) and has taken over the use of TR1 and TF1 from Timer 1. (Thus TH0 controls the 'Timer 1' interrupt.)

Normally Timer 0 would not be put into mode 3 unless Timer 1 was already in use as a baud rate generator for the serial port. This mode is provided specifically for applications in which two independent timer/event counters plus the serial port are required. Timer 1 would be set up as a baud rate generator (mode 2) and Timer 0 would be operated in mode 3. With Timer 0 in mode 3, an 8051 can appear to have three timers/counters, and the 8052 to have four.

3.7.4 Timer/event counter control and status register

Two SFRs, TMOD and TCON, are used to define the operating modes and control the functions of the timer/event counters. When the instruction changes the content of TMOD or TCON, the change is latched into the SFR and takes effect at S1P1 of the next instruction's first cycle. The registers are shown below.

TMOD: Timer mode control register

Bit:

7	6	5	4	3	2	1	0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
←——Timer 1——→				←——Timer 0——→			

where M1, M0 specify the mode as follows:

M1	M0	Mode	Description
0	0	0	13-bit counter
0	1	1	16-bit counter
1	0	2	8-bit counter with automatic reload
1	1	3	split Timer 0 into two 8-bit counters or stop Timer 1

C/$\overline{T}$ selects 'counter' or 'timer' function. Set for 'counter' function (count negative transitions at T0 or T1 pin). Clear for 'timer' function (count machine cycles).

GATE is the gating control. When set, Timer 'x' is enabled only while the $\overline{INT}$x pin is high and the TRx bit is set. When cleared, Timer 'x' is enabled whenever the TRx bit is set.

Note that all bits of TMOD are cleared by reset.

TCON: Timer control register

Bit:

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

where

TF1: is the Timer 1 overflow interrupt flag. It is set by hardware when Timer 1 overflows and is cleared by hardware when the processor transfers control to the interrupt service routine.

TR1: is the Timer 1 run control bit which is set/cleared by software to turn Timer 1 on/off.

TF0 : is the Timer 0 overflow interrupt flag. It is set by hardware when Timer 0 overflows and is cleared by hardware when the processor transfers control to the interrupt service routine.

TR0: is the Timer 0 run control bit which is set/cleared by software to turn Timer 0 on/off.

IE1: is the external interrupt 1 edge flag. If IT1 = 1, this bit is set by hardware when INT1 is detected to have made a 1-to-0 transition. This bit is cleared by hardware when the processor transfers control to the interrupt service routine.

IT1: determines whether external interrupt 1 is edge-triggered or level-triggered. If IT1 = 1, external interrupt 1 is edge-triggered. If IT1 = 0, external interrupt 1 is triggered by a detected LOW at INT1.

IE0: is the external interrupt 0 edge flag. If IT0 = 1, this bit is set by hardware when INT0 is detected to have made a 1-to-0 transition. This bit is cleared by hardware when the processor transfers control to the interrupt service routine.

IT0: determines whether external interrupt 0 is edge-triggered or level-triggered. If IT0 = 1, external interrupt 0 is edge-triggered. If IT0 = 0, external interrupt 0 is triggered by a detected LOW at INT0.

Bits IE1 to IT0 have to do with the external interrupts, and have been more fully discussed in the previous section on the interrupt structure.

Note that all the bits in TCON are cleared by reset.

3.7.5 Timer 2 (8052 only)

Timer 2 is 16-bit timer counter which is only present in the 8052. Just like Timers 0 and T1, it can operate either as a timer or an event counter. This is selected by bit C/T2 in the special function register T2CON. It has three operating modes: 'capture', 'auto-reload and 'baud-rate generator' which are selected by bits in T2CON as shown in Table 3.1.

T2CON: Timer 2 control signals

Bit:	7	6	5	4	3	2	1	0
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

where:

TF2: Timer 2 overflow flag is set by a timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK or TCLK = 1.

EXF2: Timer 2 external flag is set when either a capture or reload is caused by a negative transition on T2EX when EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.

RCLK: Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.

TCLK: Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.

EXEN2: Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.

TR2: Start/stop control for Timer 2. A logic 1 starts the timer.

C/T2: Timer or counter select:

0 = Internal timer (OSC/12)
1 = External event counter (falling edge triggered)

CP/RL2: Capture/reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto-reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Table 3.1 Timer 2 operating modes

RCLK+TCLK	CP/RL2	TR2	MODE
0	0	1	16-bit auto reload
0	1	1	16-bit capture
1	0	1	baud rate generator
X	X	0	(off)

In the capture mode there are two options which are selected by bit EXEN2 in the T2CON register. If EXEN2 = 0, then Timer 2 is a sixteen bit counter which upon overflow sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt. The capture mode is illustrated in figure 3.8.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2. The auto-reload mode is illustrated in figure 3.9.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It is described in conjunction with the serial port.

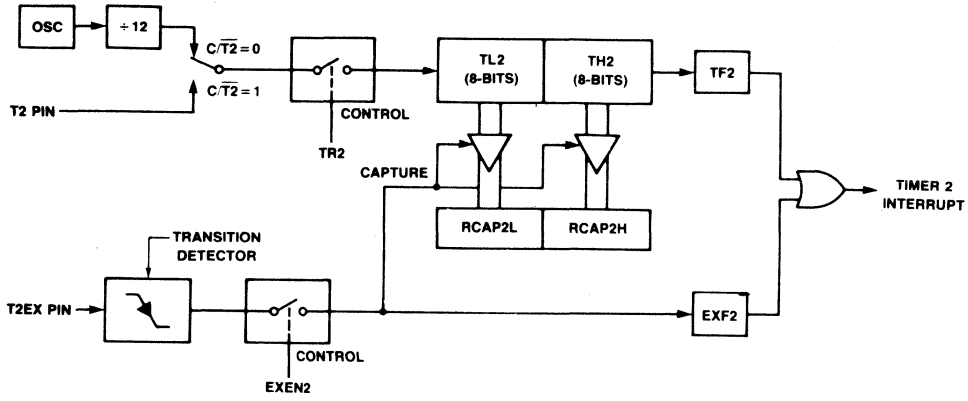


Fig. 3.8 Timer 2 in capture mode.

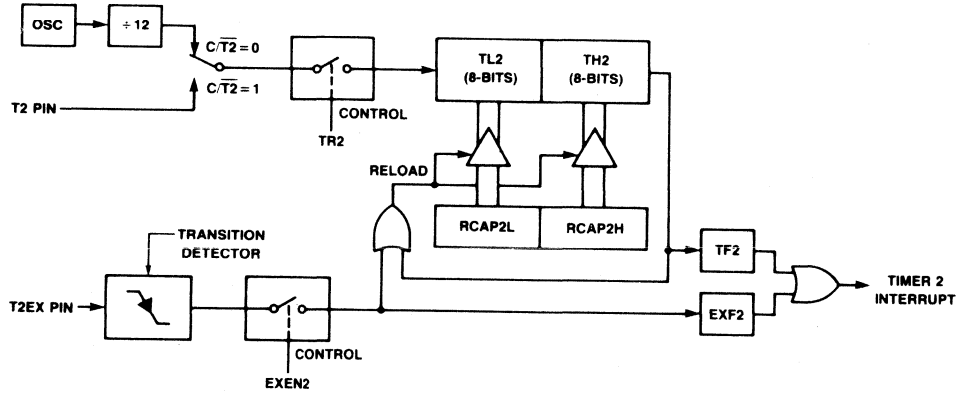


Fig. 3.9 Timer 2 in auto-reload mode.

3.8 Serial I/O interface

One of the main features of the 8051 is the full duplex serial I/O port which means it can transmit and receive simultaneously. This serial port is also receive-buffered, meaning it can commence reception of a second byte before the previously received byte has been read from the receive register. If, however, the first byte has still not been read by the time reception of the second byte is complete, one of the bytes will be lost. The serial port registers are both accessed at SFR SBUF. A 'write to SBUF' loads the transmit register, and a 'read' accesses a physically separate receive register.

The serial port can operate in 4 modes:

Mode 0 - synchronous operation:

Serial data is transmitted and received through RXD, and TXD outputs the shift clock. 8 data bits are transmitted/received LSB first. The baud rate is fixed at 1/12 of the oscillator frequency.

Mode 1 - asynchronous operation:

10 bits are transmitted via TXD or received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SFR SCON. The baud rate is variable.

Mode 2 - asynchronous operation

11 bits are transmitted via TXD or received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. With nominal software TB8 can be made a parity bit as shown in Fig. 3.10. In the receive case, the 9th data bit goes into RB8 in the SFR SCON, and the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency.

```
MOV C,P      ; Parity moved to carry (byte
               already in A)
MOV TB8, C   ; Put carry into Transmit
               Bit 8
MOV SBUF, A  ; Load Transmit Register
```

Fig. 3.10 Generating parity and initiating a transmission.

Mode 3 - asynchronous operation

Mode 3 is the same as Mode 2 in every respect except the baud rate which is variable.

Modes 2 and 3 have special provision made for multiprocessor communications. In these modes 9 data bits are received and the 9th of these goes into RB8. This 9th bit is followed by the stop bit. The port can be programmed so that when the stop bit is received, the serial port interrupt will be activated if, and only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. The way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte to see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that were not being addressed leave their SM2 bits set and continue with the functions they were carrying out before they were interrupted, ignoring the coming data bytes.

SM2 has no effect in mode 0, but in mode 1 it can be used to check the validity of the stop bit. In a mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

3.8.1 Baud rates

The baud rate for Mode 0 is fixed at 1/12 of the oscillator frequency. For Mode 2 the baud rate is either 1/64 or 1/32 of the oscillator frequency, depending on the value of the bit SMOD in SFR PCON. If SMOD = 0 (which is its value on reset), the baud rate will be 1/64 of the oscillator frequency. If SMOD = 1, then the baud rate will be 1/32 of the oscillator frequency.

In the 8051 the baud rates in modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit the other for receive).

3.8.1.1 Using Timer 1 to generate baud rates

Baud rate = (Timer 1 overflow rate)/n

The value of integer n is determined by the value of SMOD. If SMOD = 0 (which is its value on reset), then n = 32, and n = 16 when SMOD = 1. Timer 1 can be configured in any mode and the overflow rate is determined by its count rate and how many counts it takes to reach overflow.

For example, Timer 1 can be configured in the auto reload mode (TMOD.5 = 1, TMOD.4 = 0). The Timer must be running (TCON.6 = 1), and to keep the overflows from generating unnecessary interrupts the Timer 1 interrupt should be disabled (IE.3 = 0). Then the overflow rate depends on the reload value in TH1, as follows:

Overflow rate = (count rate)/[256-(TH1)]

For very low baud rates one might select the 16-bit Timer 1 mode (TMOD.5 = 0, TMOD.4 = 1), and use the Timer 1 interrupt to do a software reload. In this case one would want to have the Timer 1 interrupt enabled (IE.3 = 1). In any case, if Timer 1 is running with bit C/T = 0, the count rate is 1/12 of the oscillator frequency. If the timer is running with C/T = 1, the count rate is the external input frequency, whose maximum practical value is 1/24 of the oscillator frequency.

Table 3.2 lists the various commonly used baud rates and how they can be achieved with the 8051.

BAUD RATE	f_{osc}	SMOD	TIMER 1		
			C/T	MODE	RELOAD VALUE
MODE 0 MAX: 1MHZ	12 MHZ	X	X	X	X
MODE 2 MAX: 375K	12 MHZ	1	X	X	X
MODES 1,3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5	11.986 MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FEEBH

Table 3.2 Baud rates in common use.

3.8.1.2 Using Timer 2 to generate baud rates (8052 only)

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON. Note that the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode as shown in Fig. 3.11.

The baud rate generator mode is similar to the auto-reload mode, in that a roll-over in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset in software.

Now, the baud rates in modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{modes 1 and 3 baud rate} = \frac{\text{Timer 2 overflow rate}}{16}$$

The timer can be configured for either 'timer' or 'counter' operation. In the most typical applications, it is configured for 'timer' operation i.e. C/T2 = 0. 'Timer' operation is a little different for Timer 2 when it's being used for baud rate generation. Normally as a timer it would increment every machine cycle (thus at 1/12 of the oscillator frequency). As a baud rate generator however, it increments every state time (at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

$$\text{modes 1, 3 baud rate} = \frac{\text{Oscillator frequency}}{32 \times \left[65536 - (\text{RCAP2H, RCAP2L})\right]}$$

Where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as an unsigned 16-bit binary integer.

Timer 2 as a baud rate generator is shown in fig. 3.11. This figure is valid only if RCLK + TCLK = 1 in TCON. Note that a roll-over in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from RCAP2H and RCAP2L to TH2 and TL2. Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt if desired.

It should be noted that when Timer 2 is running (TR = 2) in timer function in the baud rate generator mode, a read/write from TH2 or TL2 should not be undertaken. This is because the timer is being incremented every state time and as such results may be inaccurate. The RCAP (re-load registers) may be read, but it is not advised to write to them as it may overlap a reload and errors. Clear TR2 (i.e. switch off the timer) before accessing Timer 2 or RCAP registers.

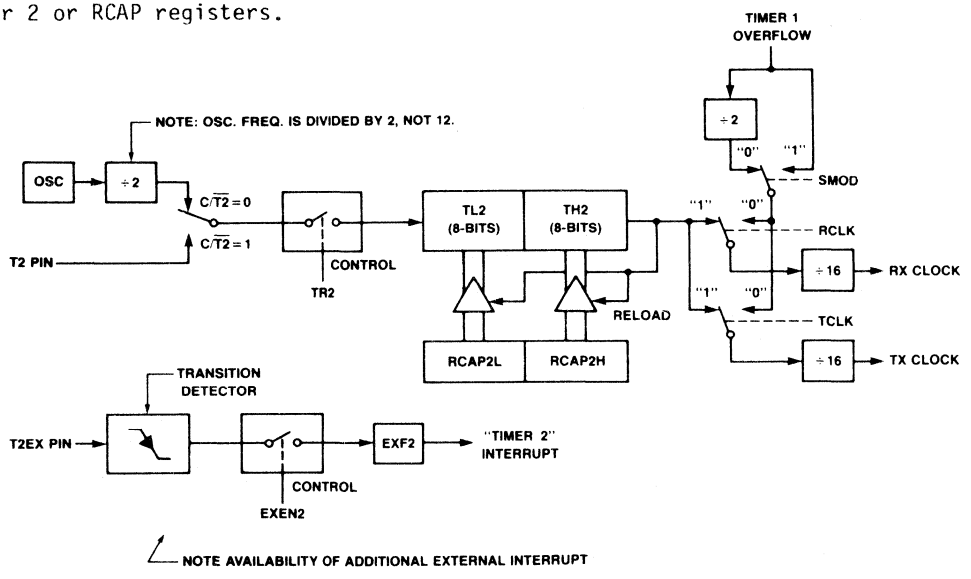


Fig 3.11 Timer 2 in baud rate generator mode.

3.8.2 Baud rate bit in PCON

PCON is an SFR (address = 87H) which has been added to the 8051 to implement certain power control options in the PCB80C51/80C31 versions (see section 3.12). In the MAB8051/8031 versions all bits, except bit 7, in PCON are dummy bits. Bit 7 is SMOD, which is used in all versions to double the baud rate in modes 1, 2, and 3. PCON is not bit addressable.

The reset value of SMOD is 0. Writing a 1 to SMOD (MOV PCON,#FFH or MOV 87H,#FFH) doubles the baud rate in modes 1, 2, and 3.

3.8.3 Serial port control register

SFR SCON is used to define the operating mode and control certain functions of the serial port. It also receives the 9th data bit (RB8), and contains the transmit and receive interrupt flags (TI and RI). The register is as shown below:

SCON: Serial port control register

Bit:	7	6	5	4	3	2	1	0
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

where SM0 and SM1 specify the serial port mode as follows:

SM0	SM1	Mode	Desription	Baud rate
0	0	0	shift register	$f_{osc}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{osc}/64$ or $f_{osc}/32$
1	1	3	9-bit UART	variable

SM2: enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit is not received. In mode 0, SM2 should be 0.

REN: enables serial reception. Set by software to enable reception and cleared by software to disable reception.

TB8: is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as required.

RB8: in modes 2 and 3, is the 9th data bit that is received. In mode 1, if SM2 = 0, RB8 is the stop bit that is received. In mode 0, RB8 is not used.

TI: is the transmit interrupt flag. It is set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmision. TI must be cleared by software.

RI: is the receive interrupt flag. It is set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit in the other modes, in any serial reception (except see SM2). RI must also be cleared by software.

Note: All bits of SCON are cleared by reset.

When an instruction changes the content of SCON, the change is latched into the SFR and takes effect at S1P1 of the next instruction's first cycle. If, however, a serial transmission is already in progress, the TB8 that will be output is the old and not the new value.

When transmission of a serial frame is complete, flag bit TI is activated, which in turn activates the serial port interrupt. The same interrupt is activated by flag bit RI when an incoming frame has been received. Thus the CPU normally enters the serial port interrupt routine without knowing beforehand whether the interrupt was generated by TI or RI. Both flags are located in the SFR SCON. Neither flag is cleared by the hardware, and therefore the interrupt service routine must clear the flag that generated the interrupt. If this were not the case, another interrupt would be generated immediately by the same flag.

3.8.4 Serial port data registers

In all serial modes a 'write to SBUF' loads the same 9-bit shift register. The data byte occupies the first 8 bits, with the LSB at the output bit of the register. The write to SBUF also loads the 9th bit of the shift register with either a 1 or TB8, depending on the mode. It is this bit that initiates the transmission.

The receive registers are an input shift register which is 8 bits wide in mode 0 and 9 bits wide in the other modes, plus SBUF itself, a read-only register which is loaded by the hardware with the data byte at the same time as RI is activated. In the UART modes, the 9th bit is loaded into RB8 in SCON at the same time as the data byte is loaded into SBUF. RB8 and SBUF are not changed if SM2 causes the received data to be ignored.

3.8.5 Mode 0

In mode 0 the serial data enters and exits through RXD, with TXD outputting the shift clock. 8 bits are transmitted/received, i.e. 8 data bits LSB first. The baud rate is fixed at 1/12 of the oscillator frequency.

Fig. 3.12 shows a simplified functional diagram of the serial port in mode 0, and the associated timing diagrams.

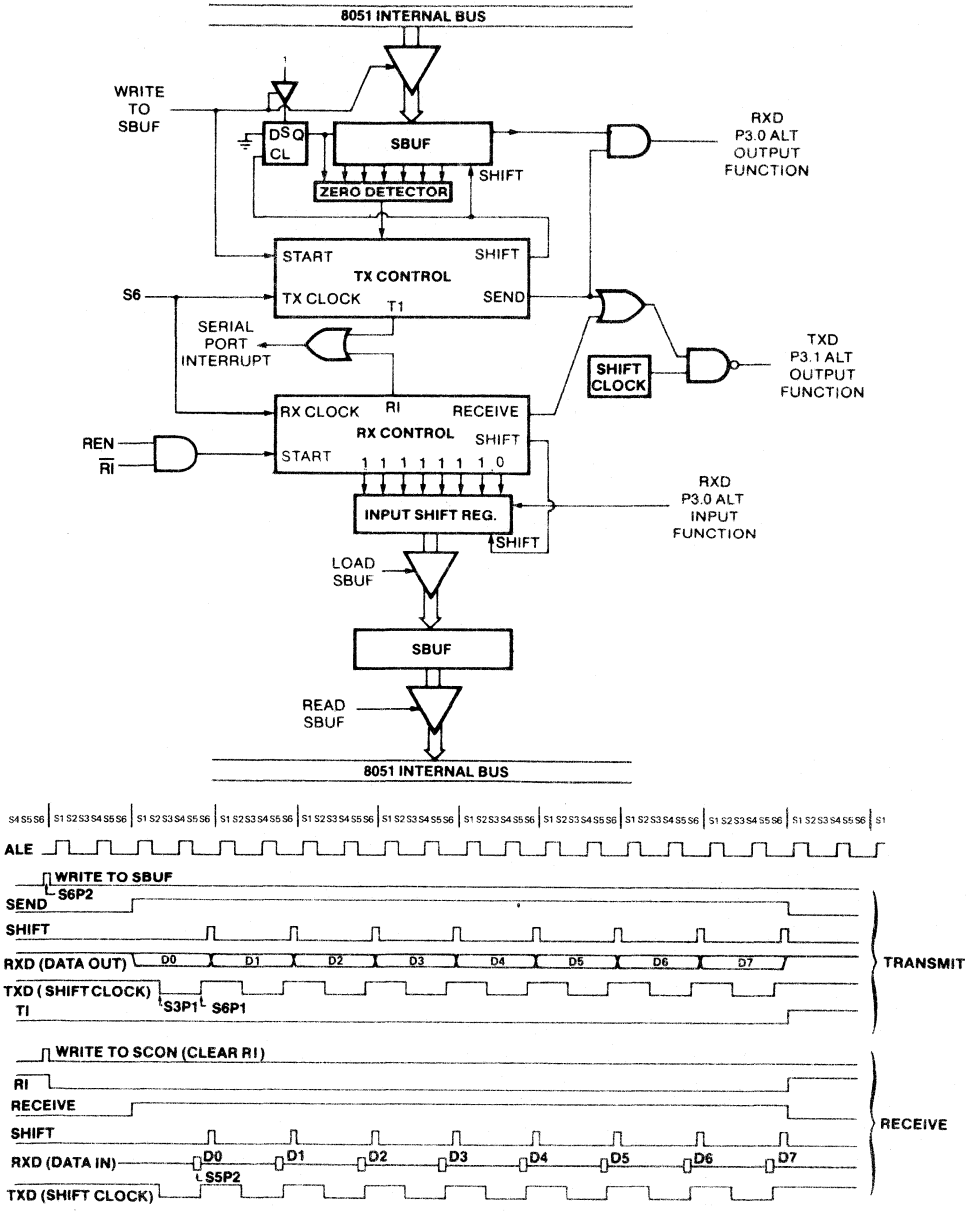


Fig. 3.12 The serial port in mode 0.

Transmission is initiated by any instruction that uses SBUF as a destination register. The 'write to SBUF' signal at the S6P2 period of an instruction also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between 'write to SBUF' and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function of line P3.1. SHIFT CLOCK is LOW during S3, S4, and S5 of every machine cycle, and HIGH during S6, S1, and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right by one position.

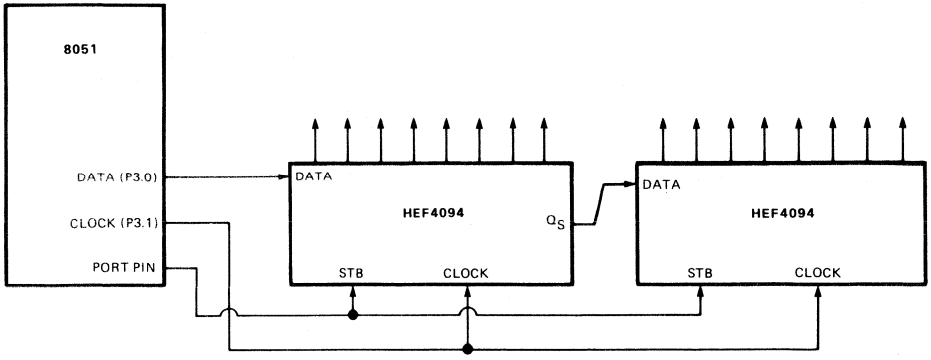
As the data bits shift out to the right, zeroes come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control block to do one last shift,then deactivate SEND and set TI. Both of these actions occur at S1P1 of the 10th machine cycle after 'write to SBUF'.

Reception is initiated by clearing RI, provided that REN = 1. At S6P2 of the next machine cycle the RX Control unit writes the bits 11111110 to the receive register, and in the next clock phase activates RECEIVE.

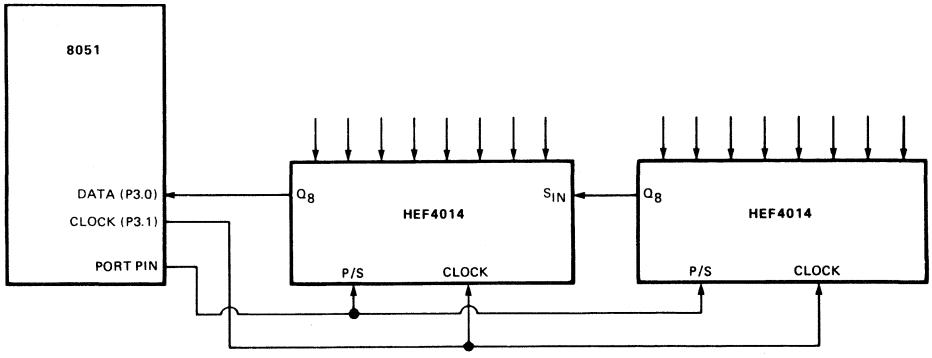
RECEIVE enables SHIFT CLOCK to the alternate output function of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the furthest right position arrives at the furthest left in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

Mode 0 was intended primarily for I/O expansion using CMOS or TTL shift registers, as shown in Fig. 3.13.



(a) Additional outputs



(b) Additional inputs

Fig. 3.13 Mode 0 applications.

3.8.6 Mode 1

In mode 1, ten bits are transmitted via TXD or received through RXD: a start bit (0), 8 data bits LSB first, and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. The baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined by either the Timer 1 or Timer 2 overflow rate, or both (one for transmit the other for receive).

Fig. 3.14 shows a simplified functional diagram of the serial port in mode 1 and the associated timing diagrams for transmit and receive.

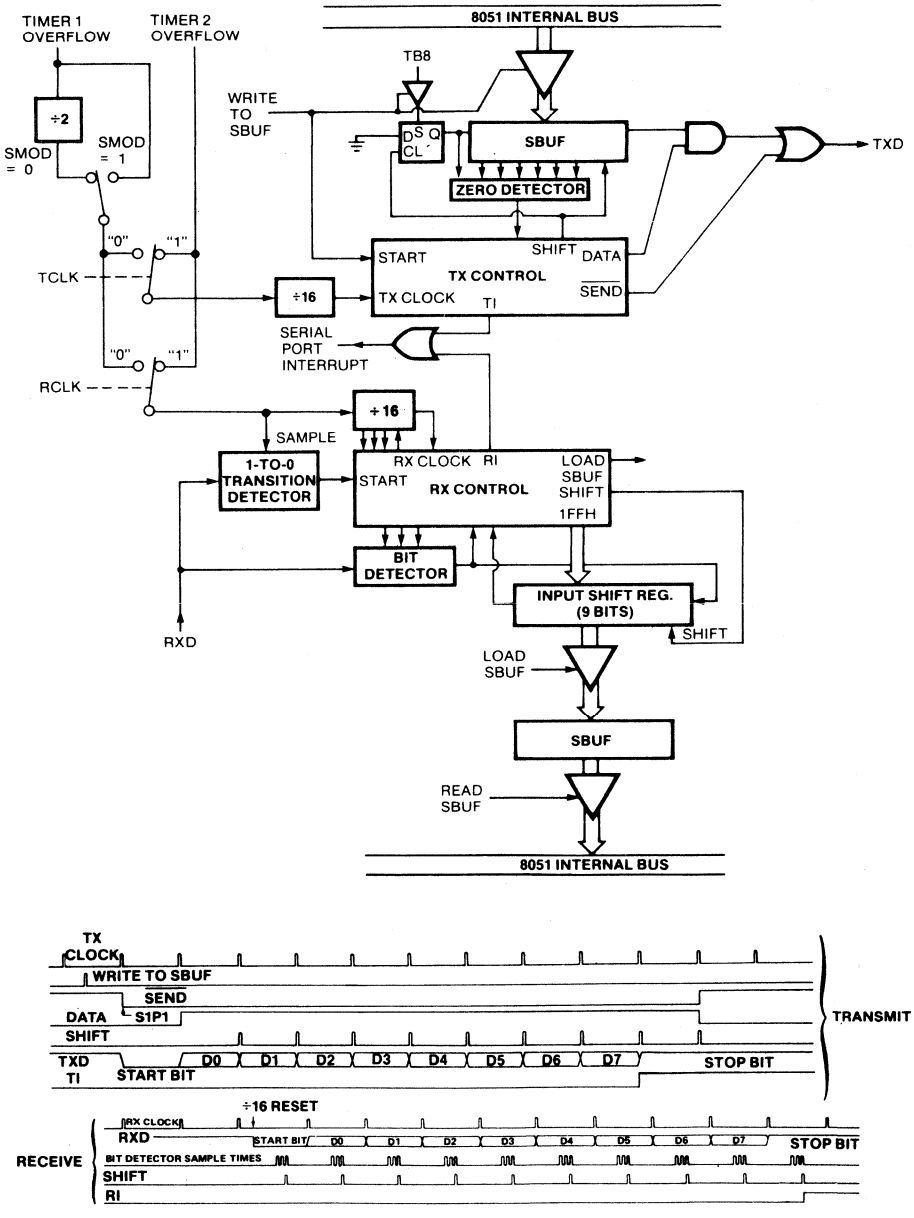


Fig. 3.14 The serial port in mode 1. TCLK, RCLK and Timer 2 are present in the 8052 only.

Transmission is initiated by any instruction that uses SBUF as a destination register. The 'write to SBUF' signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus the bit times are synchronized to the divide-by-16 counter, not to the 'write to SBUF' signal.)

Transmission begins with the activation of $\overline{\text{SEND}}$, which puts the start bit at TXD. One bit time later DATA is activated and this enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeroes are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift, then deactivate $\overline{\text{SEND}}$ and set TI. This occurs at the 10th divide-by-16 rollover after 'write to SBUF'.

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value which is accepted is the value that has been detected in at least two of the last three samples. This is done for noise rejection purposes. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the furthest left position in the shift register (which in mode 1 is a 9-bit register), it flags the RX Control block to carry out one last shift, load SBUF and RB8, and to set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

a) RI = 0, and
b) Either SM2 = 0 or the received stop bit = 1

If either of these conditions is not met, the received frame will be irretrievably lost. If both conditions are met, the stop bit goes into RB8, and the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

3.8.7 Modes 2 and 3

In modes 2 and 3, eleven bits are transmitted via TXD or received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. In the receive state the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/64 or 1/32 of the oscillator frequency in mode 2 and is variable in mode 3.

In the 8052, mode 3 may have a variable baud rate, generated from either Timer 1 or Timer 2 depending upon the state of TCLK and RCLK.

Fig. 3.15 and 3.16 show a simplified functional diagram of the serial port in modes 2 and 3 respectively. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

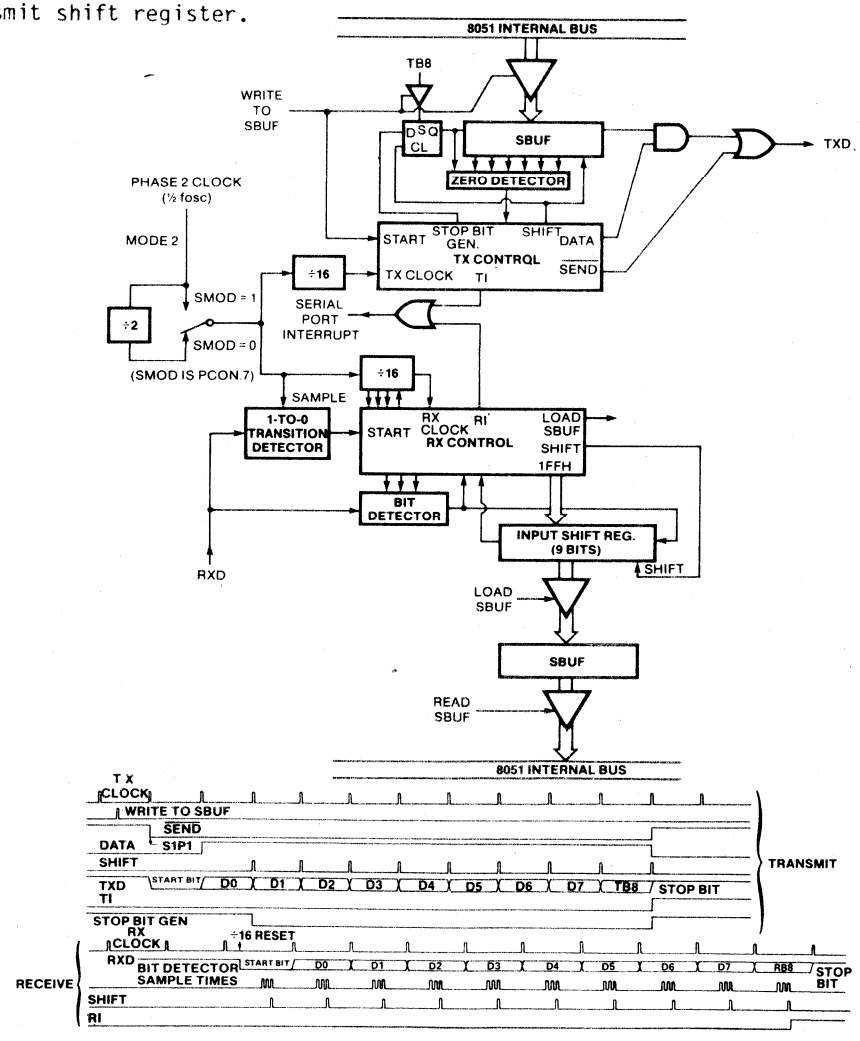


Fig. 3.15 The serial port in mode 2

Transmission is initiated by any instruction that uses SBUF as a destination register. The 'write to SBUF' signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next roll-over in the divide-by-16 counter. (Thus the bit times are synchronized to the divide-by-16 counter, not to the 'write to SBUF' signal.)

The transmission begins with the activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register, and thereafter only zeroes are clocked in.

Thus as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all the positions to the left of that contain zeroes. This condition flags the TX Control unit to carry out one last shift, then deactivate SEND, and finally set TI. This occurs at the 11th divide-by-16 rollover after 'write to SBUF'.

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is reset immediately, and 1FFH is written to the input shift register.

At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was detected in at least two of the last three samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the furthest left position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to carry out one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

a) RI = 0, and
b) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions are met or not, the unit goes back to looking for a 0-to-1 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

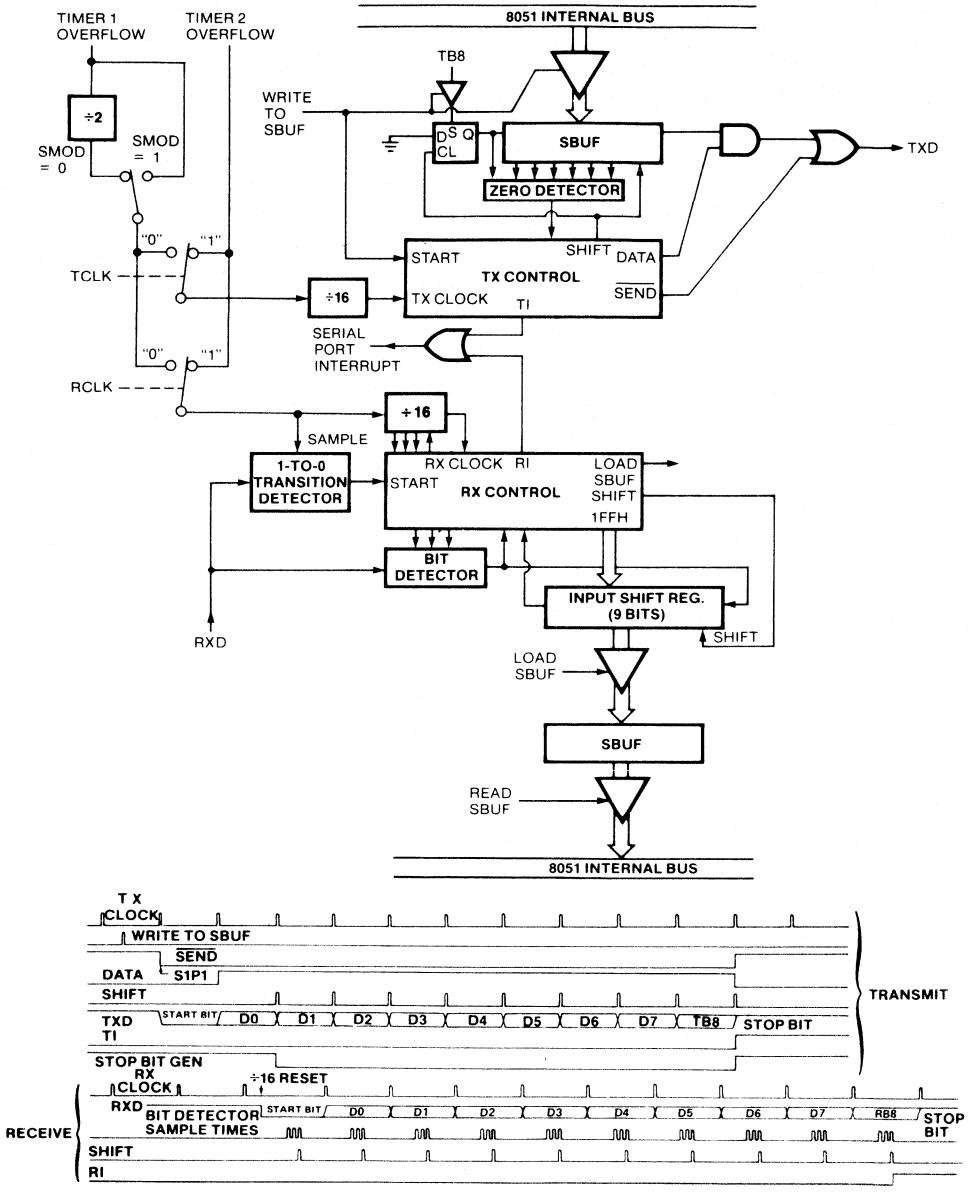


Fig. 3.16 The serial port in mode 3. TCLK, RCLK and Timer 2 are present in the 8052 only.

3.9 Accessing external memory

Access to external memory is of two types: accesses to external program memory and accesses to external data memory. Accesses to external program memory use signal $\overline{PSEN}$ (program store enable) as the read strobe. Accesses to external data memory use $\overline{RD}$ or $\overline{WR}$ (alternate functions of P3.7 and P3.6, respectively) to strobe the memory.

3.9.1 Accessing external data memory

Access to external data memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out of Port 2, where it is held high for the duration of the read or write cycle. During this time the Port 2 latch (SFR) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear shortly after the read or write strobe is deactivated.

If an 8-bit address is being used, the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This facilitates paging.

In either case, the low-order byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus in this application the Port 0 pins are not open-drain outputs, and they do not require external pull-ups. Signal ALE (address latch enable) should be used to lock the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before $\overline{WR}$ is activated, and remains there until after the $\overline{WR}$ is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes OFFH to the Port 0 SFR, thus obliterating whatever information the Port 0 SFR may have been holding.

3.9.2 Accessing external program memory

Fetches from external program memory always use a 16-bit address. The external program memory is accessed under two conditions:

a) Whenever the program counter (PC) contains a number that is larger than OFFFH (1FFFH in the 8052).

b) Whenever signal $\overline{EA}$ is active, regardless of the contents of PC.

The 8031 is an 8051 without internal program memory and $\overline{EA}$ must be externally wired LOW to enable the 8031 to fetch the lower 4 K (lower 8 K in the 8032) program bytes from the external memory.

When the CPU is executing out of external program memory, all 8 bits of Port 2 are dedicated to an output function: during external program fetches they output the high byte of the PC, and during accesses to external data memory they output either DPH or the Port 2 SFR (depending on whether the external data memory access is a MOVX @DPTR or a MOVX @Ri instruction).

The read strobe for external fetches is $\overline{PSEN}$, which is not activated for internal fetches. When the CPU is accessing external program memory, PSEN is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When $\overline{PSEN}$ is activated its timing is not the same as $\overline{RD}$. A complete $\overline{RD}$ cycle, including activation and deactivation of ALE and $\overline{RD}$, takes 12 oscillator periods. A complete $\overline{PSEN}$ cycle, including activation and deactivation of ALE and $\overline{PSEN}$, takes 6 oscillator periods. The execution sequence for these two types of read cycle are shown in Fig. 3.17 for comparison.

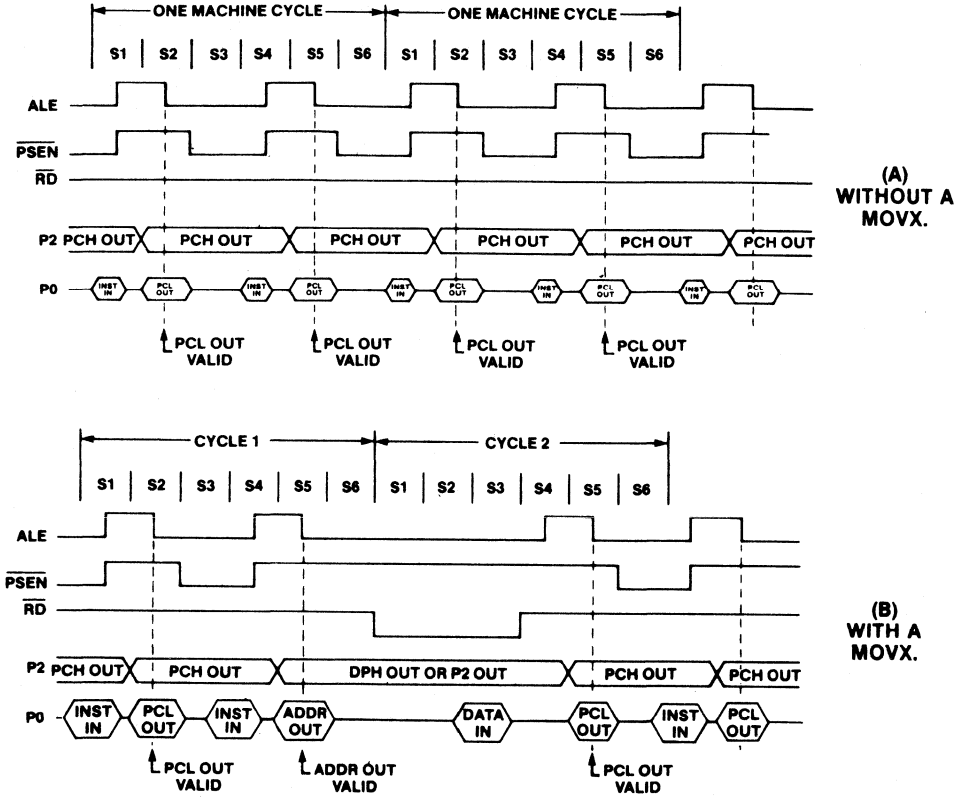


Fig. 3.17 Executing out of the external program memory.

3.9.3 Overlapping program and data memory spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051 the external program and data memory spaces can be combined by ANDing $\overline{PSEN}$ and $\overline{RD}$. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the $\overline{PSEN}$ cycle is faster than the $\overline{RD}$ cycle, the external memory needs to be fast enough to cater for the $\overline{PSEN}$ cycle.

3.9.4 The address latch enable (ALE)

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external program memory. For that purpose ALE is activated twice every machine cycle and this activation takes place even when the cycle involves no external fetch. The only time an ALE pulse does not come out is during an access to external data memory. The first ALE of the second cycle of a MOVX instruction is missing (see Fig. 3.17). Consequently, in any system that does not use external data memory, ALE is activated at a constant rate of 1/6 of the oscillator frequency, and can be used for external clocking or timing purposes.

3.10 Interrupts

The 8051 has five interrupt sources (the 8052 has six), each of which can be programmed to one of two priority levels. The five interrupt sources are listed below in order of priority:

Source	Description
$\overline{\text{INT0}}$	External request from P3.2 pin (sampled at S5P2 of every machine cycle).
Timer 0	Overflow from Timer 0 activates interrupt request flag TF0.
$\overline{\text{INT1}}$	External request from P3.3 pin (sampled at S5P2 of every machine cycle).
Timer 1	Overflow from Timer 1 activates interrupt request flag TF1.
Serial port	Completion of transmission or reception of one serial frame activates request flag TI, on transmission, or RI, on reception.
* Timer 2	Logical OR of EXF2 and TF2 flags generates interrupt. These flags must be cleared in software.

* 8052 version only

Each source can be individually enabled or disabled by setting or clearing a bit in SFR IE. All interrupt sources can also be globally enabled or disabled using bit $\overline{\text{EA}}$ in the interrupt enable register IE.

Each source can be programmed to a high-priority or a low-priority level by setting or clearing a bit in register IP. A low-priority interrupt can itself be interrupted by an interrupt of high-priority, but not by another of low-priority. A high-priority interrupt cannot be interrupted. To implement these rules, the interrupt system contains two non-addressable 'priority level active' flip-flops. One indicates that a high-priority interrupt is being serviced, and blocks all other interrupts. The other indicates that a low-priority interrupt is being serviced, and blocks all other low-priority interrupts. The interrupt system is detailed in Fig. 3.18.

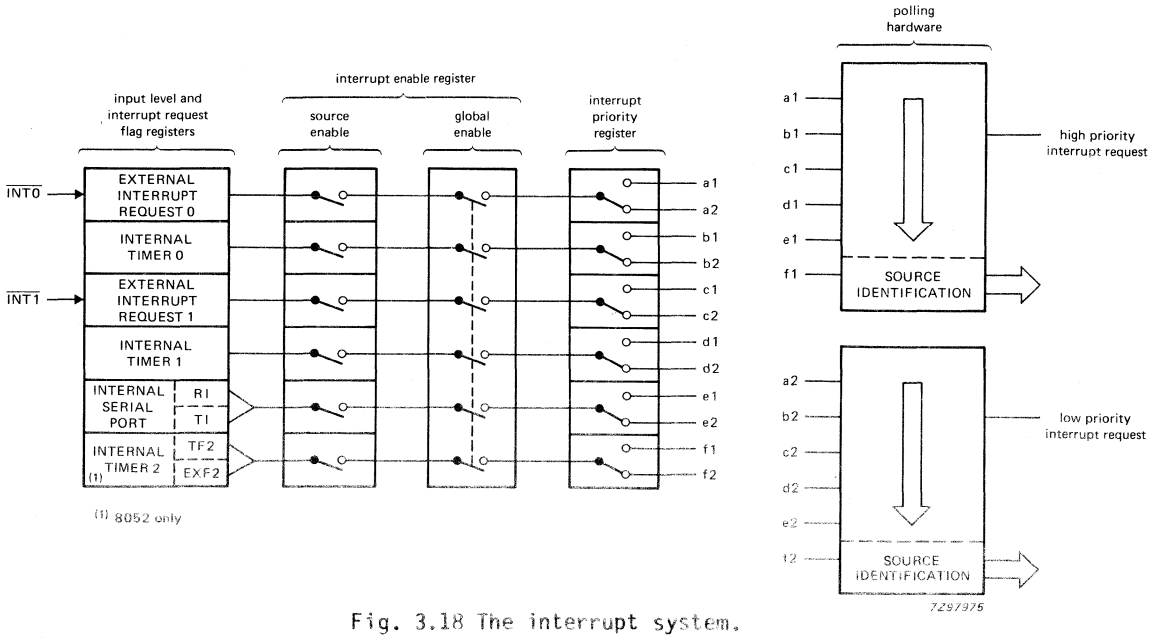


Fig. 3.18 The interrupt system.

In the event that requests of the same priority are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

Source	Priority within level
External interrupt 0	(highest)
Timer 0 overflow	
External interrupt 1	
Timer 1 overflow	
Serial port	(lowest 8051)
Timer 2	(lowest 8052)

All the interrupt sources are sampled at S5P2 of every machine cycle and polled during the next cycle, such that by S6 of any cycle all active interrupt requests have been found and their priority established. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate routine, provided this hardware generated LCALL is not blocked by any of the following conditions:

a) An interrupt of higher or equal priority level is already in progress.
b) The current machine cycle is not the final cycle in the execution of the instruction in progress. (In other words, no interrupt request will be responded to until the instruction in progress is completed).
c) The instruction in progress is RETI or an access to SFR IE or IP. (In other words, an interrupt request will not be responded to after RETI or after a read or write to IE or IP until at least one other instruction has been executed.)

If any of the above conditions exists, the result of the interrupt poll is ignored. If none of these conditions exists, the result of the interrupt poll is acted upon in the next machine cycle.

The polling cycle is repeated with every machine cycle, and the values polled will be the ones present at S5P2 of the previous machine cycle. Note, that if an interrupt flag is active but is not being responded to for one of the stated conditions, and if the flag is inactive when the blocking condition is removed, then the unfortunate interrupt will be ignored. In short, the fact that the interrupt flag was once active but not serviced is not 'remembered'. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in figure 3.19.

Note, that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in figure 3.19, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

The processor acknowledges an interrupt request by executing a hardware-generated LCALL to the correct service routine. In some cases it also clears the flag which generated the interrupt. It will not clear the serial port and, in the 8052, the timer 2 flags, this has to be done in users software. It clears an external interrupt flag, either (IE0 or IE1) only if it was transition-activated.

The LCALL pushes the contents of the program counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to as shown below:

Source	Address
External interrupt 0	0003H
Timer 0 overflow	000BH
External interrupt 1	0013H
Timer 1 overflow	001BH
Serial port	0023H
Timer 2 (8052)	002BH

Execution proceeds from that address until the RETI instruction is encountered.

The RETI instruction clears the 'priority level active' flip-flop that was set when this interrupt was acknowledged. It then pops the top two bytes from the stack and reloads the program counter. Execution of the interrupted program continues from where it left off.

3.10.1 External interrupt logic

The external interrupt sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT0 or IT1 in register TCON. If ITx = 0, external interrupt x is triggered by a detected LOW at the INTx pin and if ITx = 1, external interrupt x is edge-triggered. In the second case, if successive samples of the INTx pin show a HIGH in one cycle and a LOW in the next, the interrupt request flag IEx in TCON is set and it requests the interrupt.

Since the external interrupt pins are sampled once during every machine cycle, an input HIGH or LOW should be held for at least 12 oscillator periods to ensure that it is sampled. If the external interrupt is transition-activated, the request pin must be held HIGH for at least one cycle, and then held LOW for at least one cycle to ensure that the transition has been seen so that the interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. It then has to deactivate the request before the interrupt service routine is completed, otherwise another interrupt will be generated.

The $\overline{INT0}$ and $\overline{INT1}$ levels are latched into an internal holding register at S5P2 of every machine cycle but their values are not actually polled until the next machine cycle. If a request is active and conditions right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction executed. This call itself will take two cycles and therefore a minimum of three machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Fig. 3.19 shows the interrupt response timings.

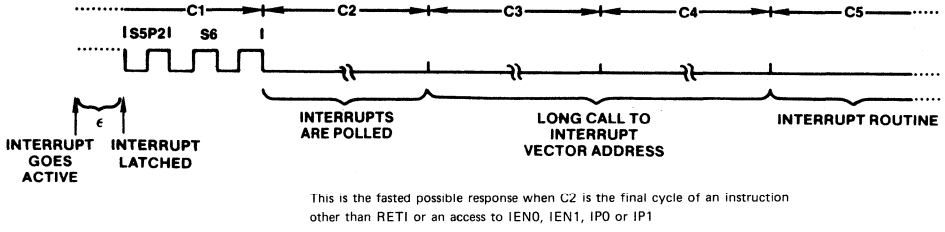


Fig. 3.19 The interrupt response timings.

A longer response time would result if the request was blocked by any of the three previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be any longer than 3 cycles, since the longest instructions are only 4 cycles long. If the instruction is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always no less than 3 cycles and no more than 8 cycles.

3.10.2 Interrupt control registers

The interrupt request flags are in two different registers and two port pins, as listed below:

Source	Request flag	Location
External interrupt 0	$\overline{INT0}$, if IT0 = 0	P3.2
	IE0, if IT0 = 1	TCON.1
Timer 0 overflow	TF0	TCON.5
External interrupt 1	$\overline{INT1}$, if IT1 = 0	P3.3
	IE1, if IT1 = 1	TCON.3
Timer 1 overflow	TF1	TCON.7
Serial port	TI (on transmission)	SCON.1
	RI (on reception)	SCON.2
Timer 2	TF2, EXF2	T2CON.7/6

If EXEN2 in T2CON = 0, TF2 is set on a Timer 2 16-bit overflow.

If EXEN2 in T2CON = 1, TF2 is again set on a Timer 2 overflow, but in addition EXF2 is set on a 0-to-1 transition at pin T2EX.

External interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All interrupt flags can be set or cleared by software, which gives the same result as if it had been carried out by hardware.
The enable and priority control registers are shown below. All these control bits are set or cleared by software. All are cleared by reset.

IE: Interrupt enable register

Bit:	7	6	5	4	3	2	1	0
	EA	X	ET2*	ES	ET1	EX1	ET0	EX0

* 8052 only

where

EA: disables all interrupts.If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is enabled or disabled by setting or clearing its enable bit.

ET2 enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.

ES: enables or disables the serial port interrupt. If ES = 0, the serial port interrupt is disabled.

ET1: enables or disables the Timer 1 overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.

EX1: enables or disables external interrupt 1. If EX1 = 0, external interrupt 1 is disabled.

ET0: enables or disables Timer 0 overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

EX0: enables or disables external interrupt 0. If EX0 = 0, external interrupt 0 is disabled.

IP: Interrupt priority register

Bit:

7	6	5	4	3	2	1	0
X	X	PT2*	PS	PT1	PX1	PT0	PX0

* 8052 only

where

PT2 defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.

PS: defines the serial port interrupt priority level. PS = 1 programs it to the higher priority level.

PT1: defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.

PX1: defines the external interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.

PT0: defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.

PX0: defines the external interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

3.10.3 Single step mode

One property of the 8051 interrupt structure enables single stepping with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority is still in progress, nor will it be responded to after RETI until at least one other instruction has been carried out. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program has been executed.

There are a number of ways that this feature can be used to single step the 8051. One way is to program one of the external interrupts, say INT0, to be level-activated. The service routine for this interrupt will then terminate with:

```
JNB     P3.2,$    ;WAIT HERE TILL INT0 GOES HIGH
JB      P3.2,$    ;NOW WAIT HERE TILL IT GOES LOW
RETI              ;GO BACK AND EXECUTE ONE INSTRUCTION
```

Now if the $\overline{\text{INT0}}$ pin, which is also the P3.2 pin, is held normally LOW, the CPU will go directly into the external interrupt 0 routine and stay there until $\overline{\text{INT0}}$ is pulsed (from LOW to HIGH to LOW). It will then execute RETI, return to the task program, execute one instruction, and immediately re-enter the external interrupt 0 routine to await the next pulse to P3.2. One step of the task program is executed each time P3.2 is pulsed.

3.11 Reset

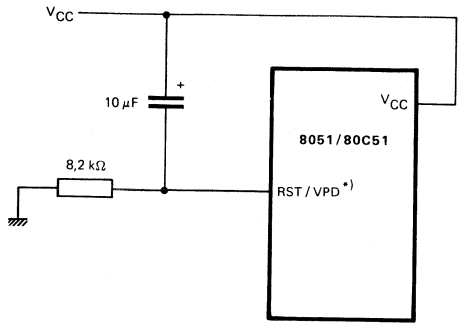
Reset is achieved by holding the RST pin HIGH for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset and it also configures the ALE and $\overline{PSEN}$ pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is HIGH and is repeated every cycle until RST goes LOW. It leaves the internal registers as follows:

Register	Content	Register	Content
PC	0000H	TMOD	00H
A	00H	TCON	00H
B	00H	TH0	00H
PSW	00H	TL0	00H
SP	07H	TH1	00H
DPTR	0000H	TL1	00H
P0-P3	0FFH	SCON	00H
IP	(XXX00000)	SBUF	Indeterminate
IP (8052)	(XX000000)	PCON	(0XXXXXXX)*
IE	(0XX00000)	T2CON (8052)	00H
IE (8052)	(0X000000)	TH2	00H
RCAP2H (8052)	00H	TL2	00H
RCAP2L (8052)	00H		

* (0XXX0000) for 80C51

The internal RAM is not affected by reset. When V_{CC} is turned on, the RAM content is indeterminate (see section 3.11.3).

An automatic reset at power-on can be obtained by connecting the RST pin to V_{CC} through a 10 µF capacitor and V_{SS} through an 8.2 kΩ resistor, provided that the V_{CC} risetime does not exceed 1 ms or so and that the oscillator start-up does not exceed 10 ms. A power-on reset circuit is shown in Fig. 3.20. When the power comes on, the current drawn by RST commences to charge up the capacitor. The voltage at RST is the difference between V_{CC} and the capacitor voltage, and decreases from V_{CC} as the capacitor charges up. The larger the capacitor, the more slowly V_{RST} decreases. V_{RST} must remain above the lower threshold of the Schmitt trigger long enough to effect a complete reset. The time required is the oscillator start-up time plus two machine cycles. If the V_{CC} risetime is less than 1 ms, a 10 µF capacitor will provide a reliable power-on reset.



* This pin termed RST in PCB80C51

Fig. 3.20 Power-on reset circuitry.

3.11.1 The RST/VPD pin of the MAB8051/8031

The circuitry connected to the RST/VPD pin is shown in Fig. 3.21. A Schmitt trigger is used at the input to the reset circuitry for noise rejection. The output of the Schmitt trigger is sampled by the reset circuitry at S5P2 of every machine cycle. At least two consecutive samples must show a HIGH in order to effect a complete reset and initialization.

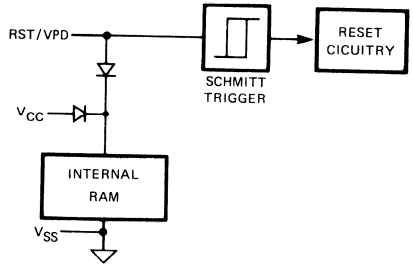


Fig. 3.21 RST/VPD circuitry

3.11.2 The RST pin of the PCB80C51/80C31

In the CMOS versions, the pin termed the RST/VPD(*) in the NMOS versions is just called RST. This is because in the PCB80C51/80C31, the power-down function, see 3.12.2, has been transferred to the VCC pin. It follows therefore, that this pin for the CMOS versions only serves as the reset pin.

The nature of the internal RST pull-down differs between the MAB8051/8031 and the PCB80C51/80C31. Both device types have an internal pull-down on the pin to implement the power-on reset feature. In the NMOS versions the internal pull-down is an nFET, and is therefore more accurately modeled as a current sink than a resistor. In the CMOS versions the internal pull down is a diffused resistor.

* V_{PD} option for MAB8051AH only available upon request

3.11.3 Power-down operation in the MAB8051/8031

During normal operation the internal RAM draws its power from V_{CC}. However, as can be seen from Fig. 3.21, if the voltage at RST/VPD exceeds V_{CC} then it becomes the source of power for the RAM. This allows a backup power supply to be used to hold RAM data in the event of a power failure.

To take advantage of this feature, the user's system, upon detecting that a power failure is imminent, would interrupt the processor via $\overline{INT0}$ or $\overline{INT1}$ to transfer relevant data to the RAM and enable the backup power supply to the RST/VPD pin before V_{CC} falls below its operating limit. When power returns, VPD needs to stay on long enough to effect a complete reset (oscillator startup time plus 2 machine cycles), and then normal operation can resume.

Fig. 3.22 suggests one possible implementation of the power-down feature. Assuming imminent power failure detection interrupts the processor via INT0, the external interrupt 0 service routine transfers relevant data to the RAM and then writes a 0 to P1.0. The LOW at P1.0 triggers the 555, which is configured as a one-shot whose output pulse width depends on RC and the presence of V_{CC}. If V_{CC} is still present when the 555 times out, it is assumed that the 'imminent power failure' was a false alarm, and the operation resumes from reset. If V_{CC} does in fact go down before the 555 times out, the 555 will hold power to the RST/VPD pin during the outage, and will continue to hold it after V_{CC} comes back, for a time determined by R and C. R and C should be selected so as to obtain a reliable power-on reset.

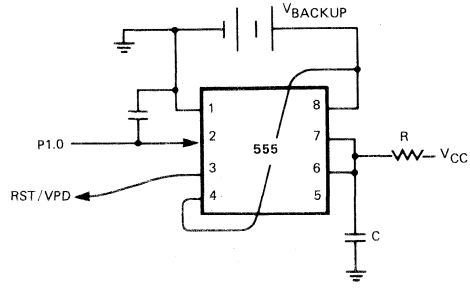


Fig. 3.22 Power-down circuit.

3.12 The idle and power-down modes in the PCB80C51/80C31

Fig. 3.23 gives the internal idle and power-down configuration. As illustrated, power-down mode freezes the oscillator, idle mode operation allows the interrupt, serial port, and timer blocks to continue to be clocked, but halts the clock signal to the CPU. These modes are activated by software control via the special function register, PCON. Its hardware address is 87H and PCON is not bit addressable.

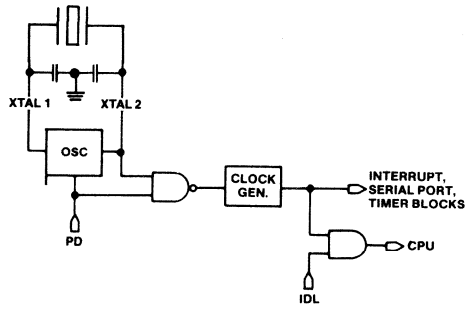


Fig. 3.23 Idle mode and power-down circuitry.

PCON: Power control register

Bit:	7	6	5	4	3	2	1	0
	SMOD	-	-	-	GF1	GF0	PD	IDL

where

SMOD: is used to double the baud rate whatever the operating mode of the serial port. When SMOD is set the baud rate is doubled.

-: reserved bits.

GF1: is the general purpose flag 1 bit.

GF0: is the general purpose flag 0 bit.

PD: is the power-down bit. The power-down mode is activated when this bit is set.

IDL: is the idle mode bit. The idle mode is activated when this bit is set.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

3.12.1 The idle mode

The instruction that sets PCON.0 is the last instruction in the normal operating mode before the idle mode is activated. Once in the idle mode, the internal clock signal to the CPU is turned off, while the interrupt, timer, and serial port functions continue to be clocked. During idle, the CPU status is preserved in its entirety: the stack pointer, program counter, PSW, accumulator and all other registers hold their data. The port pins also maintain the logical states they had when the idle mode was activated. ALE and $\overline{PSEN}$ are held at logic HIGH levels.

There are two ways to exit from the idle mode. Activation of any enabled interrupt will cause the hardware to clear PCON.0, terminating the idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into the idle mode.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or in the idle mode. For example, an instruction that activates the idle mode can also set one or both flag bits. On exit from the idle mode via an interrupt, the interrupt service routine can examine the flag bits.

The other way to leave the idle mode is by a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

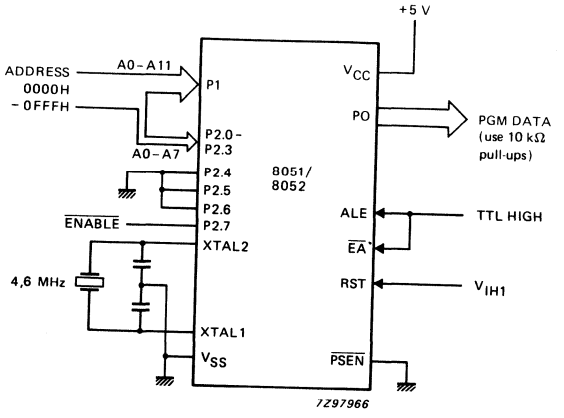
3.12.2 The power-down mode

The instruction that sets PCON.1 is the last instruction executed in the normal operating mode before the power-down mode is activated. Once in the power-down mode, the on-chip oscillator is stopped. This freezes all functions; only the on-chip RAM is held. The special function registers are not saved. The only exit from the power-down mode is via a hardware reset.

In the power-down mode, V_{CC} can be reduced to minimize power consumption. Care must be taken, however, to ensure that V_{CC} is not reduced before the power-down mode is activated, and that V_{CC} is fully restored before exit from the power-down mode. The reset that provides exit from the power-down mode also frees the oscillator. The reset should not be activated before V_{CC} is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 ms).

3.13 Program verification

Fig. 3.24 gives the arrangement for program verification of the 8051. To read the program memory of the 8051, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of the program memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and $\overline{PSEN}$ are held LOW, while the ALE, RST and $\overline{EA}$ pins are held HIGH. (These are TTL levels except RST, which requires 2,5 V for a HIGH.) Port 0 will be the data output lines and P2.7 can be used as the read strobe. While P2.7 is held at TTL HIGH, the Port 0 pins float. When P2.7 is strobed LOW, the contents of the addressed location will appear at Port 0. External pull-ups (e.g.,10 kΩ) are required on Port 0 during this operation.



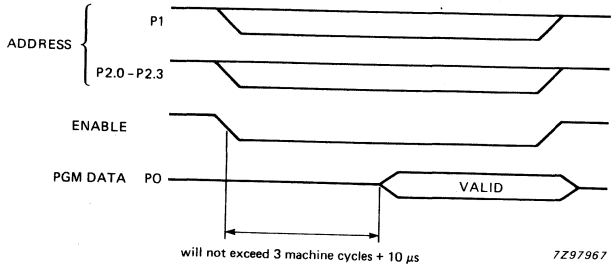


Fig. 3.24 Program verification.

3.14 NMOS and CMOS versions of the on-chip oscillator

3.14.1 NMOS versions

The on-chip oscillator circuitry for the NMOS members of the 8051/52 family is a single-stage linear inverter (Fig. 3.25) and is intended for use as a crystal-controlled, positive reactance oscillator (Fig. 3.26). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with a capacitance external to the crystal. The capacitance values (C1 and C2 in Fig. 3.26) are not critical, 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator may be used in place of the crystal in cost-effective applications. If ceramic resonators are used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

When driving NMOS oscillator circuits with an external clock source, apply the external clock signal to XTAL2 and ground XTAL1 as shown in Fig. 3.27 A pull-up resistor may be used (increasing noise margin), but is optional if V_{OH} of the driving gate exceeds the $V_{IH\ (min)}$ characteristic of XTAL2.

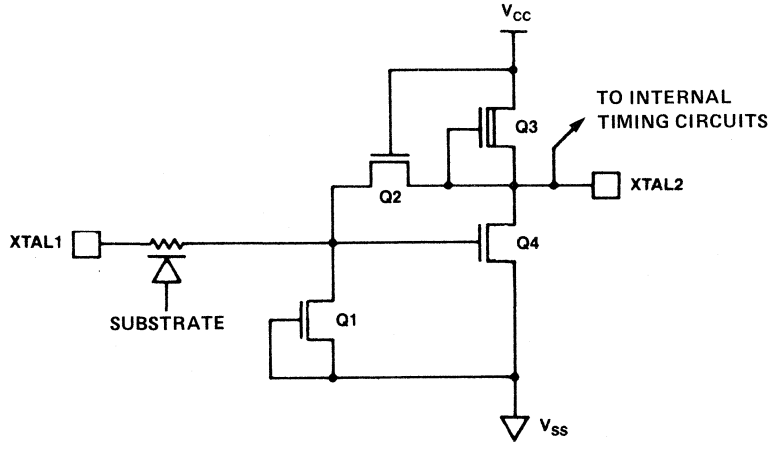


Fig. 3.25 On-chip oscillator circuitry in NMOS versions.

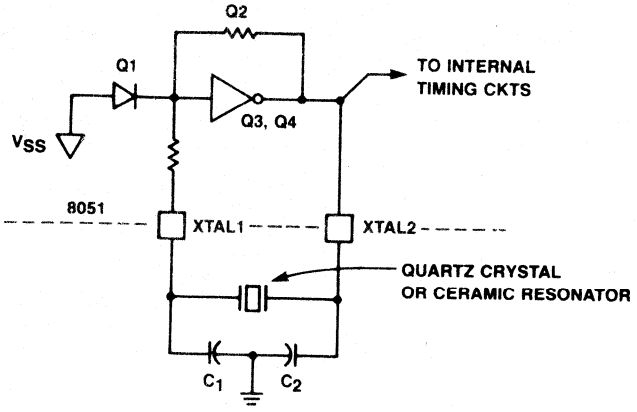


Fig. 3.26 A typical NMOS oscillator circuit.

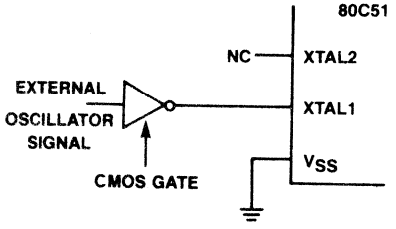


Fig. 3.27 Driving the NMOS parts with an external clock source.

3.14.2 CMOS versions

The on-chip oscillator circuitry shown in Fig.3.28, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the previous NMOS description. However there are important differerences to note:

- The 80C51 is able to turn off its oscillator under software control by writing a 1 to the PD bit in PCON. Another difference is that, in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas the NMOS version is driven by the signal at XTAL2.

- The feedback resistor R_f in Fig. 3.28 consists of paralleled n-and p-channel FETs controlled by the PD bit, such that R_f is open circuit when PD = 1. The ESD protection diodes D1 and D2 act as clamps to V_{CC} and V_{SS} and are parasitic to the R_f FETs. The oscillator can be used with the same external components as the NMOS versions, as shown in Fig. 3.29. Typically C1 = C2 = 30 pF when the feedback element is a quartz crystal, and C1 = C2 = 47 pF when a ceramic resonator is used.

  To drive the CMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave the XTAL2 connection floating as shown in Fig 3.30.

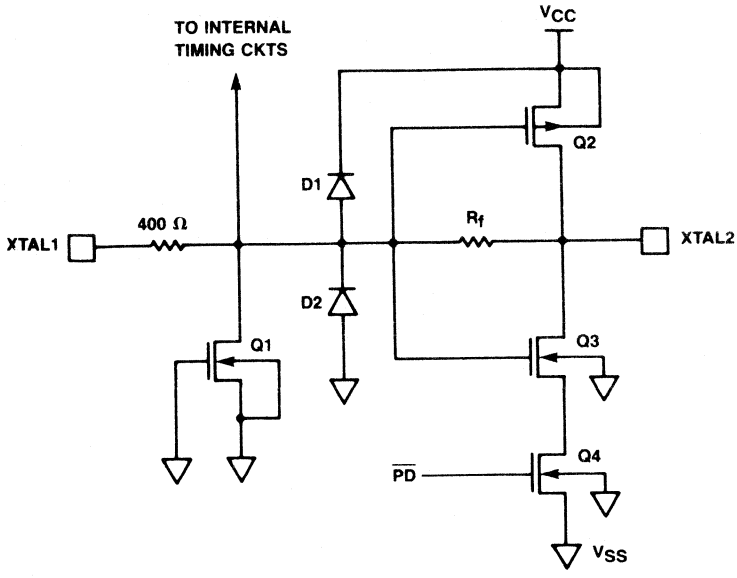


Fig. 3.28 On chip oscillator circuitry in CMOS versions.

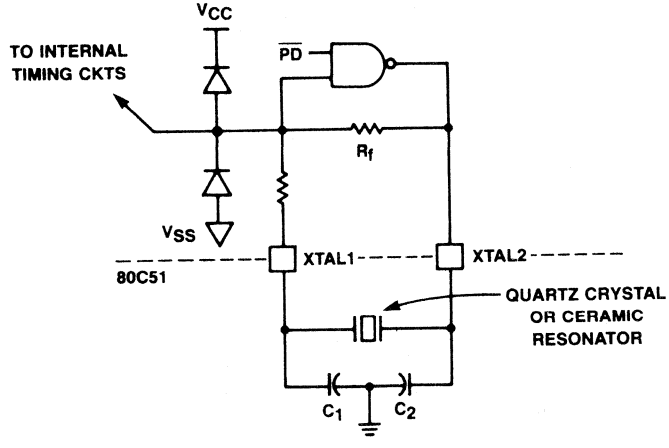


Fig. 3.29 A typical CMOS oscillator circuit.

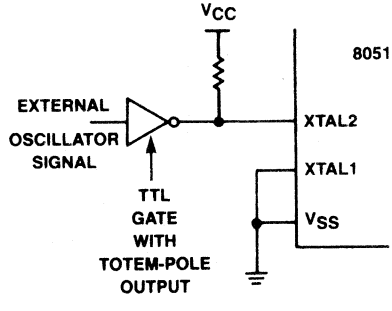


Fig. 3.30 Driving the CMOS parts with an external clock source.

4.0 MEMORY, ORGANIZATION, ADDRESSING MODES AND DATA MANIPULATION

4.1 Memory organization

In the 8051 family the memory is organized in three separate addressable spaces. These memory spaces are shown in Fig. 3.1 and Fig. 3.2.

- 64K-byte program memory address space
- 64K-byte external data memory address space
- 384-byte internal data memory address space

The 16-bit program counter register provides the 8051 with its 64 K addressing capabilities. The program counter permits the user to execute calls and branches to any location within the program memory space. There are no instructions that permit program execution to move from the program memory space to any of the data memory spaces.

In the MAB8051 and PCB80C51 the lower 4 K (8 K in the 8052) of the 64 K program memory address space is filled by internal ROM. By tying the $\overline{EA}$ pin HIGH, the processor is forced to fetch all program memory addresses i.e. 0 to 4095 (0 to 8191 in the 8052) from the internal ROM. Bus expansion for accessing program memory from 4 K upwards is automatic since external instruction fetches occur automatically when the program counter exceeds 4095 (8191 in the 8052). If the $\overline{EA}$ pin is tied LOW all program memory fetches are from external memory. The execution speed of the 8051 is the same regardless of whether fetches are from external or internal program memory. If all program storage is on-chip, then byte location 4095 should be left vacant to prevent an undesired prefetch from external program memory address 4096.

Certain locations in program memory are reserved for specific programs. Locations 0000 to 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 to 0042 (0051 for 8052) are reserved for the five (six for 8052) interrupt-request service routines. Each resource which can request an interrupt requires that its service routine is started in its reserved location.

The 64 K-byte external data memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the internal data memory is the most flexible of the address spaces. The internal data memory space is subdivided into a 256-byte internal data RAM address space and a 128-byte special function register (SFR) address space, as shown in Fig. 4.1.

The internal data RAM address space is 0 to 255. Four 8-register banks occupy locations 0 to 31 and the stack can be located anywhere in the remaining bytes of the internal data RAM address space. Reset initializes the stack pointer to location 07H and is incremented once to start from location 08H which is the first register R0 of the second register bank. Thus, if more than one register bank is used, the SP should be initialized to a different location of the RAM which is not being used for data storage. In addition, 128 bit locations of the internal data RAM are accessible through direct addressing. These bits reside in 16 bytes of internal data RAM at locations 32 to 47. At present, locations 0 to 127 of the internal data RAM address space are filled with on-chip RAM. The stack depth is limited only by the available internal data RAM, due to an 8-bit reloadable stack pointer.

The stack is used for storing the program counter during subroutine calls and may be used for passing parameters. Any byte of internal data RAM or SFR accessible through direct addressing can be pushed/popped.

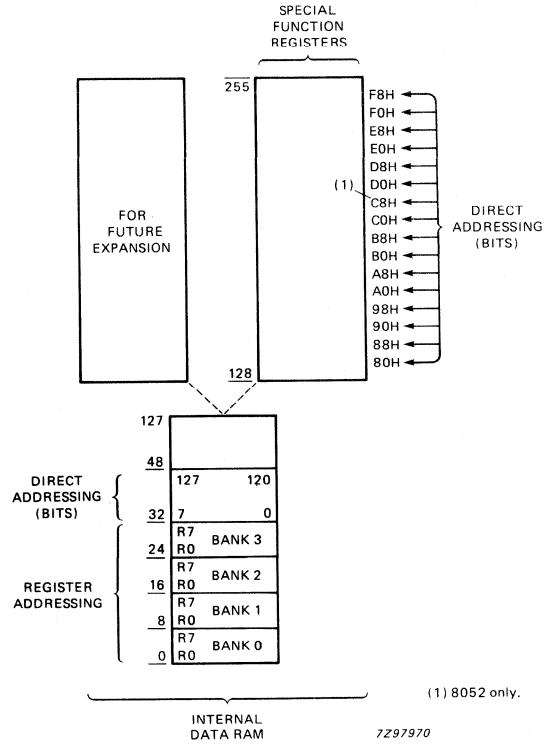


Fig. 4.1 Internal data memory address space.

The SFR address space is 128 to 255. All registers except the program counter and the four 8-register banks reside in this address space. Memory mapping of the SFRs allows them to be accessed as easily as internal RAM and as such, they can be operated on by most instructions. The twenty one SFRs, twenty six in the 8052 are listed in Fig. 4.2 and their mapping in the SFR address space is shown in Fig. 4.3 and Fig. 4.4.

symbol	name	address	contents after reset
ACC*	accumulator	0E0H	00H
B*	B register	0F0H	00H
PSW*	program status word	0D0H	00H
SP	stack pointer	81H	07H
DPTR	data pointer		
	DPH	83H	0000H
	DPL	82H	0000H
P0*	port 0	80H	0FFH
P1*	port 1	90H	0FFH
P2*	port 2	0A0H	0FFH
P3*	port 3	0B0H	0FFH
IP*	interrupt priority control	0B8H	XX000000B
IE*	interrupt enable control	0A8H	0X000000B
TMOD	timer/counter mode control	89H	00H
T2CON*	timer/counter 2 control	0C8H	00H
TCON	timer/counter control	88H	00H
TH0	timer/counter 0 (high byte)	8CH	00H
TL0	timer/counter 0 (low byte)	8AH	00H
TH1	timer/counter 1 (high byte)	8DH	00H
TL1	timer/counter 1 (low byte)	8BH	00H
TH2	timer/counter 2 (high byte)	0CDH	00H
TL2	timer/counter 2 (low byte)	0CCH	00H
RCAP2H	timer/counter 2 capture register (high byte)	0CBH	00H
RCAP2L	timer/counter 2 capture register (low byte)	0CAH	00H
SCON*	serial control	98H	00H
SBUF	serial data buffer	99H	indeterminate
PCON	power control	87H	0XXXXXXXB

Where

H = Hexadecimal
B = Binary

* Registers are both byte and bit addressable.

Fig. 4.2 Special function registers.

Performing a read from a location in the internal data memory where neither a byte of internal data RAM nor an SFR exists will access data of indeterminate value.

Architecturally, each memory space in a linear sequence of 8-bit wide bytes. The storage of multi-byte address and data operands in program and data memories is with the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte Z, the most significant bit is represented by Z.7 while the least significant bit is Z.0. Any deviation from these conventions will be explicitly stated in the text.

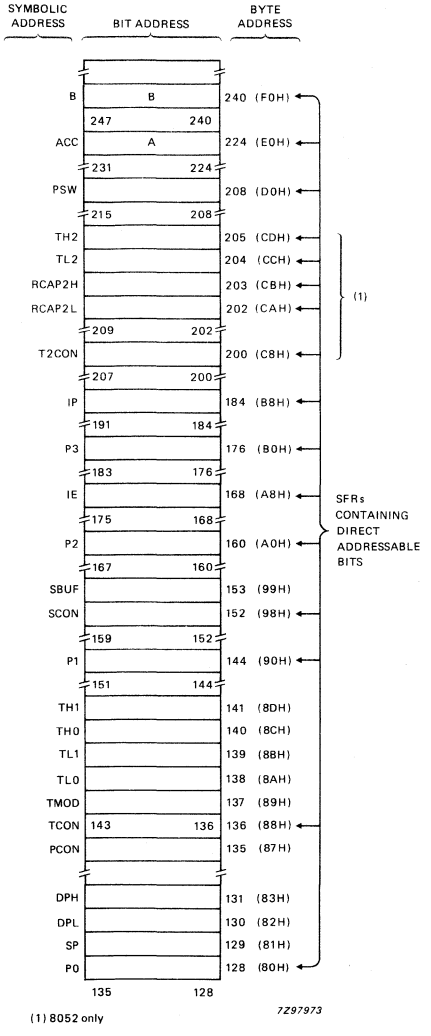


Fig. 4.3 Mapping of special function registers.

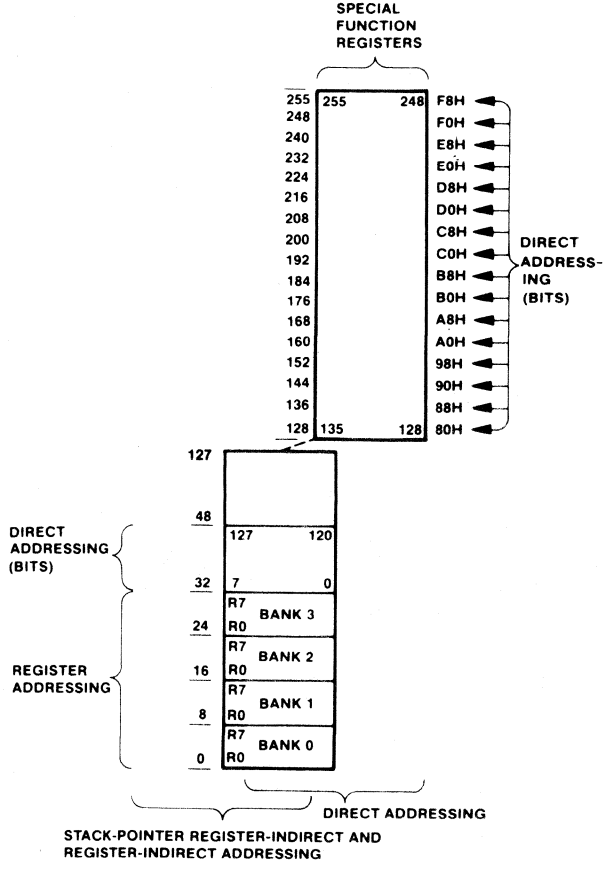


Fig. 4.4 Special function register bit address.

4.2 Operand addressing

There are five methods of addressing source operands. They are register addressing, direct addressing, register-indirect addressing, immediate addressing and base-register- plus index-register- indirect addressing. The first three of these methods can be used to address a destination operand. Since operations in the 8051 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing modes are used in combinations to provide the 8051 with its 21 addressing sub-modes.

Most instructions have a 'destination, source' field that specifies the data type, addressing modes and operands involved. For operations other than moves, the destination operand is also the source operand. For example, in 'subtract-with-borrow A,#5' the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in the internal data memory. The selection of the program memory space or external data memory space for a second operand is determined by the mnemonic unless it is an immediate operand. The subset of the internal data memory being addressed is determned by the addressing mode and the address value. For example, the SFRs can be accessed only through direct addressing with an address of 128 to 255. A summary of the operand addressing modes is given in Fig. 4.5. The following paragraphs describe the five addressing modes.

- **Register Addressing**
  - **R7-R0**
  - **A,B,C (bit), AB (two bytes), DPTR (double byte)**
- **Direct Addressing**
  - **Lower 128 bytes of Internal Data RAM**
  - **Special Function Registers**
  - **128 bits in subset of Internal Data RAM address space**
  - **128 bits in subset of Special Function Register address space**
- **Register-Indirect Addressing**
  - **Internal Data RAM [@R1, @R0, @SP (PUSH and POP only)]**
  - **Least Significant Nibbles in Internal Data RAM (@R1, @R0)**
  - **External Data Memory (@R1, @R0, @DPTR)**
- **Immediate Addressing**
  - **Program Memory (in-code constant)**
- **Base-Register- plus Index-Register-Indirect Addressing**
  - **Program Memory (@ DPTR + A, @ PC + A)**

Fig. 4.5 Operand addressing modes.

4.2.1 Register addressing

Register addressing permits access to the eight registers (R7-R0) of the selected register bank (RB). One of the four 8-register banks is selected by a two-bit field in the PSW. The registers may also be accessed by direct and register-indirect addressing, since the four register banks are mapped onto the lowest 32 bytes of the internal data RAM as shown in Fig. 4.6 and Fig. 4.7. Other internal data memory locations that are addressed as registers are A, B, C, AB and DPTR.

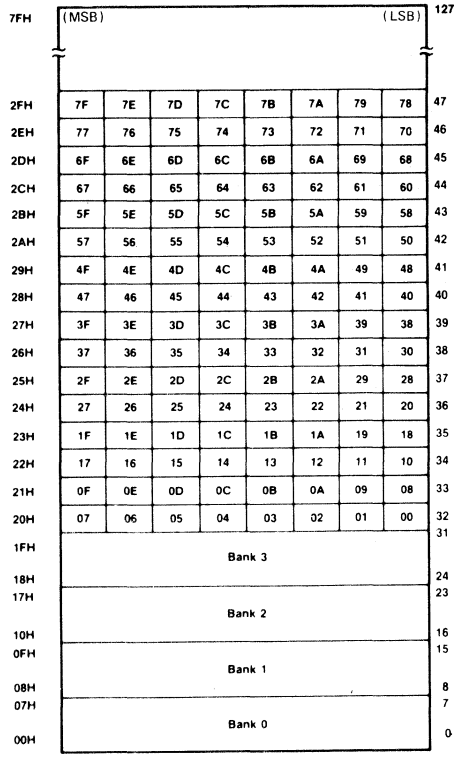


Fig. 4.6 RAM bit addresses.

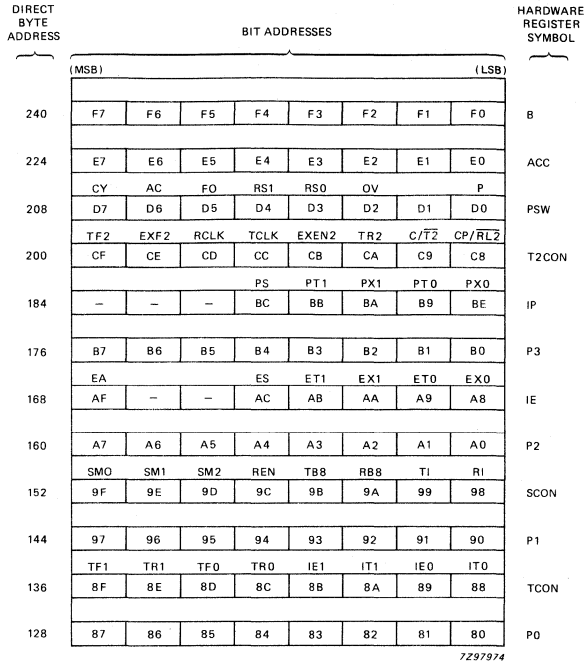


Fig. 4.7 Special function register bit address.

4.2.2 Direct addressing

Direct addressing provides the only means of accessing the memory-mapped byte-wide SFRs and memory-mapped bits within the SFRs and internal data RAM. Direct addressing of bytes may also be used to access the lower 128 bytes of internal data RAM. Direct addressing of bits gains access to a 128 bit subset of the internal data RAM and a 128 bit subset of the SFRs as shown in Figs. 4.3, 4.4, 4.6, and 4.7. Further changes to the internal data RAM from address 128 to 255 can only be done using indirect addressing. This address area lies parallel to the SFR range of addresses which uses direct addressing only.

4.2.3 Register-indirect addressing

Register-indirect addressing uses the contents of R1 or R0 in the selected register bank, or the content of the stack pointer (PUSH and POP only), as the address of a required operand or data byte. Register-indirect addressing is also used for accessing the external data memory. In this case, either R1 or R0 in the selected register bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit data pointer may be used for accessing any location within the full 64 K external address space.

4.2.4 Immediate addressing

Immediate addressing enables numeric constants which are part of the instruction, to be accessed from the program memory.

4.2.5 Base-register- plus index-register- indirect addressing

Base-register- plus index-register- indirect addressing simplifies the accessing of look-up-tables (LUT) resident in the program memory. A byte may be accessed from a LUT via an indirect move from a location whose address is the sum of a base register (the DPTR or PC) and the index register (A).

4.3 Data manipulation

The 8051 is a controller and an arithmetic processor. In addition to the capabilities of its 8048 predecessor, the 8051 was enhanced with improved data transfer, logic manipulation, arithmetic processing, and real-time control capabilities.

The 8051 performs operations on bit, nibble (4-bit), byte (8-bit) and double-byte (16-bit) data types. It is classified as an 8-bit machine since the internal ROM, RAM, SFRs, arithmetic logic unit (ALU) and the external data bus are each 8-bits wide. The double-byte data type is used only by the data pointer and the program counter. The data pointer can be manipulated as a single double-byte register (DPTR) or as two locations in the internal data memory (DPH and DPL). The program counter is always manipulated as a single double-byte register.

4.4 Boolean processor

Although the Boolean processor is an integral part of the 8051's architecture, it may be considered an independent bit processor since it has its own instruction set, accumulator (the carry flag), and bit-addressable RAM and I/O.

The bit-manipulation instructions allow the direct addressing of 128 bits within the internal data RAM and 128 bits within the SFRs. The SFRs with an address evenly divisible by eight (P0, TCON, P1, SCON, P2, IEC, P3, IPC, PSW, A, and B) contain direct addressable bits. The Boolean processor can perform the bit operations of set, clear, complement, jump-if-set, jump-if-not-set, jump-if-set-then-clear and move to/from carry on any addressable bit. Between any addressable bit (or its complement) and the carry flag it can perform the operation of logical AND or logical OR with the result returned to the carry flag.

The bit-manipulation instructions provide optimum code and speed efficiency in applications such as the control of the 8051's on-chip peripherals. The Boolean processor also provides a straightforward means of converting logic equations (like those used in random logic design) directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking and test-and-branch trees.

4.5 Data transfer operations

Look-up tables resident in the program memory can be accessed by indirect moves. A byte constant can be transferred to the A register (the accumulator) from the program memory location whose address is the sum of the base register (the PC or DPTR) and the index register (A). This provides a convenient means for programming translation algorithms such as ASCII to seven segment conversions. The program memory move operations are shown diagrammatically in Fig. 4.8.

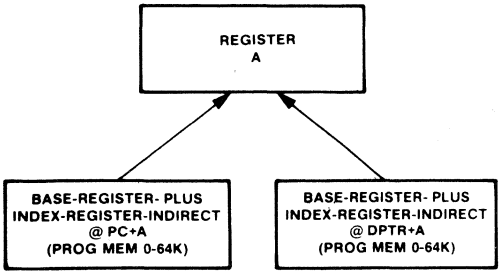


Fig. 4.8 Program memory move operations.

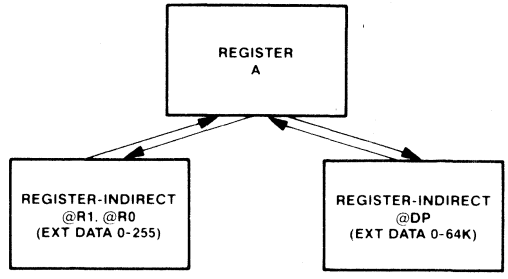


Fig. 4.9 External data memory move operations.

A byte location within a 256-byte block of external data memory can be accessed using R1 or R0 in register-indirect addressing. Any location within the full 64 K external data memory address space can be accessed through register-indirect addressing using a 16-bit base register (the data pointer). These moves are shown in Fig. 4.9.

The byte in-code-constant (immediate) moves and byte variable moves in the 8051 are detailed in Fig. 4.10. When one considers that the accumulator and the registers in the register banks can be directly addressed, the two-operand data transfer operations allow a byte to be moved between any two of the RB registers, internal data RAM, accumulator and SFRs. Also, immediate operands can be moved to any of these locations. Of particular interest is the direct address to direct address move which permits the value in a port to be moved to the internal data RAM without using any RB registers or the accumulator. The data pointer register can be loaded with a double-byte immediate value. The 8051's Boolean processor can also move any direct addressed bit to or from the carry register.

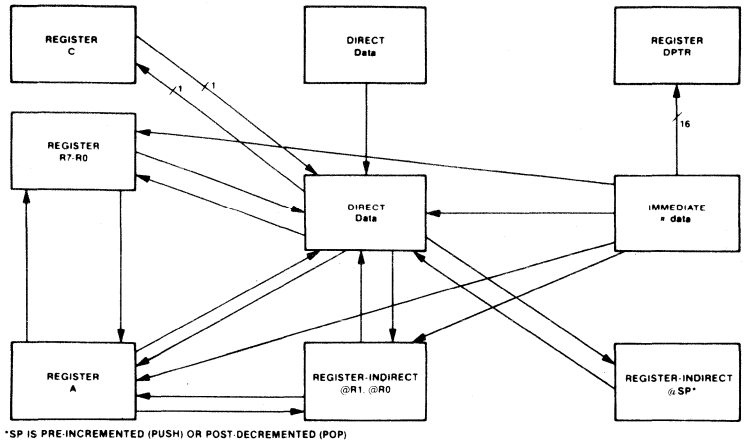


Fig. 4.10 Internal data memory move operations.

The A register can be exchanged with a register in the selected register bank, with a register-indirect addressed byte in the internal data RAM or SFR. The least significant nibble of the A register can also be exchanged with the least significant nibble of a register-indirect addressed byte in the internal data RAM. The exchange operation is shown in Fig. 4.11.

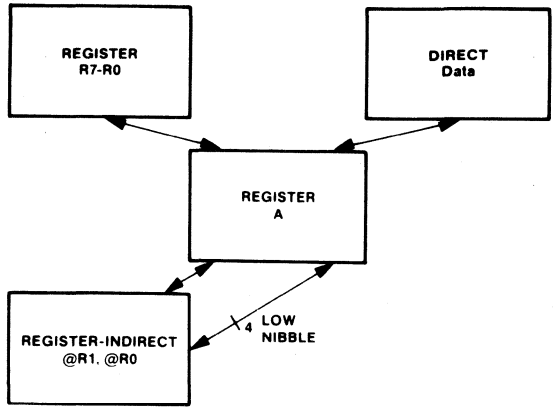


Fig. 4.11 Internal data memory exchange operations.

4.6 Logic operations

The 8051 permits the logic operations of AND, OR, and exclusive-OR to be performed on the A register by a second operand which can be an immediate value, a register in the selected register bank, a register-indirect addressed byte of internal data RAM, a direct addressed byte of internal data RAM or an SFR. In addition, these logic operations can be performed on a direct addressed byte of the internal data RAM or an SFR using the A register as the second operand. Also, the use of immediate addressing with direct addressing permits these logic operations to set, clear or complement any bit anywhere within the internal RAM or SFRs without affecting the PSW, RB registers or the accumulator. When one considers that registers R7-R0 and the accumulator can be directly addressed, two-operand logic operations allow the destination (first operand) to be a byte in the internal data RAM, an SFR, RB registers (R7-R0) or the accumulator while the source (second operand) can be any of those given above or an immediate value. The 8051 can also perform a logical OR, or a logical AND, between the Boolean accumulator (the carry flag) and any bit that can be accessed through direct addressing. The AND, OR, and exclusive-OR logic operations are summarized in Fig. 4.12

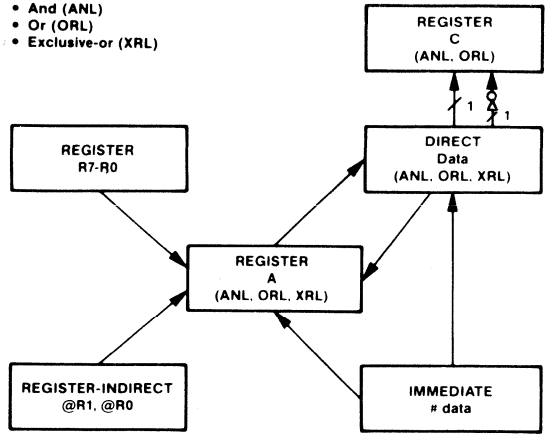


Fig. 4.12 Internal data memory logic operations.

In addition to the logic operations that are performed on the internal data memory as shown in Fig. 4.12, there are also logic operations that are performed specifically on the A register. These are listed in Fig. 4.13.

In addition to the AND and OR bit logic operations shown in Fig. 4.12, there are logicals that operate solely on a direct addressed bit. These operations are detailed in Fig. 4.14. The carry flag is also addressed as a register and can be set, cleared or complemented with one byte instructions.

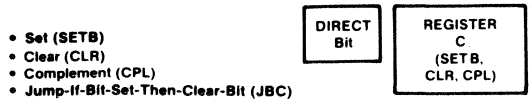


Fig. 4.13 Internal data memory logic operations (Register A specific).

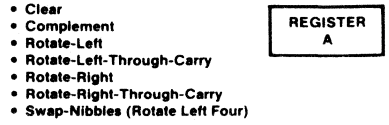


Fig. 4.14 Internal data memory logic operations (Bit-specific).

4.7 Arithmetic operations

Together with the existing 8048 arithmetic operations of add, increment, decrement, compare-to-zero, decrement-and-compare-to-zero, and decimal-add-adjust, the 8051 implements subtract-with-borrow, compare, multiply and divide.

Only unsigned binary integer arithmetic is performed in the arithmetic logic unit (ALU). In the two operand operations of add, add-with-carry and subtract-with-borrow, the A register is the first operand and receives the result of the operation. The second operand can be an immediate byte, a register in the selected register bank, a register-indirect addressed byte or a direct addressed byte. These instructions affect the overflow, carry, auxiliary-carry and parity flags in the PSW. The carry flag facilitates non-signed integer and multi-precision addition and subtraction and multi-precision rotation. Handling two's-complement-integer (signed) addition and subtraction can easily be accomodated with software monitoring of the PSW's overflow flag. The auxiliary-carry flag simplifies BCD arithmetic.

An operation that has an arithmetic aspect similar to a subtract is the compare-and-jump-if-not-equal operation. This operation performs a conditional jump if a register in the selected register bank, or an indirect addressed byte of the internal data RAM, does not equal an immediate value; or if the A register does not equal a byte in the direct addressable internal data RAM, or an SFR. Although the destination operand is not updated and neither source operand is affected by the compare operation, the carry flag is affected. A summary of the two-operand add/subtract operations is given in Fig. 4.15.

There are three arithmetic operations that operate exclusively on the A register, decimal-adjust for BCD addition and the two test conditions shown in Fig. 4.16. The decimal-adjust operation converts the result from a binary addition of two two-digit BCD values to yield the correct two-digit BCD result. During this operation the auxiliary-carry flag helps effect the proper adjustment. Conditional branches may be taken based on the value in the A register being zero or non-zero.

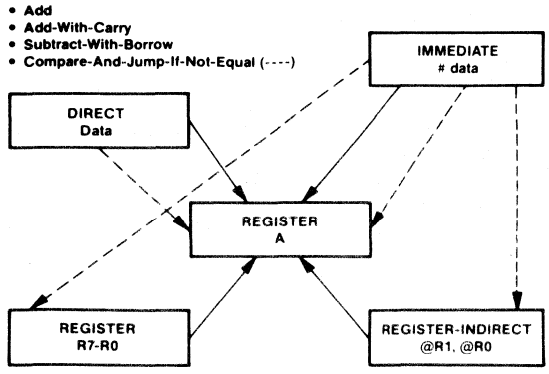


Fig. 4.15 Internal data memory arithmetic operations.

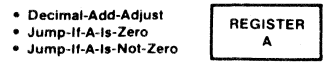


Fig. 4.16 Internal data memory arithmetic operations (Register A specific).

The 8051 simplifies the implementation of software counters since the increment and decrement operations can be performed on the A register, a register in the selected register bank, an indirectly addressed byte in the internal data RAM or a byte in the directly addressed internal data RAM or SFR. The 16-bit data pointer can be incremented. For efficient loop control the decrement-and-jump-if-not-zero operation is provided. This operation can test a register in the selected register bank, an SFR or any byte in the internal RAM accessible through direct addressing and force a branch if it is not zero. The increment/decrement operations are summarized in Fig. 4.17.

The multiply operation multiplies the one-byte A register by the one-byte B register and returns a double-byte result (high byte in B, low byte in A). The divide operation divides the one-byte A register by the one-byte B register and returns a byte quotient to the A register and a byte remainder to the B register. These are shown in Fig. 4.18.

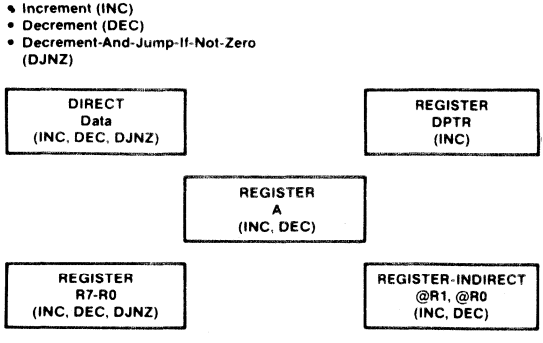


Fig. 4.17 Internal data memory arithmetic operations.

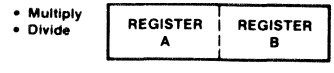


Fig. 4.18 Internal data memory arithmetic operations (Register A with B specific).

4.8 Control transfer

The 8051 has a non-paged program memory to accomodate relocatable code. The advantage of a non-paged memory is that a minor change to a program that causes a shift of the code's position in memory will not result in boundary readjustments being necessary. This also makes relocation possible. Relocation is desirable since it permits several programmers to write relocatable modules in various assembly and high-level languages which can later be linked together to form the machine object code.

Sixteen-bit jumps and calls are provided to allow branching to any location in the contiguous 64 K program memory address space and avoid program memory bank switching. Eleven-bit jumps and calls are also provided to maintain compatibility with the 8048 and to provide an efficient jump within a 2 K program module. Unlike the 8048, the 8051's call operations do not push the PSW onto the stack along with the program counter, since many subroutines written for the 8051 do not affect the PSW. The 8051 return operations therefore, only pop the program counter. The 8051's branch, return and call operations are shown diagrammatically in Figs. 4.19, 4.20, and 4.21, respectively.

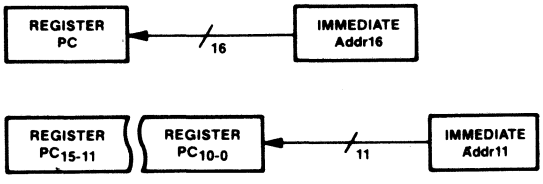


Fig. 4.19 Unconditional branch operations.

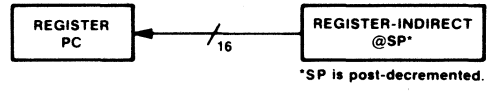


Fig. 4.20 Return operations.

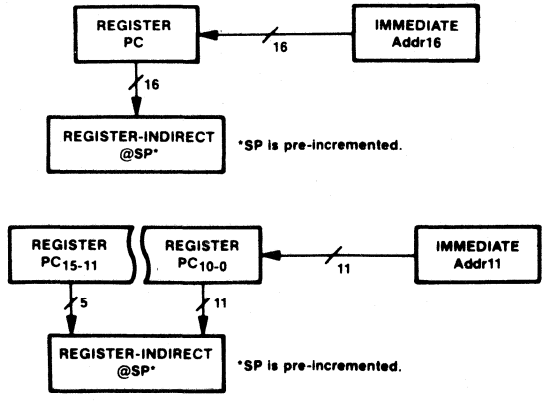


Fig. 4.21 Call operations.

The 8051 also provides a method for performing conditional and unconditional branching relative to the the starting address of the next instruction (PC-128 to PC+127). The bit test operations allow a conditional branch to be taken on the condition of a direct addressed bit being set or not set. The accumulator test operations allow a conditional branch based on the state of the accumulator being zero or non-zero. Also provided are compare-and-jump-if-not-equal and decrement-and-compare-to-zero operations. These are shown in Fig. 4.22.

The register-indirect jump in the 8051 permits branching relative to a base register (DPTR) with an offset provided by the non-signed integer value in the index register (A). This accommodates N-way branching. The indirect jump is shown in Fig. 4.23.

- Short Jump
- Jump-If-Bit-Set
- Jump-If-Bit-Not-Set
- Jump-If-Bit-Set-Then-Clear-Bit
- Jump-If-A-Zero
- Jump-If-A-Not-Zero
- Decrement-And-Jump-If-Not-Zero
- Compare-And-Jump-If-Not-Equal

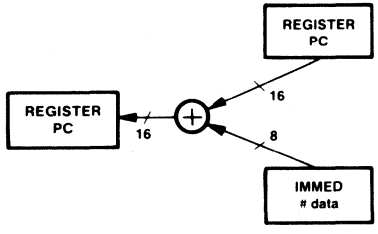


Fig. 4.22 Unconditional short branch and conditional branch operations.

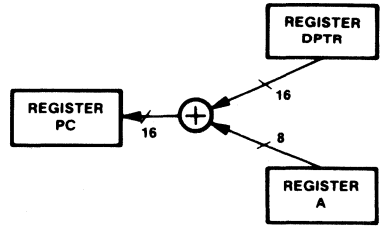


Fig. 4.23 Unconditional branch (indirect) operation.

5.0 INSTRUCTION SET OF THE 8051

The 8051 assembly language needs only forty-two mnemonics to specify the 8051's functions. A function may have several mnemonics (e.g. MOV, MOVX, MOVC) since the function mnemonic specifies when the program memory or external data memory is used in conjunction with the internal data memory. When the function mnemonics are combined with unique address combinations specified in the 'destination, source' field, 111 instructions are possible. The 'destination, source' field specifies the data type and the combination of addressing methods to be used to address the destination and source operands. A summary of the 8051 instruction set is given in Table 5.1.

The syntax of most 8051 assembly language instructions consists of a function mnemonic followed by a 'destination, source' operand field. Thus 'MOV @R0, Data' may be interpreted as 'The content of the internal data memory location addressed by the content of Register 0 receives the content of the internal data memory addressed by Data.'. In two operand instructions, the destination address also serves as the address of the first source. As an example of this, 'ANL Data,#5' may be interpreted as 'The content of the internal data memory location addressed by Data receives the result of the operation when the content of the memory location specified by Data is ANDed with the immediate decimal value 5.'.

The 8051's instruction set is an enhancement of the instruction set familiar to MAB8048/PCB80C48 users. It is enhanced to allow on-chip CPU peripherals and to optimize byte efficiency and execution speed. Efficient use of program memory comes about through an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that adds on to a long or a short address. For example, register addressing allows a two-byte equivalent of the three-byte direct addressing instructions. Also, short branches are more code efficient than long branches. 64 of the instructions execute in 12 oscillator periods, 45 of the instructions execute in 24 oscillator periods, and multiply and divide instructions take only 48 oscillator periods. The number of bytes in each instruction and the number of oscillator periods required for the execution are listed in Table 5.1.

5.1 Organization of the instruction set

Instructions are described here in four functional groups:

- Data transfer
- Arithmetic
- Logic
- Control transfer

The data transfer, arithmetic and logic groups given above are further subdivided into an array of codes that specify whether the operation is to act upon immediate, RB register, accumulator, SFR or memory locations; whether bits, nibbles, bytes or double-bytes are to be processed and what addressing modes are to be employed.

5.1.1 Data transfer

Data transfer operations are divided into three classes:

General purpose
Accumulator-specific
Address-object

These operations do not affect the flag settings except a POP or MOV into the PSW.

There are three general-purpose data transfer operations which may be applied to most operands, with the exception of some specific cases.

MOV	performs a bit or byte transfer from the source operand to the destination operand.
PUSH	increments the SP register and then transfers a byte from the source operand to the stack element currently addressed by SP.
POP	transfers a byte operand from the stack element addressed by the SP register to the destination operand and then decrements SP.

Four accumulator-specific transfer operations are provided:

XCH	exchanges the byte source operand with register A (accumulator).
XCHD	exchanges the low-order nibble of the byte source operand with the low-order nibble of register A.
MOVX	performs a byte move between the external data memory and register A. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
MOVC	performs a move of a byte from the program memory to register A as follows. The operand in register A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to A. MOVC is used for table-look-up byte translation and for accessing operands from code-in-line tables.

There is one address-object operation provided:

MOV DPTR,#data	loads 16-bits of immediate data into a pair of destination registers, DPH and DPL (DPL from low-order address, DPH from high-order address).

5.1.2 Logic

The 8051 performs the basic logic operations on both bit and byte operands. These logic operations are divided into two groups, single-operand operations and two-operand operations.

There are seven single-operand operations provided:

CLR	is used to set either the A register, the C register, or any direct addressed bit to zero (0).
SETB	sets either the C register or any direct addressed bit to one (1).
CPL	either forms the one's complement of the operand in register A and returns the result to the A register without affecting flags or forms the one's complement of the C register or any direct addressed bit.
RL, RLC, RR, RRC, SWAP	are five rotate operations which can be performed on the A register; RL (rotate left), RR (rotate right), RLC (rotate left through C), RRC (rotate right through C), and SWAP (rotate left four). For RLC and RRC, the C flag becomes equal to the last bit rotated out. Swap rotates the A register left four places to exchange bits 3 to 0 with 7 to 4.

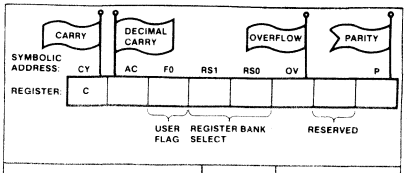
There are three two-operand operations provided:

ANL	performs the bit-wise logical AND of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
ORL	performs the bit-wise logical inclusive-OR of two source operands (for both bit and byte operands) and returns he result to the location of the first operand.
XRL	performs the bit-wise logical exclusive-OR of two source operands (byte operands) and returns the result to the location of the first operand.

5.1.3 Arithmetic

The 8051 provides the four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag permits the addition and subtraction operation to serve for both unsigned and signed binary digits. A correction operation is also provided to allow arithmetic to be performed directly on binary coded decimal (BCD) representations.

There are three one-bit flag registers which are set or cleared by arithmetic operations to reflect certain properties of the result of the operation. These flags are not affected by the increment and decrement of instructions. A fourth flag (P) denotes the parity of the eight accumulator bits. These flag registers are located in the program status word (PSW) register. Their bit assignment is shown below in Fig. 5.1. A list of the instuctions that affect these flags is given in Table 5.1.



Function	Flag	Bit Location
Carry Flag (also the C register)	CY:	PSW.7:
Auxiliary Carry Flag	AC:	PSW.6:
Overflow Flag	OV:	PSW.2:
Parity Flag	P:	PSW.0:
User Flag 0	F0:	PSW.5:
reserved	---:	PSW.1:
Register Select MSb	RS1:	PSW.4:
Register Select LSb	RS0:	PSW.3:

Fig. 5.1 Program status word bit assignment.

CY — is set if the operation results in a carry out of (during addition) or a borrow into (during subtraction) the high-order bit of the result; otherwise CY is cleared.

AC — is set if the operation results in a carry out of the low-order four bits of the result (during addition) or a borrow from the high-order bits into the low-order four bits (during subtraction); otherwise AC is cleared.

OV — is set if the operation results in a carry into the high-order bit of the result but not a carry out of the high-order bit, or vice versa; otherwise OV is cleared. OV is of use in two's complement arithmetic, since it becomes set when the signed result cannot be represented in 8-bits (see Fig. 5.2).

P — is set if the modulo-2 sum of the 8 bits in the accumulator is 1; otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

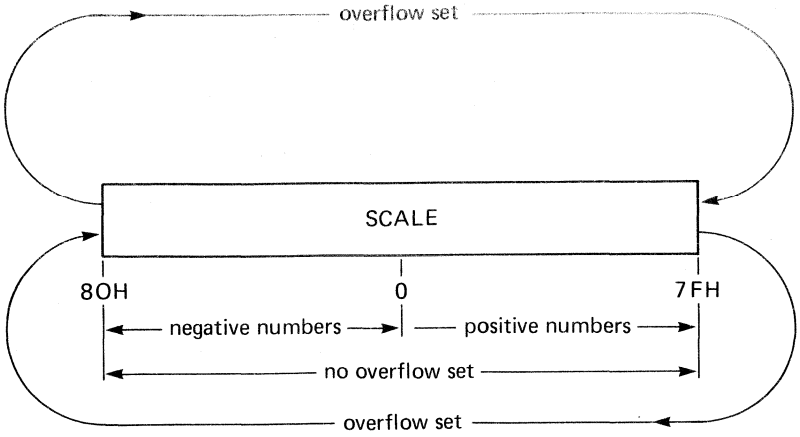


Fig. 5.2 The overflow bit.

There are four addition operations provided:

INC — increment performs an addition of the source operand and one (1) and returns the result to the operand.

ADD — performs an addition between the A register and the second source operand and returns the result to the A register.

ADDC — add with carry performs an addition between the A register and the second source operand; adds one (1) if the C flag is found to have been previously set and returns the result to the A register.

DA — decimal-add-adjust for BCD addition performs a correction to the sum which resulted from the binary addition of two two-digit decimal operands. The binary coded decimal sum formed by DA is returned to A. The carry flag is set if the BCD result is greater than 99; otherwise it is cleared.

There are two subtraction operations provided:

SUBB — subtract with borrow performs a subtraction of the second source operand from the first operand (the accumulator), subtracts one (1) if the C flag is found to have been previously set and returns the result to the A register.

DEC — decrement performs a subtraction of one (1) from the source operand and returns the result to the operand.

The multiplication operand provided:

MUL	performs an unsigned multiplication of the A register and the B register, returning a double-byte result. Register A receives the low-order byte and register B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. C is cleared and AC remains unaltered.

The division operation provided:

DIV	performs an unsigned division of the A register by the B register and returns the integer quotient to register A and the fractional remainder to register B. Division by zero leaves indeterminate data in registers A and B and sets OV, otherwise OV is cleared. C is cleared and AC remains unchanged.

5.1.4 Control transfer

There are three classes of control transfer operations; unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations, some upon a specific condition, cause the program execution to continue at a non-sequential location in program memory.

Unconditional calls, returns and jumps transfer control from the current value of the program counter to a target address. Both direct and indirect transfers are supported. The three transfer operations are desribed below.

ACALL and LCALL	push the address of the next instruction onto the stack (PCL to low-order address, PCH to high-order address) and then transfer control to a target address. Absolute Call is a 2-byte instruction used when the target address is in the current 2K page. Long Call is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K memory page then the Call will be made to the next page since the PC will be incremented to the next instruction prior to execution.
RET	transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.
AJMP, LJMP and SJMP	transfer control to a target operand. The operation of AJMP and LJMP is analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256 byte range centred around the starting address of the next instruction (-128 to +127). The PC-relative short jump facilitates relocatable code.

JMP @A+DPTR — performs a jump relative to the DPTR register. The operand in the A register is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the program memory space. This indirect jump is also useful for implementing N-way branches.

In the control transfer group, the conditional jumps perform a jump dependent upon a specific condition. The destination will be within a 256-byte range centred around the starting address of the next instruction (-128 to +127).

JZ — performs a jump if the accumulator is zero.

JNZ — performs a jump if the accumulator is not zero.

JC — performs a jump if the carry flag is set.

JNC — performs a jump if the carry flag is not set.

JB — performs a jump if the direct address bit is set.

JNB — performs a jump if the direct address bit is not set.

JBC — performs a jump if the direct address bit is set and then clears the direct address bit.

CJNE — compares the first operand to the second and performs a jump if they are not equal. C is set if the first operand is less than the second operand, otherwise it is cleared. Comparisons can be made between the A register and direct addressable bytes in the internal data memory or between an immediate value and either the A register, an RB register in the selected register bank, or a register-indirect addressed byte in the internal data RAM.

DJNZ — DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The DJNZ instruction makes a RAM location efficient for use as a program loop counter by allowing the programmer to decrement and test the counter in a single instruction. The source operand of the DJNZ instruction may be any byte in the internal data memory. Either direct or register addressing may be used to address the source operand.

Program execution control may be transferred by means of internal and external interrupts. All interrupts perform a transfer by pushing the program counter onto the stack and then branching to programs located at absolute locations 3, 11, 19, 27 and 35 in the program memory. The programmer must push all registers that will be altered by his interrupt service routine onto the stack to avoid corruption. Only one interrupt transfer operation is necessary:

RETI	transfers control in a manner identical to RET. In addition, RETI re-enables interrupts for the current priority level.

Further details on the operation and control of the interrupt system is given in 3.7.

Table 5.1 The instruction set summary.

Interrupt Response Time: to finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7μs @ 12 MHz).

Instructions that affect flag settings *

Instruction	Flag C	OV	AC	Instruction	Flag C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

* Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e. the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

Rn - Register R7-R0 of the currently selected Register Bank

direct - 8-bit internal data location's address. This could be an internal data RAM location (0-127) or a SFR [i.e. I/O port, control register, status register, etc. (128-255)].

@Ri - 8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0

#data - 8-bit constant included in instruction

#data 16 - 16-bit constant included in instruction

addr16 - 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space

addr11 - 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction

rel - Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.

bit - Direct addressed bit in internal data RAM or special function register

* - New operation not provided by 8048/8049/8050

Table 5.1 The instruction set summary (continued).

Arithmetic operations

	Mnemonic	Description	Byte	Oscillator period
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Accumulator with Carry	2	12
SUBB	A,Rn	Subtract register from Accumulator with borrow	1	12
SUBB	A,direct	Subtract direct byte from Accumulator with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from Accumulator with borrow	1	12
SUBB	A,#data	Subtract immediate data from Accumulator with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment data pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

Table 5.1 The instruction set summary (continued).

Logical operations

	Mnemonic	Description	Byte	Oscillator period
ANL	A,Rn	AND register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

Boolean variable manipulation

	Mnemonic	Description	Byte	Oscillator period
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to Carry	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct bit is set	3	24
JNB	bit,rel	Jump if direct bit is not set	3	24
JBC	bit,rel	Jump if direct bit is set and clear bit	3	24

Table 5.1 The instruction set summary (continued).

Data transfer

	Mnemonic	Description	Byte	Oscillator period
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct byte	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load data pointer with 16-bit constant	3	24
MOVC	A,@A + DPTR	Move code byte relative to DPTR to Accumulator	1	24
MOVC	A,@A + PC	Move code byte relative to PC and Accumulator	1	24
MOVX	A,@Ri	Move external RAM (8-bit address) to Accumulator	1	24
MOVX	A,@DPTR	Move external RAM (16-bit address) to Accumulator	1	24
MOVX	@Ri,A	Move Accumulator to external RAM (8-bit address)	1	24
MOVX	@DPTR,A	Move Accumulator to external RAM (16-bit address)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order digit indirect RAM with Accumulator	1	12

Table 5.1 The instruction set summary (continued).

Programing branching

	Mnemonic	Description	Byte	Oscillator period
ACALL	addr11	Absolute subroutine call	2	24
LCALL	addr16	Long subroutine call	3	24
RET		Return for subroutine	1	24
RETI		Return for interrupt	1	24
AJMP	addr11	Absolute jump	2	24
LJMP	addr16	Long jump	3	24
SJMP	rel	Short jump (relative address)	2	24
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is zero	2	24
JNZ	rel	Jump if Accumulator is not zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and jump if not equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and jump if not equal	3	24
CJNE	RN,#data,rel	Compare immediate to register and jump if not equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and jump if not equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if not zero	3	24
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	24
NOP		No operation	1	12

Notes on Table 5.1

Data addressing modes

Rr	Working register R0 - R7.
direct	128 internal RAM locations and any special function register (SFR).
@Ri	Indirect internal RAM location addressed by register R0 or R1.
#data	8-bit constant included in the instruction.
#data16	16-bit constant included in instruction.
bit	Direct addressed bit in internal RAM or SFR.
addr16	16-bit destination address. Used by LCALL and LJMP. The branch will be anywhere within the 64K-byte program memory address space.
addr11	11-bit destination address. Used by ACALL and AJMP" The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
rel	Signed (two's complement) (-bit offset byte. Used by SJMP anf all conditional jumps. Range is -128 to +127 bytes relative to the first byte of the following instruction.

Hexadecimal opcode cross-reference to Table 5.2.

*: 8, 9, A, B, C, D, E, F.

o: 1, 3, 5, 7, 9, B, D, F.

: 0, 2, 4, 6, 8, A, C, E.

5.2 Instruction definitions

The rest of this chapter defines all the instructions and operations the 8051 can perform. There is a separate section for each of the 51 basic operations, ordered alphabetically according to the operation mnemonic.

When an operation may apply to more than one data type (generally bit and byte data), the 8051 assembly language uses the same mnemonic for each, reducing the number of mnemonics the programmer requires to remember. The assembler determines which instruction is appropriate from the operands specified. Thus, the mnemonic 'CLR' can operate on the eight-bit accumulator ('CLR A'), or on one-bit variables ('CLR F0'). The mnemonics ANL, ORL, CPL, and MOV can relate to more than one data type as well. These operations present each data type in a separate section.

Each section then describes the action taken by the operation, the flags, and registers affected, and shows a short example of how an instruction might be used in a program. Next comes the number of bytes and machine cycles required, the corresponding binary machine-language encoding, and a symbolic description or restatement of the function implemented.

Note: Only the carry, auxilliary-carry, and overflow flags are discussed in these instruction descriptions. Since the parity bit (PSW.0) is recomputed after every instruction cycle any instruction that alters the accumulator - either inherently or as a special function register - could affect the parity flag. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by the generalized bit-manipulation instructions.

Nineteen operations allow more than one addressing mode for the source and/or destination operand. The headings for these sections show the instruction format with such operands enclosed in brackets (for example, MOV <dest-byte> <src-byte>). The operation description tells what modes (or combination of modes) are allowed, and gives the assembly language notation, byte and cycle counts, encoding format, and a symbolic description for each.

ACALL adr11

Function: Absolute call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2 K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

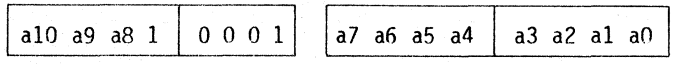
Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

```
ACALL          SUBRTN
```

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2
Cycles: 2

Opcode 11H



Operation: ACALL
$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15-8})$
$(PC_{10-0}) \leftarrow$ page address

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set if there is a carry-out from bit 7 or bit 3 respectively, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

```
ADD       A,R0
```

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1
Cycles: 1

Opcode 28-2FH

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2
Cycles: 1

Opcode 25H

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A) ← (A) + (direct)

ADD A,@Ri

Bytes: 1
Cycles: 1

Opcode 26, 27H

0 0 1 0	0 1 1 i

Operation: ADD
(A) ← (A) + ((R_i))

ADD A,#data

Bytes: 2
Cycles: 1

Opcode 24H

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A) ← (A) + #data

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set if there is a carry-out from bit 7 or bit 3 respectively, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

```
ADD       A,R0
```

will leave 6EH (01101110B) in the accumulator with AC flag cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1
Cycles: 1

Opcode 38-3FH

0 0 1 1	1 r r r

Operation: ADDC
(A) ← (A) + (C) + (Rn)

ADDC A,direct

Bytes: 2
Cycles: 1

Opcode 35H

0 0 1 1	0 1 0 1	direct address

Operation: ADDC
(A) ← (A) + (C) + (direct)

ADDC A,@Ri

Bytes: 1
Cycles: 1

Opcode 36, 37H

0 0 1 1	0 1 1 i

Operation: ADDC
(A) ← (A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2
Cycles: 1

Opcode 34H

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC
(A) ← (A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (AFTER incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2 K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H.
The instruction;
AJMP JMPADR
is at location 0345H and will load the PC with 0123H.

Bytes: 2
Cycles: 2

Opcode 01, 21, 41, 61, 81, A1, C1, E1

a10 a9 a8 0	0 0 0 1

a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP
(PC) ← (PC) + 2
(PC_{10-0}) ← page address

ANL <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six address mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulaor or immediate data.

NOTE: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, NOT the input pins.

Example: If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
ANL     A,R0
```

will leave 82H (10000010B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time. The instruction,

```
ANL     P1,#01110011B
```

will clear bits 7, 3 and 2 of output port 1.

ANL A,Rn

Bytes: 1
Cycles: 1

Opcode 58-5FH

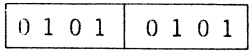
0 1 0 1	1 r r r

Operation: ANL
(A) ← (A) ^ (Rn)

ANL A,direct

Bytes: 2
Cycles: 1

Opcode 55H

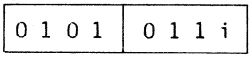


Operation: ANL
(A) ← (A) ^ (direct)

ANL A,@Ri

Bytes: 1
Cycles: 1

Opcode 56-57H

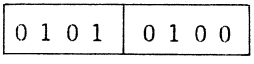


Operation: ANL
(A) ← (A) ^ ((Ri))

ANL A,#data

Bytes: 2
Cycles: 1

Opcode 54H

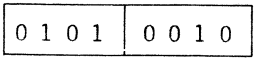


Operation: ANL
(A) ← (A) ^ #data

ANL direct,A

Bytes: 2
Cycles: 1

Opcode 52H



Operation: ANL
(direct) ← (direct) ^ (A)

ANL direct,#data

Bytes: 3
Cycles: 2

Opcode 53H

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct) ← (direct) ^ #data

ANL C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected. Only direct bit addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 =1, ACC.7 =1, and OV =0:

```
MOV  C,P1.0          ;LOAD CARRY WITH INPUT PIN STATE
ANL  C.ACC.7         ;AND CARRY WITH ACCUM. BIT 7
ANL  C./OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2
Cycles: 2

Opcode 82H

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← (C) ^ (bit)

ANL C,/bit

Bytes: 2
Cycles: 2

Opcode B0H

1 0 1 1	0 0 0 0	bit address

Operation: ANL
(C) ← (C) | (bit)

CJNE <dest-byte>,<src-byte>,rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of [dest-byte] is less than the unsigned integer value of [src-byte]; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
          CJNE  R7,#60H,NOT_EQ
;         ...   .....          ; R7 = 60H
NOT_EQ:   JC    REQ_LOW        ; IF R7<60H
;         ...   .....          ; R7>60H
```

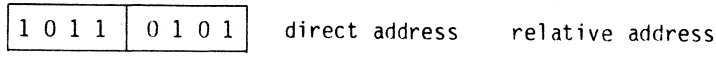
sets the carry flag and branches to the instruction at label NOT EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to P1 is also 34H, then the instruction, WAIT: CJNE A,P1,WAIT clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H).

CJNE A,direct,rel

Bytes: 3
Cycles: 2

Opcode B5H



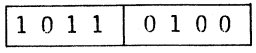
Operation:

```
CJNE
(PC) ← (PC) + 3
IF (direct) < (A)
     THEN (PC) ← (PC) + relative offset and (C) ← 0
IF (direct) > (A)
     THEN (PC) ← (PC) + relative offset and (C) ← 1
IF (direct) = (A)
     THEN (C) ← 0
```

CJNE A,#data,rel

Bytes: 3
Cycles: 2

Opcode B4H

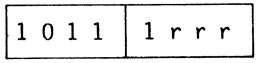


Operation: CJNE
(PC) ← (PC) + 3
IF #data < (A)
THEN (PC) ← (PC) + relative offset and (C) ← 0
IF #data > (A)
THEN (PC) ← (PC) + relative offset and (C) ← 1
IF #data = (A)
THEN (C) ← 0

CJNE Rn,#data,rel

Bytes: 3
Cycles: 2

Opcode B8-BFH

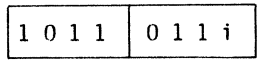


Operation: CJNE
(PC) ← (PC) + 3
IF #data < (Rn)
THEN (PC) ← (PC) + relative offset and (C) ← 0
IF #data > (Rn)
THEN (PC) ← (PC) + relative offset and (C) ← 1
IF #data = (Rn)
THEN (C) ← 0

CJNE @Ri,#data,rel

Bytes: 3
Cycles: 2

Opcode B6, B7H



Operation: CJNE
(PC) ← (PC) + 3
IF #data < ((Ri))
THEN (PC) ← (PC) + relative offset and (C) ← 0
IF #data > ((Ri))
THEN (PC) ← (PC) + relative offset and (C) ← 1
IF #data = ((Ri))
THEN (C) ← 0

CLR A

Function: Clear Accumulator

Description: The accumulator is cleared (all bits set to zero). No flags are affected.

Example: The accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the accumulator set to 00H (00000000B).

Bytes: 1
Cycles: 1

Opcode E4H

1 1 1 0	0 1 0 0

Operation: CLR
(A) ← 0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1
Cycles: 1

Opcode C3H

1 1 0 0	0 0 1 1

Operation: CLR
(C) ← 0

CLR bit

Bytes: 2
Cycles: 1

Opcode C2H

1 1 0 0	0 0 1 0	bit address

Operation: CLR
(bit) ← 0

CPL A

Function: Complement Accumulator

Description: Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a 1 are changed to 0 and vice-versa. No flags are affected.

Example: The accumulator contains 5CH (01011100B). The instruction,

CPL A

will leave the accumulator set to 0A3H (10100011B).

Bytes: 1
Cycles: 1

Opcode F4H

1 1 1 1	0 1 0 0

Operation: CPL
(A) ← ¬ (A)

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a 1 is changed to 0 and vice-versa. No other flags are affected. CLR can operate on the carry or on any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, NOT the input pin.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction sequence,

```
CPL  P1.1
CPL  P1.2
```

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1
Cycles: 1

Opcode B3H

1 0 1 1	0 0 1 1

Operation: CPL
(C) ← ¬ (C)

CPL bit

Bytes: 2
Cycles: 1

Opcode B2H

1 0 1 1	0 0 1 0	bit address

Operation: CPL
(bit) ← ¬ (bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the correct BCD digit in the low-order nibble. This internal addition sets the carry flag if a carry-out of the low-order four-bit field propagates through all high-order bits. If a carry-out does not propagate through all high-order bits, the carry flag is not cleared.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the correct BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but would not clear the carry. The carry flag thus indicates if the sum of the original two BCD variable is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H or 66H to the accumulator, depending on initial accumulator and PSW conditions.

Note: DA A cannot convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC   A,R3
DA     A
```

will first perform a standard two's-complement binary addition, resulting in the value 0BEH (10111110) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67 and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow has occurred. The true sum of 56, 67 and 1 is 124. BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD    A,#99H
DA     A
```

will leave the carry set and 29H in the accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted as 30 - 1 = 29.

Bytes: 1
Cycles: 1

Opcode D4H

1	1	0	1	0	1	0	0

Operation:

```
DA
-contents of Accumulator are BCD
IF [[(A3-0)>9] v [(AC) = 1]]
     THEN (A3-0) ← (A3-0) + 6
     AND
IF [[(A7-4)>9] v [(C) = 1]]
     THEN (A7-4) ← (A7-4) + 6
```

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, NOT the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH respectively.

DEC A

Bytes: 1
Cycles: 1

Opcode 14H

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) - 1

DEC Rn

Bytes: 1
Cycles: 1

Opcode 18-1FH

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) - 1

DEC direct

Bytes: 2
Cycles: 1

Opcode 15H

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct) ← (direct) - 1

DEC @Ri

Bytes: 1
Cycles: 1

Opcode 16-17H

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any event.

Example: The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 x 18) + 17. Carry and OV will both be cleared.

Bytes: 1
Cycles: 4

Opcode 84H

1 0 0 0	0 1 0 0

Operation: DIV

$(A)_{15-8}$ ← (A) / (B)

$(B)_{7-0}$

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, NOT the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ   40H,LABEL_1
DJNZ   50H,LABEL_2
DJNZ   60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

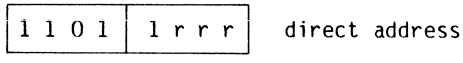
```
          MOV  R2,#8
TOGGLE:   CPL  P1.7
          DJNZ R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output P1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2
Cycles: 2

Opcode D8-DFH



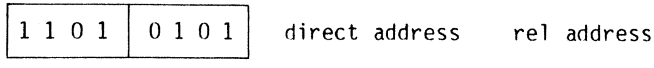
Operation:

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) - 1
IF (Rn) > 0 or (Rn) < 0
     THEN
     (PC) ← (PC) + rel
```

DNJZ direct,rel

Bytes: 3
Cycles: 2

Opcode D5H



Operation:

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) - 1
IF (direct) > 0 or (direct) < 0
     THEN
     (PC) ← (PC) + rel
```

INC byte

Function: Increment

INC increments the indicated variable by 1. An original value of OFFH will overflow the 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC  @R0
INC  R0
INC  @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding 00H and 41H respectively.

INC A

Bytes: 1
Cycles: 1

Opcode 04H

0 0 0 0	0 1 0 0

Operation: INC
(A) ← (A) + 1

INC Rn

Bytes: 1
Cycles: 1

Opcode 08-0FH

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn) + 1

INC direct

Bytes: 2
Cycles: 1

Opcode 05H

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct) ← (direct) + 1

INC @Ri

Bytes: 1
Cycles: 1

Opcode 06, 07H

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo-2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from OFFH to 00H will increment the high-order byte (DPH). No flags are affected. This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and OFEH, respectively. The instruction sequence,

```
INC   DPTR
INC   DPTR
INC   DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1
Cycles: 2

Opcode A3H

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a 1, jump to the address indicated, otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input P1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

```
JB   P1.2,LABEL1
JB   ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3
Cycles: 2

Opcode 20H

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF (bit) = 1
     THEN
     (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is 1, branch to the address indicated; otherwise proceed with the next instruction. In either case clear the desiganated bit. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: when this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, NOT the input pin.

Example: The accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).

Bytes: 3
Cycles: 2

Opcode 10H

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC) + 3
IF (bit) = 1
     THEN
     (bit) ← 0
     (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Opcode 40 H

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC) + 2
IF (C) = 1
     THEN
     (PC)← (PC) + rel
```

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo-2^{16}): a carry-out from the low-order eight bits propagates through the high-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
          MOV    DPTR,#JMP_TBL
          JMP    @A+DPTR
JMP_TBL:  AJMP   LABEL0
          AJMP   LABEL1
          AJMP   LABEL2
          AJMP   LABEL3
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1
Cycles: 2

Opcode 73H

0	1	1	1	0	0	1	1

Operation: JMP
(PC) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a 0, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input Port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB  P1.3,LABEL1
JNB  ACC.3,LABEL2
```

will cause program execution to continue at the instruction label LABEL2.

Bytes: 3
Cycles: 2

Opcode 30H

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF (bit) = 0
     THEN (PC) ← (PC) + rel
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is 0, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The flag is not modified.

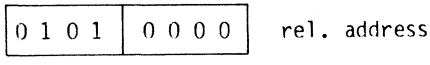
Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Opcode 50H



Operation:
```
JNC
(PC) ← (PC) + 2
IF (C) = 0
     THEN (PC) ← (PC) + rel
```

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the accumulator is a 1, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

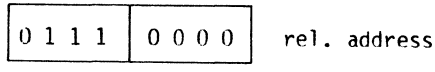
Example: The accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   C
JNZ   LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2
Cycles: 2

Opcode 70H



Operation:

```
JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0
     THEN (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the accumulator are 0, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

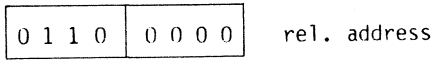
Example: The accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC   C
JZ    LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Opcode 60H



Operation:
```
JZ
(PC) ← (PC) + 2
IF (A) = 0
     THEN (PC) ← (PC) + rel
```

LCALL addr16

Function: Long Call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result on to the stack (low-order byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded with the second and third bytes (respectively) of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64 K-byte program memory address space. No flags are affected.

Example: Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3
Cycles: 2

Opcode 12H

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64 K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3
Cycles: 2

Opcode 02H

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP
(PC) ← $addr_{15-0}$

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30 holds 40H. The value of RAM location 40H is 10H. The data present at input Port 1 is 11001010B (CAH).

```
MOV   R0,#30H       ;R0 ← 30H
MOV   A,@R0         ;A ← 40H
MOV   R1,A          ;R1 ← 40H
MOV   B,@R1         ;B ← 10H
MOV   @R1,P1        ;RAM(40H) ← 0CAH
MOV   P2,P1         ;P2 ← 0CAH
```

leaves the value 30H in register 0, 40H in the accumulator and register R1, 10H in register B, and 0CAH (11001010B) in RAM location 40H and output on Port 2.

MOV A,Rn

Bytes: 1
Cycles: 1

Opcode E8-EFH

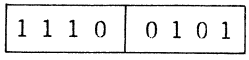
1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

MOV A,direct

Bytes: 2
Cycles: 1

Opcode E5H



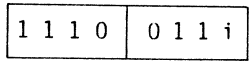
Operation: MOV
(A) ← (direct)

Note: MOV A,Acc is not a valid instruction

MOV A,@Ri

Bytes: 1
Cycles: 1

Opcode E6, E7H

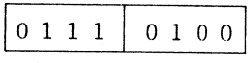


Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2
Cycles: 1

Opcode 74H

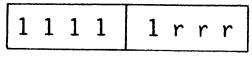


Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1
Cycles: 1

Opcode F8-FFH

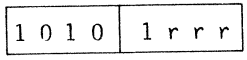


Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2
Cycles: 2

Opcode A8-AFH



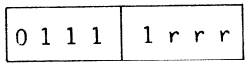
direct address

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2
Cycles: 1

Opcode 78-7FH



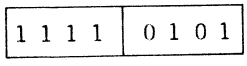
immediate data

Operation: MOV
(Rn) ← #data

MOV direct,A

Bytes: 2
Cycles: 1

Opcode F5H



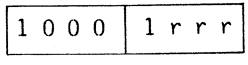
direct address

Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2
Cycles: 2

Opcode 88-8FH



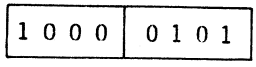
direct address

Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3
Cycles: 2

Opcode 85H



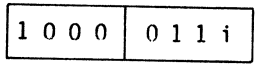
direct address (src) direct address (dest)

Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2
Cycles: 2

Opcode 86, 87H



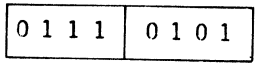
direct address

Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3
Cycles: 2

Opcode 75H



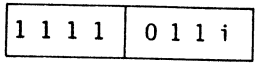
direct address immediate data

Operation: MOV
(direct) ← #data

MOV @Ri,A

Bytes: 1
Cycles: 1

Opcode F6, F7H



Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2
Cycles: 2

Opcode A6, A7H

1 0 1 0	0 1 1 i	direct address

Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2
Cycles: 1

Opcode 76, 77H

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((Ri)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must then be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV   P1.3,C
MOV   C,P3.3
MOV   P1.2,C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2
Cycles: 1

Opcode A2H

1 0 1 0	0 0 1 0	bit address

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2
Cycles: 2

Opcode 92H

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit) ← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected. This is the only instruction which moves 16-bits of data.

Example: The instruction,

MOV DPTR,#1234H

will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3
Cycles: 2

Opcode 90H

1 0 0 1	0 0 0 0	immediate data 15-8	immediate data 7-0

Operation: MOV
(DPTR) ← $\#addr_{15-0}$
DPH DPL ← $\#data_{15-8}$ $\#data_{7-0}$

MOVC A,@A+ base-reg

Function: Move Code byte

Description: The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter, the PC is incremented to the address of the following instruction before being added with the accumulator: otherwise the base register is not altered. Sixteen-bit addition is performed so that a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC:  INC     A
         MOVC    A,@A+PC
         RET
         DB      66H
         DB      77H
         DB      88H
         DB      99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

MOVC A,@A+DPTR

Bytes: 1
Cycles: 2

Opcode 93H

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A+PC

Bytes: 1
Cycles: 2

Opcode 83H

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

MOVX <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, depending upon whether they provide an eight or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 special function register retains its previous contents while P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64 K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines is connected to the MAB8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

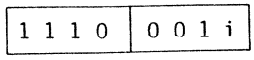
```
MOVX  A,@R1
MOVX  @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1
Cycles: 2

Opcode E2, E3H

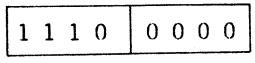


Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1
Cycles: 2

Opcode EOH

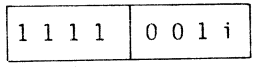


Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1
Cycles: 2

Opcode F2, F3H

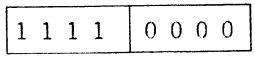


Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1
Cycles: 2

Opcode FOH



Operation: MOVX
(DPTR) ← (A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1
Cycles: 4

Opcode A4H

1 0 1 0	0 1 0 0

Operation: MUL

$(A)_{7-0} \leftarrow (A) \times (B)$
$(B)_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: A low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles is required. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR   P2,7
NOP
NOP
NOP
NOP
SETB  P2,7
```

Bytes: 1
Cycles: 1

Opcode 00H

0 0 0 0	0 0 0 0

Operation: NOP
(PC) ← (PC) + 1

ORL <dest-byte>,<src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bit-wise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, NOT the input pins.

Example: If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL A,R0

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

ORL P1,#00110010B

will set bits 5, 4 and 1 of output Port 1.

ORL A,Rn

Bytes: 1
Cycles: 1

Opcode 48-4FH

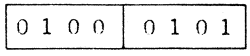
0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) v (Rn)

ORL A,direct

Bytes: 2
Cycles: 1

Opcode 45H

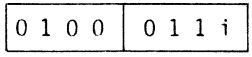


Operation: ORL
(A) ← (A) v (direct)

ORL A,@Ri

Bytes: 1
Cycles: 1

Opcode 46, 47H

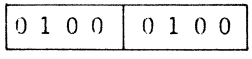


Operation: ORL
(A) ← (A) v ((Ri))

ORL A,#data

Bytes: 2
Cycles: 1

Opcode 44H

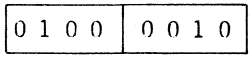


Operation: ORL
(A) ← (A) v #data

ORL direct,A

Bytes: 2
Cycles: 1

Opcode 42H



Operation: ORL
(direct) ← (direct) v (A)

ORL direct,#data

Bytes: 3
Cycles: 2

Opcode 43H

0 1 0 0	0 0 1 1	direct address	immediate data

Operation: ORL
(direct) ← (direct) v #data

ORL C,<src-bit>

Function: Logical-OR for bit variables
Description: Set the carry flag if the Boolean value is a logical 1; otherwise leave the carry in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

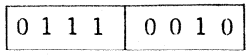
Example: Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```
MOV   C,P1.0        ;LOAD CARRY WITH INPUT PIN P1.0
ORL   C,ACC.7       ;OR CARRY WITH THE ACC. BIT 7
ORL   C,/OV         ;OR CARRY WITH INVERSE OF OV
```

ORL C,bit

Bytes: 2
Cycles: 2

Opcode 72H



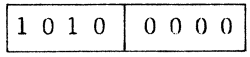
bit address

Operation: ORL
(C) ← (C) v (bit)

ORL C,/bit

Bytes: 2
Cycles: 2

Opcode A0H



bit address

Operation: ORL
(C) ← (C) v (bit)

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the stack pointer are read, and the stack pointer is decremented by one. The value read is transferred to the directly addressed byte indicated. No flags are affected.

Example: The stack pointer originally contains the value 32H, and internal RAM locations 30H to 32H contain the values 20H, 23H and 01H respectively. The instruction sequence,

```
POP   DPH
POP   DPL
```

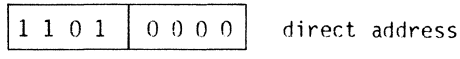
will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

```
POP SP
```

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2
Cycles: 2

Opcode D0H



Operation: POP
(direct) ← ((SP))
(SP) ← (SP) - 1

PUSH direct

Function: Push on to stack

Description: The stack pointer is incremented by one. The contents of the indicated variable are then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

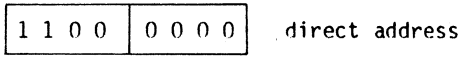
Example: On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

```
PUSH   DPL
PUSH   DPH
```

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 08H, respectively.

Bytes: 2
Cycles: 2

Opcode COH



Operation: PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1
Cycles: 2

Opcode 22H

0 0 1 0	0 0 1 0

Operation: RET

$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack and restores the interrupt logic to accept additional interrupts at the same priority level as the one previously processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM location 0AH and 0BH contain the values 23H amd 01H, respectively. The instruction,

RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H.

Bytes: 1
Cycles: 2

Opcode 32H

0 0 1 1	0 0 1 0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate accumulator Left

Description: The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

```
RL   A
```

leaves the accumulator holding the value (10001011B) with the carry unaffected.

Bytes: 1
Cycles: 1

Opcode 23H

0 0 1 0	0 0 1 1

Operation: RL
$(A_{n+1}) \leftarrow (An)$ n = 0 - 6
$(A0) \leftarrow (A7)$

RLC A

Function: Rotate accumulator Left through the Carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B) and the carry is zero. The instruction,

```
RLC   A
```

leaves the accumulator holding the value 8AH (10001010B) with the carry set.

Bytes: 1
Cycles: 1

Opcode 33H

0 0 1 1	0 0 1 1

Operation: RLC
$(A_{n+1}) \leftarrow (An)$ n = 0 - 6
$(A0) \leftarrow (C)$
$(C) \leftarrow (A7)$

RR A

Function: Rotate accumulator Right

Description: The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

```
RR   A
```

leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1
Cycles: 1

Opcode 03H

0 0 0 0	0 0 1 1

Operation:
RR
$(An) \leftarrow (A_{n+1})$ n = 0 - 6
$(A7) \leftarrow (A0)$

RRC A

Function: Rotate accumulator Right through Carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B) and the carry is zero. The instruction,

```
RRC   A
```

leaves the accumulator holding the value 62H (10001010B) with the carry set.

Bytes: 1
Cycles: 1

Opcode 13H

0 0 0 1	0 0 1 1

Operation:
RRC
$(An) \leftarrow (A_{n+1})$ n = 0 - 6
$(A7) \leftarrow (C)$
$(C) \leftarrow (A0)$

SETB bit

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34 H (00110100B). The instruction,

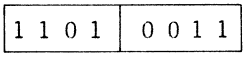
```
SETB  C
SETB  P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1
Cycles: 1

Opcode D3H

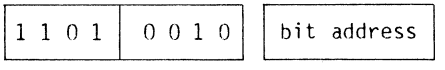


Operation: SETB
(C) ← 1

SETB bit

Bytes: 2
Cycles: 1

Opcode D2H



Operation: SETB
(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

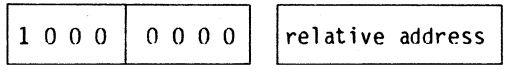
```
SJMP  RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Thus, an SJMP with a displacement of OFEH would be a one-instruction infinite loop.

Bytes: 2
Cycles: 2

Opcode 80H



Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed in to bit 6, but not in to bit 7, or in to bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Note that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1
Cycles: 1

Opcode 98H-9FH

1 0 0 1	1 r r r

Operation: SUBB
(A) ← (A) - (C) - (Rn)

SUBB A,direct

Bytes: 2
Cycles: 1

Opcode 95H

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) - (C) - (direct)

SUBB A,@Ri

Bytes: 1
Cycles: 1

Opcode 96, 97H

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) - (C) - ((Ri))

SUBB A,#data

Bytes: 2
Cycles: 1

Opcode 94H

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB
(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be considered as a four-bit rotate instruction. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the accumulator holding the value 5CH (01011100B).

Bytes: 1
Cycles: 1

Opcode C4H

1 1 0 0	0 1 0 0

Operation: SWAP $(A_{3-0}) \leftrightarrows (A_{7-4})$

XCH A,byte

Function: Exachange Accumulator with byte variable

Description: XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave the RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1
Cycles: 1

Opcode C8-CFH

1 1 0 0	1 r r r

Operation: XCH $(A) \leftrightarrows (Rn)$

XCH A,direct

Bytes: 2
Cycles: 1

Opcode C5H

1 1 0 0	0 1 0 1	direct address

Operation: XCH
(A) $\leftrightarrow$ (direct)

XCH A,@Ri

Bytes: 1
Cycles: 1

Opcode C6, C7H

1 1 0 0	0 1 1 i

Operation: XCH
(A) $\leftrightarrow$ ((Ri))

XCHD A,@Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1
Cycles: 1

Opcode D6, D7H

1 1 0 1	0 1 1 i

Operation: XCHD
$(A_{3-0}) \leftrightarrow ((Ri_{3-0}))$

XRL <dest-byte>,<src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, NOT the input pins.

Example: If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

will leave the accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

XRL P1,#00110001B

will complement bits 5, 4 and 0 of output Port 1.

XRL A,Rn

Bytes: 1
Cycles: 1

Opcode 68-6FH

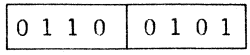
0 1 1 0	1 r r r

Operation: XRL
(A) ← (A) ∀ (Rn)

XRL A,direct

Bytes: 2
Cycles: 1

Opcode 65H

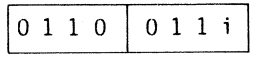


Operation: XRL
(A) ← (A) ∀ (direct)

XRL A,@Ri

Bytes: 1
Cycles: 1

Opcode 66, 67H

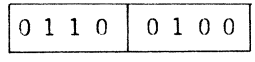


Operation: XRL
(A) ← (A) ∀ ((Ri))

XRL A,#data

Bytes: 2
Cycles: 1

Opcode 64H

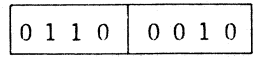


Operation: XRL
(A) ← (A) ∀ #data

XRL direct,A

Bytes: 2
Cycles: 1

Opcode 62H

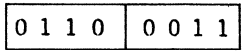


Operation: XRL
(direct) ← (direct) ∀ (A)

XRL direct,#data

Bytes: 3
Cycles: 2

Opcode 63H



Operation: XRL
(direct) ← (direct) ∀ #data

6.0 APPLICATION EXAMPLES FOT THE 8051/80C51 FAMILY OF MICROCOMPUTERS

This chapter is divided into three sections:

- 8051 programming techniques
- Peripheral interfacing techniques
- Connections to peripherals

The first section has 8051 software examples for some common routines in controller applications. Some of the routines included are multiple-precision arithmetic and table look-up techniques.

Peripheral interfacing techniques include routines for handling the 8051's I/O ports, serial channel and timer/counters. In this section the I/O port reconfiguration, software delay timing, and transmitting serial port character strings along with other routines.

6.1 8051 programming techniques

6.1.1 Radix conversion routines

The divide instruction can be used to convert a number from one radix to another. BINBCD is a short subroutine to convert an 8-bit unsigned binary integer in the accumulator (between 0 and 255) to a three digit (two-byte) BCD representation. The hundred's digit is returned in one variable (HUND) and the ten's and unit's digits returned as packed BCD in another (TENONE).

```
;
;BINBCD    CONVERT 8-BIT BINARY VARIABLE IN ACCUMULATOR
;          TO 3-DIGIT PACKED BCD FORMAT.
;          HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
;          TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND      DATA   21H
TENONE    DATA   22H
;
BINBCD:   MOV    B,#100          ;DIVIDED BY 100 TO
          DIV    AB              ;DETERMINE NUMBER OF HUNDREDS
          MOV    HUND,A
          MOV    A,#10           ;DIVIDE REMAINDER BY TEN TO
          XCH    A,B             ;DETERMINE NUMBER OF TENS LEFT
          DIV    AB              ;TEN'S DIGIT IN ACC, REMAINDER IS
                                 ;ONE'S DIGIT
          SWAP   A
          ADD    A,B             ;PACK BCD DIGITS IN ACC
          MOV    TENONE,A
          RET
;
```

The divide instruction can also separate data in the accumulator into sub-fields. For example, dividing packed BCD data by 16 will separate the two nibbles, leaving the high-order digit in the accumulator and the low-order digit (the remainder) in B. Each digit is right-justified, so they can be processed individually. This example receives two packed BCD digits in the accumulator, separates the digits, computes their product, and returns the product in packed BCD format to the accumulator.

```
;MULBCD   UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR,
;         FIND THEIR PRODUCT, AND RETURN PRODUCT
;         IN PACKED BCD FORMAT IN ACCUMULATOR
;
MULBCD:  MOV    B,#10H          ;DIVIDE INPUT BY 16
         DIV    AB              ;A & B HOLD SEPARATED DIGITS
                                ;(EACH RIGHT JUSTIFIED IN REGISTER).
         MUL    AB              ;A HOLDS PRODUCT IN BINARY FORMAT (0-
                                ;99 (DECIMAL) = 0 — 63H)
         MOV    B,#10           ;DIVIDE PRODUCT BY 10
         DIV    AB              ;A HOLDS NUMBER OF TENS, B HOLDS
                                ;REMAINDER
         SWAP   A
         ORL    A,B             ;PACK DIGITS
         RET
```

6.1.2 Multiple precision aritmetic

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple-precision calculations by repeating the operation with successively higher-order operand bytes. If the input data for a multiple-precision operation is an unsigned string of integers, the carry flag will be set upon completion if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data, the most significant bit of the original input data's most significant byte indicates the sign of the string, so the overflow flag (OV) will indicate if overflow or underflow occurred.

```
;
;SUBSTR   SUBTRACT STRING INDICATED BY R1
;         FROM STRING INDICATED BY R0 TO
;         PRECISION INDICATED BY R2.
;         CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR:  CLR    C               ;BORROW = 0.
SUBS1:   MOV    A,@R0           ;LOAD MINUEND BYTE
         SUBB   A,@R1           ;SUBTRACT SUBTRAHEND BYTE
         MOV    @R0,A           ;STORE DIFFERENCE BYTE
         INC    R0              ;BUMP POINTERS TO NEXT PLACE
         INC    R1
         DJNZ   R2,SUBS1        ;LOOP UNTIL DONE
;
;        WHEN DONE, TEST IF OVERFLOW OCCURRED
;        ON LAST ITERATION OF LOOP.
;
         JNB    OV,OV_OK
;        ...    .....           (OVERFLOW RECOVERY ROUTINE)
         OV-OK: RET             ;RETURN
```

6.1.3 Table look-up sequences

The two versions of the MOVC instructions are used as part of a three step sequence to access look-up tables in ROM. To use the DPTR version, load the data pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A,@A+DPTR. The data pointer may be loaded with a constant for short tables, or to allow more complicated data structures, and tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set

The PC-based version is used with the smaller, 'local' tables, and has the advantage of not affecting the data pointer. This makes it useful in interrupt routines or other situations where the DPTR contents might be significant. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction's address to the start of the table by adding that offset to the accumulator; then execute the MOVC A,@A+PC instruction

As a non-trivial situation where this instruction would be used, consider applications which store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in the linear (one-dimensional) program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

Entry address = BASE + (NDIMEN x INDEXI) + INDEXJ

The subroutine MATRX1 can access an entry in an array with less than 255 elements (e.g., an 11x21 array with 231 elements). The table entries are defined using the data byte (DB) directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

To handle the more general case, subroutine MATRX2 allows tables to be unlimited in size, by combining the MUL instruction, double-precision addition, and the data pointer-based version of MOVC. The only restriction is that each index lies between 0 and 255.

```
;
;MATRX1  LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;        TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;        USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A+PC'.
;        THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;        BE SMALL, I.E. LESS THAN ABOUT 255 ENTRIES.
;        TABLE USED IN THIS EXAMPLE IS 11 x 21.
;        DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;
;        [(BASE ADDRESS) + (21 X INDEXI) + (INDEXJ)]
;
INDEXI  EQU   R6              ;FIRST COORDINATE OF ENTRY (0-10).
INDEXJ  DATA  23H             ;SECOND COORDINATE OF ENTRY (0-20).
;
MATRX1: MOV   A,INDEXI
        MOV   B, #21
        MUL   AB              ;(21 X INDEXI)
        ADD   A,INDEXJ        ;ADD IN OFFSET WITHIN ROW
;
;       ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;       ENTRY (0,0).
;
        INC   A
        MOVC  A,@A+PC
        RET
BASE1:  DB    1               ;(entry 0,0)
        DB    2               ;(entry 0,1)
;       ..    ......
        DB    21              ;(entry 0,20)
        DB    22              ;(entry 1,0)
;       ..    ......
        DB    42              ;(entry 1,20)
;       ..    ......
;       ..    ......
;       DB    231             ;(entry 10,20)
MATRX2: MOV   A,INDEXI        ;LOAD FIRST COORDINATE
        MOV   B,#NDIMEN
        MUL   AB              :INDEXI X NDIMEN
        ADD   A,#LOW(BASE2)   ;ADD IN 16-BIT BASE ADDRESS
        MOV   DPL,A
        MOV   A,B
        ADDC  A,#HIGH(BASE2)
        MOV   DPH,A           ;DPTR=(BASE ADDR) + (INDEX X NDIMEN)
        MOV   A,INDEXJ
        MOVC  A,@A+DPTR       ;ADD INDEXJ AND FETCH BYTE
        RET
        ...   ......
BASE2:  DB    0               ;(entry 0,0)
        DB    0               ;(entry 0,1)
;       ...   ......
        DB    0               ;(entry 0, NDIMEN-1)
        DB    0               ;(entry 1,0)
;       ...   ......
        DB    0               ;(entry 1, NDIMEN-1)
;       ...   ......
;       ...   ......
        DB    0               ;(entry MDIMEN-1, NDIMEN-1)
```

6.1.4 Saving CPU status during interrupts

When the 8051 hardware recognizes an interrupt request, the program control branches automatically to the corresponding service routine by forcing the CPU to process a Long Call (LCALL) instruction to the appropriate address. The return address is stored on the top of the stack. After completing the service routine, an RETI instruction returns the processor to the background program at the point where the interrupt occurred.

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program might not resume correctly (Such a change could look like a spontaneous random error. An example of this will be given later in this section, in the second method of I/O port reconfiguration.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save such registers on the stack.

```
;
LOC_TMP EQU     $                   ;REMEMBER LOCATION COUNTER
;
        ORG     0003H               ;STARTING ADDRESS FOR INTERRUPT ROUTINE
        LJMP    SERVER              ;JUMP TO ACTUAL SERVICE ROUTINE LOCATE
                                    ;ELSEWHERE
;       ...     ......
        ORG     LOC_TMP             ;RESTORE LOCATION COUNTER
SERVER: PUSH    PSW
        PUSH    ACC                 ;SAVE ACCUMULATOR (NOTE DIRECT ADDRESS
                                    ;NOTATION)
        PUSH    B                   ;SAVE B REGISTER
        PUSH    DPL                 ;SAVE DATA POINTER
        PUSH    DPH                 ;
        MOV     PSW,#00001000B      ;SELECT REGISTER BANK 1
;       ...     ......
;       ...     ......
        POP     DPH                 ;RESTORE REGISTERS IN REVERSE ORDER
        POP     DPL
        POP     B
        POP     ACC
        POP     PSW                 ;RESTORE PSW AND RE-SELECT ORIGINAL
                                    ;REGISTER BANK
        RETI                        ;RETURN TO MAIN PROGRAM AND RESTORE
                                    ;INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Fig. 6.1; SP would contain 26H. This is the most general case; if the service routine does not alter the B register and data pointer, for example, the instructions saving and restoring those registers could be omitted.

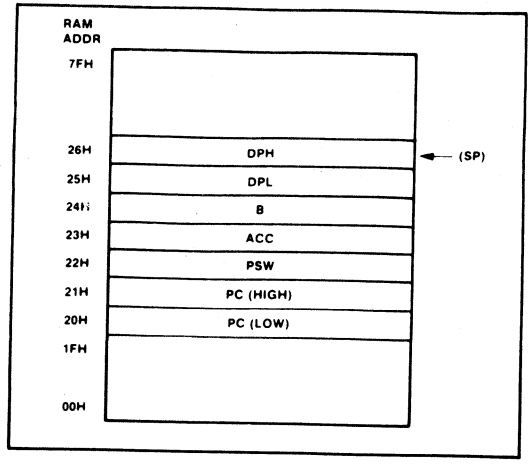


Fig. 6.1 The stack contents during an interrupt.

6.1.5 Passing parameters on the stack

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer, or simply pop the stack into registers before processing.

```
HEXASC: MOV   R0,SP
        DEC   R0                ;ACCESS LOCATION PARAMETER PUSHED INTO
        DEC   R0
        XCH   A,@R0             ;READ INPUT PARAMETER AND SAVE AC-
                                 CUMULATOR
        ANL   A,#0FH            ;MASK ALL BUT LOW-ORDER 4 BITS
        ADD   A,#2              ;ALLOW FOR OFFSET FROM MOVC TO TABLE
        MOVC  A,@A+PC           ;READ LOOK-UP TABLE ENTRY
        XCH   A,@R0             ;PASS BACK TRANSLATED VALUE AND RESTORE
                                 ;ACCUMULATOR
        RET                     ;RETURN TO BACKGROUND PROGRAM
ASCTBL: DB    '0'               ;ASCII CODE FOR 00H
        DB    '1'               ;ASCII CODE FOR 01H
        DB    '2'               ;ASCII CODE FOR 02H
        DB    '3'               ;ASCII CODE FOR 03H
        DB    '4'               ;ASCII CODE FOR 04H
        DB    '5'               ;ASCII CODE FOR 05H
        DB    '6'               ;ASCII CODE FOR 06H
        DB    '7'               ;ASCII CODE FOR 07H
        DB    '8'               ;ASCII CODE FOR 08H
        DB    '9'               ;ASCII CODE FOR 09H
        DB    'A'               ;ASCII CODE FOR 0AH
        DB    'B'               ;ASCII CODE FOR 0BH
        DB    'C'               ;ASCII CODE FOR 0CH
        DB    'D'               ;ASCII CODE FOR 0DH
        DB    'E'               ;ASCII CODE FOR 0EH
        DB    'F'               ;ASCII CODE FOR 0FH
```

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs can use different techniques for determining the handling of variables.

For example, the subroutine HEXASC converts a hexadecimal value to ASCII code for its low-order digit. It first reads a parameter stored on the stack by the calling program, then it uses the low-order bits to access a local 16-entry table holding ASCII codes, stores the appropriate code back on the stack and then returns. The accumulator contents are left unchanged.

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result to any destination register or port later. There is even the option of leaving a value on the stack if it will not be needed until later. The example below converts the three-digit BCD value, computed in the radix conversion example above, to a three-character string, calling a subroutine SP_OUT to an 8-bit code in the accumulator.

```
;       ...     ......
        PUSH    HUND
        CALL    HEXASC          ;CONVERT HUNDREDS DIGIT
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT HUNDREDS CHARACTER
        PUSH    TENONE
        CALL    HEXASC          ;CONVERT ONE'S PLACE DIGIT
                                ;BUT LEAVE ON STACK!
        MOV     A, TENONE
        SWAP    A               ;RIGHT-JUSTIFY TEN'S PLACE
        PUSH    ACC             ;CONVERT TEN'S PLACE DIGIT
        CALL    HEXASC
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT TEN'S PLACE CHARACTER
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT ONE'S PLACE CHARACTER
        ...     ......
```

6.1.6 N-way branching

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, fixed at assembly time.) Each has advantages for different applications.

In a typical N-way branch situation the potential destinations are generally known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of offset values in ROM to indicate the relative staring addresses of the several routines.

JMP @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the 8-bit unsigned accumulator contents with the contents of the 16-bit data

pointer, just like MOVC A,@A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a 16-bit addition is performed: a carry-out from the lower-order 8-bits may be propagated through the higher-order bits. In this case, neither the accumulator nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types (and sizes) of buffer memory for different speeds and options.

```
;
MEMSEL  EQU     R3
;
JUMP_4: MOV     A,MEMSEL
        MOV     DPTR,#JMPTBL
        MOVC    A,@A+DPTR
        JMP     @A+DPTR
JMPTBL: DB      MEMSP0-JMPTBL
        DB      MEMSP1-JMPTBL
        DB      MEMSP2-JMPTBL
        DB      MEMSP3-JMPTBL
MEMSP0: MOV     A,@R0           ;READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX    A,@R0           ;READ 256 BYTE EXTERNAL RAM
        RET
MEMSP2: MOV     DPL,R0          ;READ 64K BYTE EXTERNAL RAM
        MOV     DPH,R1
        MOVX    A,@DPTR
        RET
MEMSP3: MOV     A,R1            ;READ 4K BYTE EXTERNAL RAM
        ANL     A,#07H
        ANL     P1,#11111000B
        ORL     P1,A
        MOVX    A,@R0
        RET
```

To use this approach, the size of jump table plus the length of the alternate routines must be less than 256 bytes. The jump table and routines may be located anywhere in program memory and are independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all residing in the same 2K page of memory, the following technique may be used. In the printing terminal example, this sequence could process 128 different codes for ASCII characters arriving via the 8051 serial port.

```
;
OPTION   EQU    R3
;        ...    ......
;        ...    ......
JMP128:  MOV    A,OPTION
         RL     A              ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
         MOV    DPTR,#INSTBL   ;FIRST ENTRY IN JUMP TABLE
         JMP    @A+DPTR        ;JUMP INTO JUMP TABLE
;        ...    ......         ;
INSTBL:  AJMP   PROC00         ;128 CONSECUTIVE
         AJMP   PROC01         ;AJMP INSTRUCTIONS
         AJMP   PROC02
;        ...    ......
;        ...    ......
         AJMP   PROC7E
         AJMP   PROC7F
```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining characters all causing a branch to a common routine for entering the character into the output queue.

6.1.7 Computing branch destinations at run time

In some situations, 128 options are insufficient, the destination routines may cross a 2 K page boundary, or a branch destination is not known at assembly time (for whatever reason), and therefore cannot be easily included in the assembled code. These situations can all be handled by computing the destination address at run time with standard arithmetic or table look-up instructions, then performing an indirect branch to that address. There are two simple ways to execute this final step, assuming the 16-bit destination address has already been computed. The first is to load the address into the DPH and DPL registers, clear the accumulator and branch using the JMP @A+DPTR instruction; the second is to push the destination address onto the stack, low-order byte first (so as to mimic a call instruction) then pop that address into the PC by performing a return instruction. This also adjusts the stack pointer to its previous value. The code segment below illustrates the latter possibility.

```
;
RTEMP    EQU    R7
;        ...    ......
JMP256:  MOV    DPTR,#ADRTBL   ;FIRST ADDRESS TABLE ENTRY
         MOV    A,OPTION       ;LOAD INDEX INTO TABLE
         CLR    C
         RLC    A              ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
         JNC    LOW128
         INC    DPH            ;FIX BASE IF INDEX >127.
LOW128:  MOV    RTEMP,A        ;SAVE ADJUSTED ACC FOR SECOND READ
         INC    A              ;READ LOW-ORDER BYTE FIRST
         MOVC   A,@A+DPTR      ;GET LOW-ORDER BYTE FROM TABLE
         PUSH   ACC
         MOV    A,RTEMP        ;RELOAD ADJUSTED ACC
         MOVC   A,@A+DPTR      ;GET HIGH-ORDERED BYTE FROM TABLE
         PUSH   ACC
```

```
;
;         THE TWO ACC PUSHES HAVE PRODUCED
;         A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;         TO THE DESIRED STARTING ADDRESS.
;         IT MAY BE REACHED BY POPPING THE STACK
;         INTO THE PC.
          RET
;         ...     ......
;         ...     ......
;         ...     ......
ADRTBL:   DW      PROC00          ;UP TO 256 CONSECUTIVE DATA
          DW      PROC01          ;WORDS INDICATING STARTING ADDRESSES
;         ...     ......
;         ...     ......
          DW      PROCFF
;
```

6.1.8 In-line-code parameter passing

Parameters can be passed by loading appropriate registers with values before calling the subroutine. This technique is inefficient if many of the parameters are constants, since each would require a separate register to carry it, and a separate instruction to load the register each time the routine is called.

If the routine is called frequently, a more code-efficient way to transfer constants is 'in-line-code' parameter passing. The constants are actually part of the program code, immediately following the call instruction. The subroutine determines where to find them from the return address on the stack, and then reads the parameters it needs from program memory.

For example, assume a utility named ADDBCD adds a 16-bit packed BCD constant with a two-byte BCD variable in internal RAM and stores the sum in a different two-byte buffer. The utility must be given the constant and both buffer addresses. Rather than using four working registers to carry this information, all four bytes could be inserted into program memory each time the utility is called. Specifically, the calling sequence below invokes the utility to add 1234 (decimal) with the string at internal RAM address 56H, and store the sum in a buffer at location 78H.

The ADDBCD subroutine determines at what point the call is made by popping the return address from the stack into the data pointer high- and low-order bytes. A MOVC instruction then reads the parameters from the program memory as they are required. When complete, ADDBCD resumes execution by jumping to the instruction following the last parameter.

```
          ...     ......
          CALL    ADDBCD
          DW      1234H           ;BCD CONSTANT
          DB      56H             ;SOURCE STRING ADDRESS
          DB      78H             ;DESTINATION STRING ADDRESS
          ...     ......          ;CONTINUATION OF PROGRAM
;
;
;
```

```
ADDBCD: POP   DPH          ;POP RETURN ADDRESS INTO DPTR
        POP   DPL
        MOV   A,#2         ;INDEX FOR SOURCE STRING PARAMETER
        MOVC  A,@A+DPTR    ;GET SOURCE STRING LOCATION
        MOV   R0,A
        MOV   A,#3         ;INDEX FOR DESTINATION STRING PARAMETER
        MOVC  A,@A+DPTR    ;GET DESTINATION ADDRESS
        MOV   R1,A
        MOV   A,#1         ;INDEX FOR 16-BIT CONSTANT LOW BYTE
        MOVC  A,@A+DPTR    ;GET LOW-ORDER VALUE
        ADD   A,@R0        ;COMPUTE LOW-ORDER BYTE OF SUM
        DA    A            ;DECIMAL ADJUST FOR ADDITION
        MOV   @R1,A        ;SAVE IN BUFFER
        INC   R0
        INC   R1
        CLR   A            ;INDEX FOR HIGH-BYTE = 0
        MOVC  A,@A+DPTR    ;GET HIGH-ORDER CONSTANT
        ADDC  A,@R0
        DA    A            ;DECIMAL ADJUST FOR ADDITION
        MOV   @R1,A        ;SAVE IN BUFFER
        MOV   A,#4         ;INDEX FOR CONTINUATION OF PROGRAM
        JMP   @A+DPTR      ;JUMP BACK INTO MAIN PROGRAM
```

This example illustrates several points:

1) The 'subroutine' does not end with a normal return statement; instead, an indirect jump relative to the data pointer returns execution to the first instruction after the parameter list. The two initial POP instructions correct the stack pointer contents.

2) Either an ACALL or LCALL works with the subroutine, since each pushes the address of the next instruction or data byte onto the stack. The call may be made from anywhere in the full 8051 address space, since the MOVC instruction acesses all 64K bytes.

3) The parameters passed to the utility can be listed in whatever order is most convenient, which may not be that in which they are used. The utility has essentially 'random access' to the parameter list, by loading the appropriate constant into the accumulator before each MOVC instruction.

4) Other than the data pointer, the whole calling and processing sequence only affects the accumulator, PSW and pointer registers. The utility could have pushed these registers onto the stack (after popping the parameter list starting address), and popped them before returning.

Passing parameters through the in-line-code can be used in conjunction with other variable passing techniques.

The utility can also get input variables from working registers or from the stack, and return the output to the registers or to the stack.

6.2 Peripheral interfacing techniques

6.2.1 I/O port reconfiguration (first method)

I/O ports often must transmit or receive parallel data in formats other than as 8-bit bytes. For example, if an application requires three five-bit latched output ports (called X, Y and Z). These 'virtual' ports could be mapped onto the pins of 'physical' ports 1 and 2 (see Fig. 6.2)

	PORT "Z"					PORT "Y"					PORT "X"				
-	PZ0	PZ1	PZ2	PZ3	PZ4	PY4	PY3	PY2	PY1	PY0	PX4	PX3	PX2	PX1	PX0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

Fig. 6.2 Mapping of virtual ports onto the physical ports.

This pin assignment leaves P2.7 free for use as a test pin, input data pin, or control output through software.

It should be noted that the bits of port Z are reversed. The highest-order port Z pin corresponds to pin P2.2, and the lowest-order pin of port Z is P2.6, due to P.C. board layout considerations. When connecting an 8051 to an immediately adjacent keyboard column decoder or another device with weighted inputs, the corresponding pins may not be aligned. The interconnections must be 'scrambled' to compensate either with interwoven circuit board traces or through software (see below).

```
PX_MAP  DATA   20H
PY_MAP  DATA   21H
PZ_MAP  DATA   22H
;       ...    ....
OUT_PX: ANL    A,#00011111B      ;CLEAR BITS ACC.7 - ACC. 5
        MOV    PX_MAP,A          ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P1            ;UPDATE PORT 1 OUTPUT LATCH
        RET
;       ....   ....
OUT_PY: MOV    PY_MAP,A          ;SAVE IN MAP BYTE
        ACALL  OUT_P1            ;UPDATE PORT 1
        ACALL  OUT_P2            ;AND PORT 2 OUTPUT LATCHES
        RET
;       ....   ....
OUT_PZ: MOV    PZ_MAP,A          ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P2            ;UPDATE PORT 2.
        RET
;       ....   ....
;       ....   ....
OUT_P1: MOV    A,PY_MAP          ;OUTPUT ALL P1 BITS
        SWAP   A
        RL     A                 ;SHIFT PY_MAP LEFT 5 BITS
        ANL    A,#11100000B      ;MASK OUT GARBAGE
        ORL    A,PX_MAP          ;INCLUDE PX_MAP BITS
        MOV    P1,A
        RET
```

```
;           ....        ....
OUT_P2:     MOV     C,PZ_MAP.0      ;LOAD CY WITH P2.6 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PZ_MAP.1      ;LOAD CY WITH P2.5 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PZ_MAP.2      ;LOAD CY WITH P2.4 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PZ_MAP.3      ;LOAD CY WITH P2.3 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PZ_MAP.4      ;LOAD CY WITH P2.2 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PY_MAP,4      ;LOAD CY WITH P2.1 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            MOV     C,PY_MAP.3      ;LOAD CY WITH P2.0 BIT
            RLC     A               ;AND SHIFT INTO ACC.
            SETB    ACC.7           ;(ASSUMING INPUT ON P2.7)
            MOV     P2.A
            RET
```

Writing to the virtual ports must not affect any other pins. Since the virtual output algorithms are non-trivial, a subroutine is necessary for each port: OUT_PX, OUT_PY, OUT_PZ. Each of these subroutines is called with data to output right-justified in the accumulator, and any data in bits ACC.7-ACC.5 is insignificant. Each subroutine saves the data in a 'map' variable for the virtual port, then calls other subroutines which use the data in the various map bytes to compute and output the 8-bit pattern required for each physical port affected. The two-level structure of the above subroutines can be modified somewhat if code efficiency and execution speed are critical: incorporate the code shown as subroutines OUT_P1 and OUT_P2 directly into the code for OUT_PX and OUT_PZ, in place of the corresponding ACALL instructions. OUT_PY would not be changed, but now the destinations for its ACALL instructions would be alternate entry points in OUT_PX, instead of isolated subroutines.

6.2.2 I/O port reconfiguration (second method)

A more difficult situation arises if two sections of code which write to the same port or register, or call virtual output routines like those above, need to be executed at different interrupt levels. As an example, suppose the background program wants to rewrite port X (using the port associations in the previous example), and has computed the bit pattern required for P1. An interrupt is detected just before the MOV P1,A instruction, and the service routine tries to write to port Y. The service routine would correctly update P1 and P2, but upon returning to the background program P1 is immediately re-written with the data computed before the interrupt! Now pins P2.1 and P2.0 indicate (correctly) data written to port Y in the interrupt routine, but the earlier data written to P1.7-P1.5 is no longer valid. The same sort of confusion could arise if a high-level interrupt disrupted such a output sequence.

One solution is to disable interrupts around any section of the code which must not be interrupted (this is called a 'critical section'), but this would adversely affect interrupt latency. Another solution is to have interrupt routines set or clear a flag ('semaphore') when a common resource is altered - a rather complex and elaborate system.

An easier way to ensure that any instruction which writes the port X field of P1 does not change the port Y field pins from their state at the beginning of that instruction, is given below. A number of 8051 operations read, modify, and then write the output port latches all in one instruction. These are arithmetic and logical instructions (INC, DEC, ANL, ORL, etc.), where an addressed byte is both the destination variable and one of the source operands. Using these instructions, instead of data moves, eliminates the critical section problem entirely.

```
OUT_PX: ANL   P1,#11100000B     ;CLEAR BITS P1.4 - P1.0
        ORL   P1,A              ;SET P1 PIN FOR EACH ACC BIT SET
        RET
;       ...   ......
;       ...   ......
OUT_PY: MOV   B,#20H
        MUL   AB                ;SHIFT  B   A  LEFT 5 BITS.
        ANL   P1,#00011111B     ;CLEAR PY FIELD OF PORT 1
        ORL   P1,A              ;SET PY BITS ON PORT 1
        MOV   A,B               ;LOAD 2 BITS SHIFTED INTO B
        ANL   P2,#11111100B     ;AND UPDATE P2
        ORL   P2,A
        RET
;       ...   ......
;       ...   ......
OUT_PZ: RRC   A                 ;MOVE ORIGINAL ACC.0 INTO CY
        MOV   P2.6,C            ;AND STORE TO PIN P2.6.
        RRC   A                 ;MOVE ORIGINAL ACC.1 INTO CY
        MOV   P2.5,C            ;AND STORE TO PIN P2.5.
        RCC   A                 ;MOVE ORIGINAL ACC.2 INTO CY
        MOV   P2.4,C            ;AND STORE TO PIN P2.4.
        RRC   A                 ;MOVE ORIGINAL ACC.3 INTO CY
        MOV   P2.3,C            ;AND STORE TO PIN P2.3.
        RRC   A                 ;MOVE ORIGINAL ACC.4 INTO CY
        MOV   P2.2,C            ;AND STORE TO PIN P2.2.
        RET
```

6.2.3 Software delay timing

Many 8051 applications involve exact control over output timing. A software generated output strobe, for instance, might have to be exactly 50μs. wide. The DJNZ operation can insert a one-instruction software delay into a piece of code, adding a moderate time delay of two instruction cycles per iteration. For example, two instructions can add a 49μs. software delay loop to code to generate a pulse on the WR pin.

```
CLR     WR
MOV     R2,#24
DJNZ    R2,$
SETB    WR
```

The dollar sign is this example is a special character meaning 'the address of this instruction'. It can be used to eliminate instruction labels on nearby source lines.

6.2.4 Serial port and timer configuration

Configuring the 8051's serial port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

To choose one arbitrary example, assume the 8051 should communicate with a standard CRT operating at 2400 baud. Each character is transmitted as seven data bits, odd parity, and one stop bit. The resulting character rate is 2400 baud/9 bits, approximately 265 characters per second.

For the sake of clarity, the transmit and receive subroutines here are driven straightforward software status polling code rather than interrupts. The serial port must be intitialized to 8-bit UART mode (SM0, SM1 = 0,1), enabled to receive all messages (SM2 = 0, REN = 1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. All this can be set up with instruction at label SPINIT.

Timer 1 will be used in auto-reload mode as a baud rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1MHz internal clock by

$$\frac{1 \times 10^6}{(32)(2400)}$$

which is approximately 13 instruction cycles. The timer must reload the value -13 or 0F3H, as shown by the code at label TIINIT.

```
;
;         INITIALIZE SERIAL PORT
;         FOR 8-BIT UART MODE
;         & SET TRANSMIT READY FLAG.
SPINIT:   MOV     SCON,#01010010B
;         INITIALIZE TIMER 1 FOR
;         AUTO-RELOAD AT 32 X 2400HZ
;         (T0 USED AS GATED 16-BIT COUNTER.)
;TIINIT   MOV     TMOD,#00101101B
          MOV     TH,#-13
          SETB    TR1
;         ...     .....
;
```

6.2.5 Simple serial I/O drivers

SP_OUT is a simple subroutine to transmit the character passed to it in the accumulator. First it must compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and then return.

SP_IN is an equally simple routine which waits until a character is received, sets the carry flag if there is an odd-parity error, and returns the masked 7-bit code in the accumulator.

```
;
;SP_OUT  ADD ODD PARITY TO ACC AND
;        TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT: MOV    C,P
        CPL    C
        MOV    ACC.7,C
        JNB    TI,$
        CLR    TI
        MOV    SBUF,A
        RET
;
;
;SP_IN   INPUT NEXT CHARACTER FROM SERIAL PORT.
;        SET CARRY IF ODD-PARITY ERROR
;
SP_IN:  JNB    RI,$
        CLR    RI
        MOV    A,SBUF
        MOV    C,P
        CPL    C
        ANL    A,#7FH
        RET
```

6.2.6 Transmitting serial port character strings

Any application which transmits characters through the serial port to an ASCII output device will on occasion need to output 'canned' messages, including error messages, diagnostics, or operator instructions. These character strings are most easily defined with in-line data bytes defined with the DB directive.

```
CR      EQU    0DH              ;ASCII CARRIAGE RET
LF      EQU    0AH              ;ASCII LINE-FEED
ESC     EQU    1BH              ;ASCII ESCAPE CODE
;       ...    .....
        CALL   XSTRING
        DB     CR,LF            ;NEW LINE
        DB     'HELLO'          ;MESSAGE
        DB     ESC              ;ESCAPE CHARACTER
;
;       (CONTINUATION OF PROGRAM)
;
;       ...    .....
XSTRING: POP   DPH              ;LOAD DPTR WITH FIRST CHARACTER
        POP    DPL
XSTR_1: CLR    A                ;(ZERO OFFSET)
        MOVC   A,@A+DPTR        ;FETCH FIRST CHARACTER OF STRING
XSTR_2: JNB    TI,$             ;WAIT UNTIL TRANSMITTER READY
        CLR    TI               ;MARK AS NOT READY
        MOV    SBUF,A           ;OUTPUT NEXT CHARACTER
        INC    DPTR             ;BUMP POINTER
```

```
        CLR     A
        MOVC    A,@A+DPTR               ;GET NEXT OUTPUT CHARACTER
        CJNE    A,#ESC,XSTR__2          ;LOOP UNTIL ESCAPE READ
        MOV     A,#1
        JMP     @A+DPTR                 ;RETURN TO CODE AFTER ESCAPE
```

6.2.7 Recognizing and processing special cases

Before operating on the data it receives, a subroutine might give 'special handling' to certain input values. Consider a word processing device which receives ASCII characters through the 8051's serial port and drives a thermal hard-copy printer. A standard routine translates most characters into bit patterns, but certain control characters (<DEL>, <CR>, <LF>, <BEL>, <ESC>, or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the NUL value, 00H, and processed with the printing characters. The CJNE operation provides essentially a one-instruction CASE statement.

```
;
CHAR    EQU     R7                      ;CHARACTER CODE VARIABLE
;
INTERP: CJNE    CHAR,#7FH, INTP__1      ;SKIP UNLESS RUBOUT
;       ...     .....                   (SPECIAL ROUTINE FOR RUBOUT CODE)
        RET
INTP__1: CJNE   CHAR,#07H,INTP__2       ;SKIP UNLESS BELL
;       ...     .....                   (SPECIAL ROUTINE FOR BELL CODE)
        RET
INTP__2: CJNE   CHAR,#0AH,INTP__3       ;SKIP UNLESS LFEED
;       ...     .....                   (SPECIAL ROUTINE FOR LFEED CODE)
        RET
INTP__3: CJNE   CHAR,#0DH,INTP__4       ;SKIP UNLESS RETURN
;       ...     .....                   (SPECIAL ROUTINE FOR RETURN CODE)
        RET
INTP__4: CJNE   CHAR,#1BH,INTP__5       ;SKIP UNLESS ESCAPE
;       ...     .....                   (SPECIAL ROUTINE FOR ESCAPE CODE)
        RET
INTP__5: CJNE   CHAR,#20H,INTP__6       ;SKIP UNLESS SPACE
;       ...     .....                   (SPECIAL ROUTINE FOR SPACE CODE)
        RET
INTP__6: JC     PRINTC                  ;JUMP IF CODE  20 H
        MOV     CHAR,#0                 ;REPLACE CONTROL CHARACTER WITH
                                        ;NULL CODE
PRINTC:                                 ;PROCESS STANDARD PRINTING
;       ...     .....                   ;CHARACTER
        RET
;
```

6.2.8 Synchronizing timer overflows

8051 timer overflows automatically generate an internal interrupt request, which will vector program execution to the appropriate interrupt service routine if the interrupts are enabled and no other service routines are in progress at the time. However, it is not predictable exactly how long it will take to reach the service routine. The service routine call takes two instruction cycles, but 1, 2, or 4 additional cycles may be necessary to complete the instruction in progress. If the background program ever disables interrupts, the response latency could further increase by a few instruction cycles. (Critical sections generally involve simple instruction sequences - rarely multiplies or divides.) Interrupt response delay is generally negligible, but certain time-critical applications must take the exact delay into account. For example, generating interrupts with Timer 1 every millisecond (1000 instruction cycles) or so would normally call for reloading it with the value -1000 (0FC30H). If the interrupt interval (average over time) must be accurate to 1 instruction cycle, the 16-bit value reload into the timer must be computed, taking into account when the timer actually overflowed.

This simply requires reading the appropriate timer, which has been incremented each cycle since the overflow occured. A sequence like the one below can stop the timer, compute how much time should elapse before the next interrupt, and reload and restart the timer. The double-precision calculation shown here compensates for any amount of timer overrun within the maximum interval. Note that it also takes into account that the timer is stopped for seven instruction cycles in the process. All interrupts are disabled, so a higher priority request will not be able to disrupt the time-critical code section.

```
;       ...     .....
        CLR     EA                      ;DISABLE ALL INTERRUPTS
        CLR     TR1                     ;STOP TIMER 1
        MOV     A,#LOW(-1000+7)         ;LOAD LOW-ORDER DESIRED COUNT
        ADD     A,TL1                   ;CORRECT FOR TIMER OVERRUN
        MOV     TL1,A                   ;RELOAD LOW-ORDER BYTE.
        MOV     A,#HIGH(-1000+7)        ;REPEAT FOR HIGH-ORDER BYTE.
        ADDC    A,TH1
        MOV     TH1,A
        SETB    TR1                     ;RESTART TIMER
;       ...     .....
```

6.2.9 Reading a timer/counter without disrupting the timing process

The preceding example simply stopped the timer before changing its contents. This is normally done when reloading a timer so that the time at which the timer is started (i.e. the 'run' flag is set) can be exactly controlled. There are situations, though, when it is desired to read the current count without disrupting the timing process. The 8051 timer/counter registers can all be read or written while they are running, but a few precautions must be taken.

Suppose the subroutine RDTIME should return in R1, R0, a 16-bit value indicating the count in Timer 0. The instant at which the count was sampled is not as critical as the fact that the value returned must have been valid at some point while the routine was in progress. There is a potential problem that between reading the two halves, a low-order register overflow might increment the high-order register, and the two data bytes returned would be 'out of phase'. The solution is to read the high-order byte first, then the low-order byte, and then confirm that the high-order byte has not changed. If it has, repeat the whole process.

```
RDTIME:  MOV    A,TH0           ;SAMPLE TIMER0 (HIGH)
         MOV    R0,TL0          ;SAMPLE TIMER0 (LOW)
         CJNE   A,TH0,RDTIME    ;REPEAT IF NECESSARY
         MOV    R1,A            ;STORE VALID READ
         RET
```

6.3 Connections to peripherals

This section shows in very general terms some basic circuits for expanding the 8051/80C51 family. The schematics included in this section should give the designer an insight into connecting external peripherals and memories to the 8051.

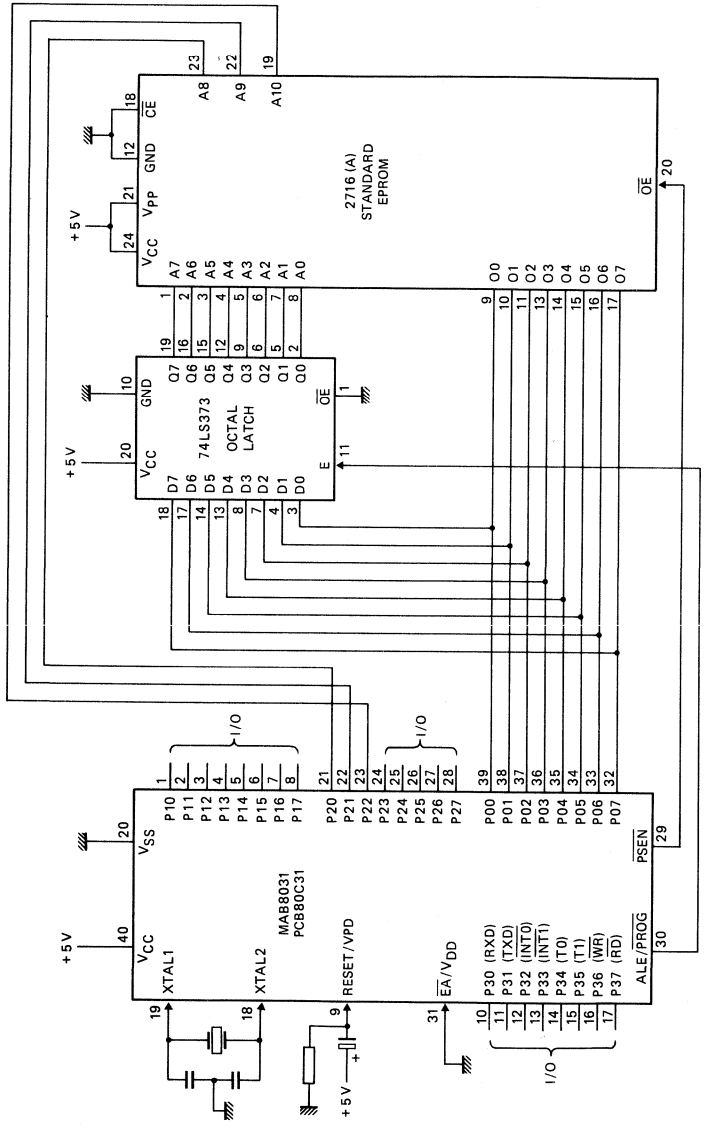


Fig. 6.3 External program memory using a standard EPROM.

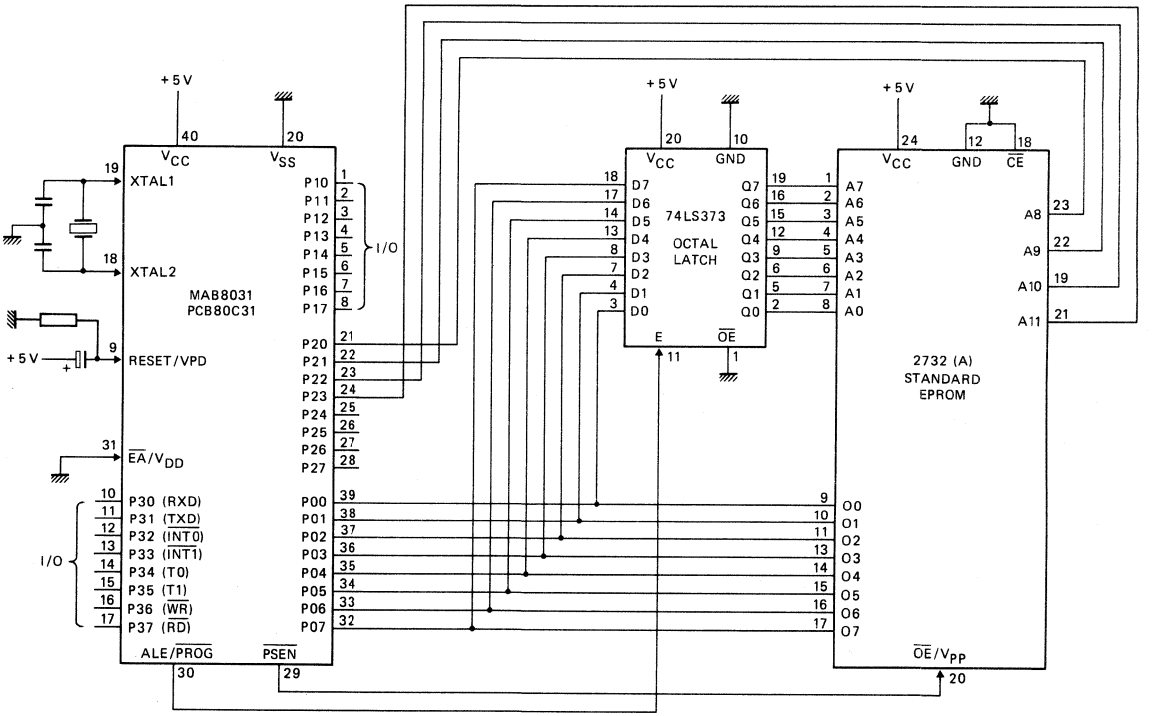


Fig. 6.4 External program memory using a standard EPROM.

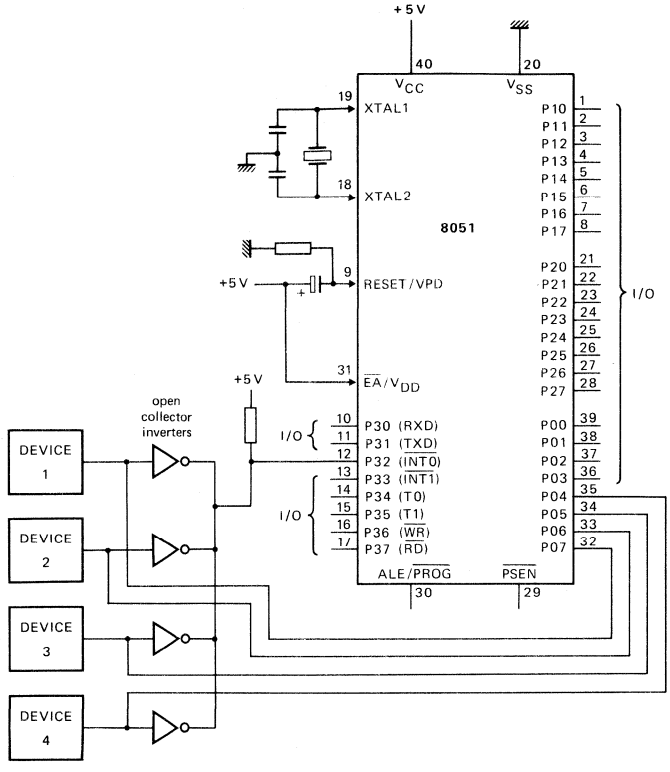


Fig. 6.5 Multiple interrupt sources.

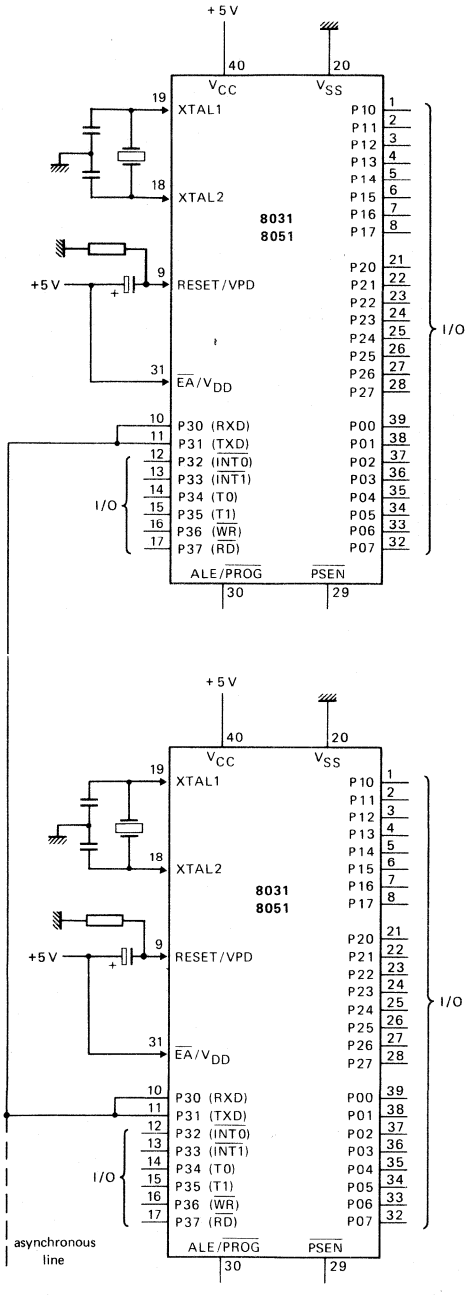


Fig. 6.6 Multiple 8051's using half-duplex serial communication.

3.1 The SC80/83C451 microcontroller

CONTENTS – THE SC80/83C451 MICROCONTROLLER

		page
1.0	DESCRIPTION	3.1-5
2.0	FEATURES	3.1-5
3.0	PIN DESIGNATION	3.1-8
4.0	FUNCTIONAL DESCRIPTION	3.1-11
	4.1 Input/Output operation	3.1-11
	4.1.1 Ports 4 and 5	3.1-11
	4.1.2 Port 6	3.1-11
	4.1.2.1 Control status register	3.1-13
5.0	OPERATION OF PORT 6 APPLICATION NOTE	3.1-15

1.0 DESCRIPTION

The SC80C451/83C451 is an I/O expanded, single-chip microcontroller fabricated in high density epitaxial CMOS technology. The SC80C451/83C451 is a functional extension of the PCB80C31/51 microcontroller with three additional I/O ports and four additional I/O control lines. These four extra control lines are associated with Port 6 and facilitate high-speed asynchronous I/O functions. Two versions of the microcontroller exist:

- SC80C451: ROM-less version of SC83C451
- SC83C451: 4 K bytes mask-programmable ROM, 128 bytes RAM

Each version is offered in a 68-pin PLCC package and a 64-pin DIL package.

The SC80/83C451 has 56 (PLCC package) or 52 (DIL) I/O lines; two 16-bit timer/counters; a five source, two priority level, nested interrupt structure; a serial I/O port for either a full duplex UART, I/O expansion or multiprocessor communications; an on-chip oscillator and clock circuits. The SC80/83C451 has two software selectable modes of reduced activity for further power reduction -Idle mode and Power-down mode. Idle mode freezes the CPU while allowing the RAM, timers, serial port and interrupt system to continue functioning. Power-down mode freezes the oscillator causing all other chip functions to become inoperative while maintaining the status of the RAM.

2.0 FEATURES

- 8-bit CPU
- 4 K bytes of ROM and 128 bytes RAM on the SC83C451
- Seven 8-bit ports (PLCC), Six 8-bit ports and one 4-bit port (DIL)
- High-speed parallel Port 6 with mailbox function
- Two 16-bit timer/counters
- Full duplex serial I/O
- Two external interrupts
- External memory addressing capability
- Two reduced power modes; Idle and Power-down

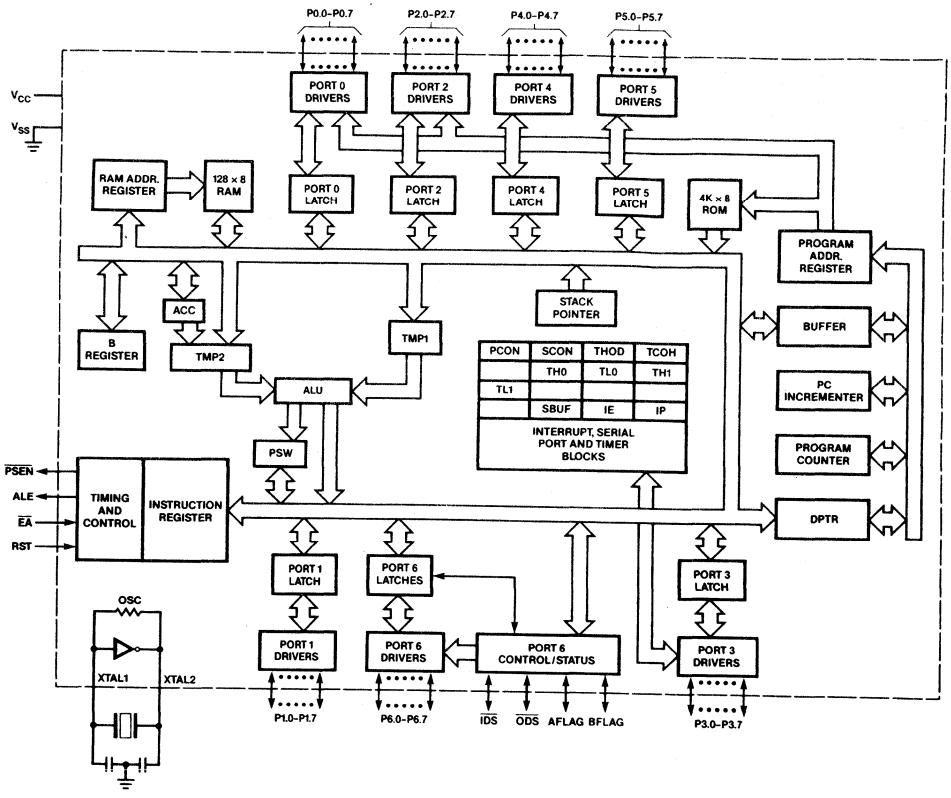
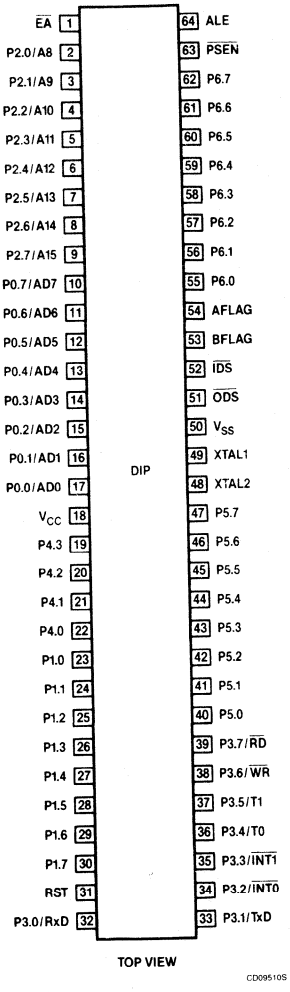
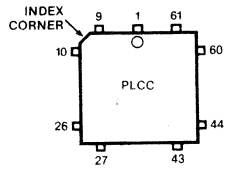


Fig. 1 Block diagram of SC80C451/83C451.





TOP VIEW

CD09520S

Pin	Function	Pin	Function
1	$\overline{EA}$	35	RST
2	P2.0/A8	36	P3.0/R ⧹ D
3	P2.1/A9	37	P3.1/T ⧹ D
4	P2.2/A10	38	P3.2/$\overline{INT0}$
5	P2.3/A11	39	P3.3/$\overline{INT1}$
6	P2.4/A12	40	P3.4/T0
7	P2.5/A13	41	P3.5/T1
8	P2.6/A14	42	P3.6/$\overline{WR}$
9	P2.7/A15	43	P3.7/$\overline{RD}$
10	P0.7/AD7	44	P5.0
11	P0.6/AD6	45	P5.1
12	P0.5/AD5	46	P5.2
13	P0.4/AD4	47	P5.3
14	P0.3/AD3	48	P5.4
15	P0.2/AD2	49	P5.5
16	P0.1/AD1	50	P5.6
17	P0.0/AD0	51	P5.7
18	V_{CC}	52	XTAL2
19	P4.7	53	XTAL1
20	P4.6	54	V_{SS}
21	P4.5	55	$\overline{ODS}$
22	P4.4	56	$\overline{IDS}$
23	P4.3	57	BFLAG
24	P4.2	58	AFLAG
25	P4.1	59	P6.0
26	P4.0	60	P6.1
27	P1.0	61	P6.2
28	P1.1	62	P6.3
29	P1.2	63	P6.4
30	P1.3	64	P6.5
31	P1.4	65	P6.6
32	P1.5	66	P6.7
33	P1.6	67	$\overline{PSEN}$
34	P1.7	68	ALE

Fig. 2 SC80/83C451 pinning diagram.

3.0 PIN DESIGNATION

MNEMONICS	PIN			FUNCTION
	DIP	PLCC	TYPE	
V_{SS}	50	54		Ground: 0 V reference
V_{CC}	18	18		Power supply: +5 V
P0.0 - P0.7	17-10	17-10	I/O	Port 0: An 8-bit open drain, bidirectional I/O port. Port 0 is also the multiplexed low-order address and data bus during accesses to external memory, and outputs instruction bytes during program verification. External pull-ups are required during program verification. Port 0 can sink/source eight LS TTL inputs.
P1.0 - P1.7	23-30	27-34	I/O	Port 1: An 8-bit bidirectional I/O port with internal pull-ups. Port 1 receives the low-order address bytes during program verification in the SC83C451. Port 1 can sink/source three LS TTL inputs, and drive CMOS inputs without external pull-ups.
P2.0 - P2.7	2-9	2-9	I/O	Port 2: An 8-bit bidirectional I/O port with internal pull-ups. Port 2 emits the high-order address bytes during accesses to external memory and receives the high-order address bits and control signals during program verification in the SC83C451. Port 2 can sink/source three LS TTL inputs and drive CMOS inputs without external pull-ups.
P3.0 - P3.7	32-39	36-43	I/O	Port 3: An 8-bit bidirectional I/O port with internal pull-ups. Port 3 can sink/source three LS TTL inputs and drive CMOS inputs without external pull-ups. Port 3 also serves the following special functions listed below:
	32	36	I	RXD (P3.0): Serial input port
	33	37	0	TXD (P3.1): Serial output port
	34	38	I	$\overline{\text{INT0}}$ (P3.2): External interrupt 0
	35	39	I	$\overline{\text{INT1}}$ (P3.3): External interrupt 1
	36	40	I	T0 (P3.4): Timer 0 external input
	37	41	I	T1 (P3.5): Timer 1 external input
	38	42	0	$\overline{\text{WR}}$ (P3.6): External data memory write strobe
	39	43	0	$\overline{\text{RD}}$ (P3.7): External data memory read strobe

P4.0 - P4.3 P4.4 - P4.7	22-19, 26-23 22-19		I/O I/O	Port 4: A 4 or 8-bit bidirectional port with internal pull-ups. Port 4 can sink/source three LS TTL inputs and drive CMOS inputs without external pull-ups.
P5.0 - P5.7	40-47	44-51	I/O	Port 5: An 8-bit port with internal pull-ups. Port 5 can sink/source three LS TTL inputs and drive CMOS inputs without external pull-ups.
P6.0 - P6.7	55-62	59-66	I/O	Port 6: A specialized 8-bit bidirectional I/O port with internal pull-ups. This special port can sink/source three LS TTL inputs and drive CMOS inputs without external pull-ups. Port 6 can be used in a strobed or non-strobed mode of operation, and in conjunction with four control pins that serve the functions listed below.
				Port 6 Control lines:
$\overline{ODS}$	51	55	I	$\overline{ODS}$: Output data strobe
$\overline{IDS}$	52	56	I	$\overline{IDS}$: Input data strobe
BFLAG	53	57	I/O	BFLAG: A bidirectional I/O pin with internal pull-ups
AFLAG	54	58	I/O	AFLAG: A bidirectional I/O pin with internal pull-ups
RST	31	35	I	RESET: A high on this pin for two machine cycles while the oscillator is running resets the device. An internal pull-down resistor permits power-on reset using only a capacitor connected to the V_{CC}.
ALE	64	68	O	Address Latch Enable: An output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access, at which time one ALE is skipped. ALE can sink/source eight LS TTL inputs and drive CMOS inputs without an external pull-up.
$\overline{PSEN}$	63	67	O	Program Store Enable: This output is the read strobe to external program memory. $\overline{PSEN}$ is activated twice each machine cycle during fetches from external program memory; however, when executing out of external memory, two activations of $\overline{PSEN}$ are skipped during each access to external data memory. $\overline{PSEN}$ is not activated during fetches from internal program memory. $\overline{PSEN}$ can sink/source eight LSTTL inputs and drive CMOS inputs without an external pull-up.

$\overline{EA}$	1	1	I	Instruction Execution Control: When $\overline{EA}$ is held high, the CPU executes out of internal program memory, unless the program counter exceeds OFFFH. When $\overline{EA}$ is held low, the CPU executes out of external program memory. $\overline{EA}$ must never be allowed to float.
XTAL1	49	53	I	Crystal 1: An input to the inverting amplifier that forms the oscillator. This input receives the external oscillator when an external oscillator is used.
XTAL2	48	52	O	Crystal 2: An output of the inverting amplifier that forms the oscillator. This pin should be floated when an external oscillator is used.

4.0 FUNCTIONAL DESCRIPTION

As the SC80/83C451 is functionally equivalent to the PCB80C31/51 only the differences are described here. For further information on controller operation turn to the MAB8051/C51 microcontroller family section of this manual.

4.1 Input/Output operation

4.1.1 Ports 4 and 5

Ports 4 and 5 are bidirectional I/O ports with internal pull-ups. Port 4 is an 8-bit port (PLCC version) or a 4-bit port (DIL version). Port 4 and Port 5 pins, when ones are written to them, are pulled high by the internal pull-ups, and in this high-impedance state can be used as inputs. Port 4 and 5 are addressed with the special function register addresses shown in Table 1.

4.1.2 Port 6

Port 6 is a special 8-bit bidirectional I/O port with internal pull-ups. (see Fig 3.) This port can be used as a standard I/O port, or in a strobed mode of operation in conjunction with four special control lines: $\overline{ODS}$, $\overline{IDS}$, AFLAG, and BFLAG.

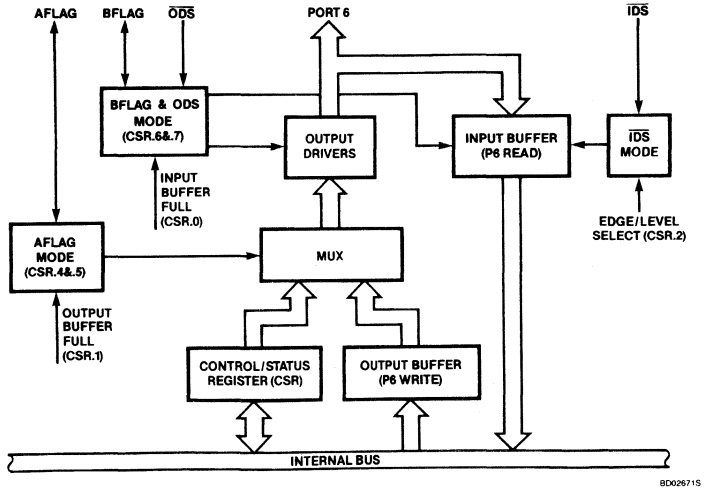


Fig. 3 Port 6 block diagram.

Port 6 operating modes are controlled by the Port 6 control status register (CSR). Port 6 and the CSR are addressed at the special function register addresses shown in Table 1. The following four control pins are used in conjunction with Port 6:

$\overline{ODS}$: Output data strobe input for Port 6. $\overline{ODS}$ can be programmed to control the Port 6 output drivers and the 'output buffer full' flag (OBF), or to clear only the OBF flag bit in the CSR (output-always mode). $\overline{ODS}$ is active low for output driver control. The OBF flag can be programmed to be cleared on the negative or positive edge of $\overline{ODS}$.

$\overline{IDS}$: Input data strobe input for Port 6. $\overline{IDS}$ is used to control the Port 6 input latch and 'input buffer full' flag (IBF) bit in the CSR. The input data latch can be programmed to be transparent when the $\overline{IDS}$ is low and latched on the positive transition of $\overline{IDS}$, or to latch only on the positive transition of $\overline{IDS}$. Correspondingly, the IBF flag is set on the negative or positive transition of $\overline{IDS}$.

AFLAG: A bidirectional I/O pin. AFLAG can be programmed to be an output set high or low under program control, or to output the state of the output buffer full flag. AFLAG can also be programmed to be an input which selects whether the contents of the output buffer, or the contents of the Port 6 control status register will be output on Port 6. This feature grants complete Port 6 status to external devices.

BFLAG: A bidirectional I/O pin. BFLAG can be programmed to be an output, set high or low under program control, or to output the state of the 'input buffer full' flag. BFLAG can also be programmed to input an enable signal for Port 6. When BFLAG is used as an enable input, Port 6 output drivers are in the high impedance state, and the input latch does not respond to the IDS strobe when BFLAG is high. Both features are enabled when BFLAG is low. This feature facilitates the use of the SC83C451 in bussed multiprocessor systems.

4.1.2.1 Control status register (CSR)

The control status register establishes the mode of operation for Port 6 and indicates the current status of the Port 6 I/O registers. All control status register bits can be read and written by the CPU, except bits 0 and 1, which are read only. Reset writes ones to bits 2 through 7 and writes zeros to bits 0 and 1.

CSR.0 - Input; Buffer Full Flag (IBF) (Read only)

The IBF bit is set to a logic 1 when Port 6 data is loaded into the input buffer under control of $\overline{IDS}$. This can occur on the negative or positive edge of $\overline{IDS}$, as determined by CSR.2. IBF is cleared when the CPU reads the input buffer register.

CSR.1 - Output Buffer Full Flag (OBF) (Read only)

The OBF flag is set to a logic 1 when the CPU writes to the Port 6 output data buffer. OBF is cleared by the positive or negative edge of $\overline{ODS}$, as determined by CSR.3.

CSR.2 - $\overline{IDS}$ Mode Select (IDSM)

When CSR.2 = 0, a low-to-high transition on the $\overline{IDS}$ pin sets the IBF flag. The Port 6 input buffer is loaded on the $\overline{IDS}$ positive edge. When CSR.2 = 1, a high-to-low transition on the $\overline{IDS}$ pin sets the IBF flag. The Port 6 input buffer is transparent when $\overline{IDS}$ is low, and latched when $\overline{IDS}$ is high.

CSR.3 - Output Buffer Full Flag Clear Mode (OBFC)

When CSR.3 = 1, the positive edge of the $\overline{ODS}$ input clears the OBF flag. When CSR.3 = 0, the negative edge of the $\overline{ODS}$ input clears the OBF flag.

CSR.4, CSR.5 - AFLAG Mode Select (MA0, MA1)

Bits 4 and 5 select the mode of operation for the AFLAG pin, as follows:

MA1	MA0	AFLAG
0	0	Logic 0 output
0	1	Logic 1 output
1	0	OBF flag output (CSR.1)
1	1	Select (SEL) input mode

The select (SEL) input mode is used to determine whether the Port 6 data register or the control status register is output on Port 6. When the select feature is enabled, the AFLAG input controls the source of Port 6 output data. A logic 0 on AFLAG input selects the Port 6 data register, and a logic 1 on the AFLAG input selects the control status register.

CSR.6, CSR.7 - BFLAG Mode Select (MB0, MB1)

Bits 6 and 7 select the mode of operation for the BFLAG pin, as follows:

MB1	MB0	BFLAG
0	0	Logic 0 output
0	1	Logic 1 output
1	0	IBF flag output (CSR.0)
1	1	Port enable (PE)

In the port enable mode, $\overline{IDS}$ and $\overline{ODS}$ inputs are disabled when BFLAG input is high. When the BFLAG input is low, the port is enabled for I/O.

Control status register bit pattern

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
(CSR)	MB1	MB0	MA1	MA0	OBFC	IDSM	OBF	IBF
	BFLAG mode select 0/0 = logic 0 output* 0/1 = logic 1 output* 1/0 = IBF output 1/1 = PE Input (0 = Select) (1 = disable I/O)		AFLAG mode select 0/0 = logic 0 output 0/1 = logic 1 output 1/0 = OBF output* 1/1 = SEL input (0 = Data) (1 = Control /status)		Output buffer flag clear mode 0 = negative edge of $\overline{ODS}$ 1 = Positive edge of $\overline{ODS}$	Input data strobe mode 0 = Positive edge of $\overline{IDS}$ 1 = Low level of $\overline{IDS}$	Output buffer full flag 0 = Output data buffer empty 1 = Output data buffer full	Input buffer full flag 0 = Input data buffer empty 1 = input data buffer full

* Output-always mode; MB1 = 0, MA1 = 1 and MA0 = 0. In this mode Port 6 is always enabled for output. $\overline{ODS}$ only clears the OBF flag.

Special function register addresses

Special function register addresses for the SC83C451 are identical to those of the PCB80C51, except for the additional registers listed in Table 1.

Table 1

Register address			Bit address							
Name	Symbol	Address	MSB							LSB
Port 4	P4	C0	C7	C6	C5	C4	C3	C2	C1	C0
Port 5	P5	C8	CF	CE	CD	CC	CB	CA	C9	C8
Port 6 data	P6	D8	DF	DE	DD	DC	DB	DA	D9	D8
Port 6 control status	CSR	E8	EF	EE	ED	EC	EB	EA	E9	E8

5.0 OPERATION OF PORT 6

APPLICATION NOTE

Microprocessor Products

INTRODUCTION

The features of the 83C451 are shared with the 80C51 or are conventional except for the operation of port 6. The flexibilty of this port facilitates high speed parallel data communications.

This applications note discusses the use of port 6 and is divided into the following sections:

1. Port 6 as a Processor Bus Interface.
2. Using Port 6 as a Standard Psuedo Bidirectional I/O Port.
3. Implementations of Parallel Printer Ports.

PORT 6 AS A PROCESSOR BUS INTERFACE

Port 6 allows use of the 83C451 as an element on a microprocessor type bus. The host processor could be a general purpose MPU or the data bus of a microcontroller like the 83C451 itself.

This feature allows single or multiple 83C451 controllers to be used on a bus as flexible peripheral processing elements. Applications might include keyboard scanners, serial I/O controllers, servo controllers, etc.

OPERATION

On reset port 6 is programmed correctly for use as a bus interface (see figure 2). This prevents the interface from disrupting data on the bus of the host processor during power up. Software initialization of the CSR (Control Status Register) is not required. A dummy read of port 6 may be required to clear the IBF (Input Buffer Full) flag since it could be set by turn on transients on the bus of the host processor. On reset the CSR of the 83C451 is programmed to allow the following:

1. AFLAG is an input controlling the port select function. If AFLAG is high the contents of the CSR is output on port 6 when the port is read by the host. If AFLAG is low, then the contents of the output latch is output when port 6 is read by the host.

2. BFLAG is an input controlling the port enable function. In this mode when BFLAG is high, the input latch and the output drivers are disabled and the flags are not affected by the IDS (Input Data Strobe) or ODS (Output Data Strobe) signals. When BFLAG is low, the port is enabled for reading and writing under the control of IDS and ODS pins.

Figure 1 shows one possible example of an 83C451 on a memory bus. This arrangement allows the main processor to query port 6 for flag status without interrupting the 83C451. If the address decoder, shown in figure 1, enables port

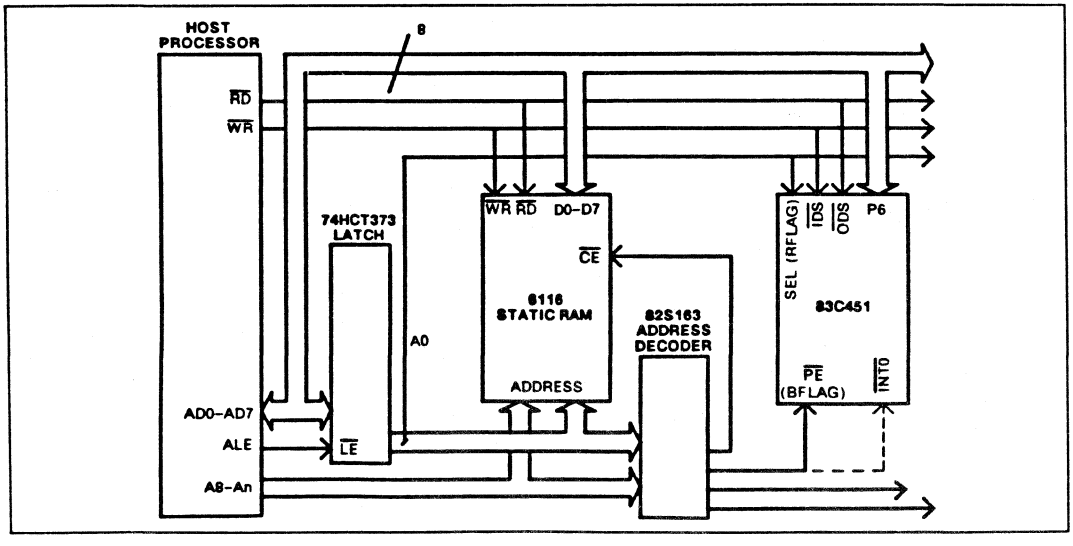


Figure 1. An 83C451 on a Microprocessor Memory Bus

6 on the 83C451 when the address is 8000H or 8001H, and the address line A0 controls the port select feature, then the host processor can read and write to port 6 using address 8000H. Since the port select function is being controlled by the address line A0, the CSR contents can be read by the host processor at address 8001H.

By testing the CSR contents in this way, the host processor can tell if new data has been written to the port 6 output latch since it last read the port or if the 83C451 has read the last byte that the host wrote to the port. Conversely the 83C451 can poll the flags in its CSR to see if the host processor has written to or read from port 6 since the last time it serviced the port.

If desired, an interrupt source for the 83C451 can be derived easily from the port enable source as shown by the dashed line in figure 1.

SOFTWARE EXAMPLES

To write to port 6 on the bus shown in figure 1, the host processor first reads the CSR contents at address 8001H, and tests the input buffer full flag (CSR bit 0). If the flag is clear the host writes a byte to address 8000H. This loads the input buffer latch of port 6 and sets the input buffer full flag.

Conversely the 83C451 polls the IBF flag and reads a byte from port 6 when it finds the flag set. The flag is automatically reset when this internal read occurs.

83C451 ROUTINE TO READ ONE BYTE FROM HOST VIA PORT 6

```
RCVR:   JNB CSR.0,RCVR      ;TEST IBF FLAG
        MOV A,P6            ;WHEN FLAG IS SET READ BYTE
        RET
```

80C51 ROUTINE TO WRITE ONE BYTE TO THE 83C451 PORT 6

If the host processor is an 80C51 the following routine will write a byte of data to the 83C451. The data involved is passed to the the routine through register 1.

```
XMIT:   MOV DPTR,8001H
TEST:   MOVX A,@DPTR        ;READ THE CSR
        JB ACC.0,TEST       ;TEST IBF FLAG
        MOV DPTR,8000H
        MOV A,R1
        MOVX @DPTR,A        ;WRITE DATA TO THE 451
        RET
```

83C451 ROUTINE TO WRITE ONE BYTE TO HOST VIA PORT 6

Routines for data transfer in the opposite direction are similar to the above two. The 83C451 version is given below.

```
XMIT:   JB CSR.1,XMIT       ;TEST OBF FLAG
        MOV P6,A            ;WRITE DATA
        RET
```

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
1	1	1	1	1	1		

Figure 2. CSR Programmed to Allow Port 6 as a Bus Interface

USING PORT 6 AS A STANDARD QUASI BIDIRECTIONAL I/O PORT

To use port 6 as a common I/O port, all of the control pins are tied to ground (see figure 3). On hardware reset, bits 2–7 in the CSR are set to one. Port operation and electrical characteristics become identical to port 1 on the 80C51 and the 83C451 ports 1, 4, and 5. No software initialization is required.

If desired, AFLAG and BFLAG can be used as outputs while port 6 is operating as a standard quasi-bidirectional I/O port (see figure 4). In this case, only IDS and ODS are tied to ground and the CSR is initialized to allow operation of AFLAG and BFLAG as simple outputs (see figure 5).

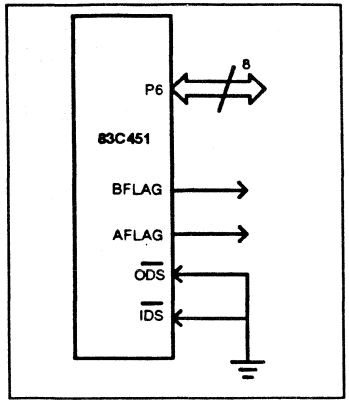


Figure 3. Standard I/O Port on Reset

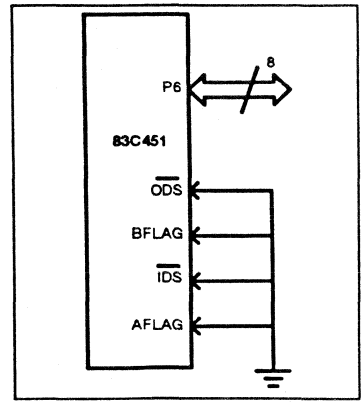


Figure 4. Standard I/O Port on Reset with AFLAG and BFLAG as Outputs

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
0	X	0	X	X	1		

Figure 5. CSR Programmed to Allow AFLAG and BFLAG to Operate as Outputs and Port 6 as a Standard I/O Port

IMPLEMENTATION OF PARALLEL PRINTER PORTS USING PORT 6

The 83C451 is an excellent choice for a printer controller. The 83C451 has the facilities to permit all of the intelligent features of a common printer to be handled by a single chip:

1. The features of port 6 allow a parallel printer port to be designed with only line driving and receiving chips required as additional hardware.
2. The onboard UART allows RS232 interfacing with only level shifting chips added.
3. The 8-bit parallel ports 0 to 6 are ample to drive onboard control functions, even when ports are used for external memory access, interrupts and other functions.
4. The RAM addressing ability of ports 0 and 2 can be used to address up to 64K bytes of a hardware buffer/spooler.
5. The 64K byte ROM addressing capability allows space for the most sophisticated software.

In addition, either end of a parallel interface can be implemented using port 6, and the interfaces can be interrupt driven or polled in either case.

THE INTERFACE

Data transfer on a parallel printer interface occurs across eleven signal lines. The other conductors on the standard plug are used as ground returns or for auxiliary functions (see figure 6). Only the data transfer signals will be considered.

The Data Transfer Format

The parallel printer interfaces are far more standardized in features than their serial counterpart. However at least three significant variations exist in handshake style in printers using generic parallel interfaces. This fact influences the design of both port hardware and software. A good transmitter should be able to drive devices with all three styles of handshakes and a good receiver should generate the handshake most likely compatible with any transmitter.

The Variations

Type 1 – Figure 7 shows a common style of handshake and is the style that will be implemented in the receiver examples. A busy signal and an acknowledge strobe pulse are generated for every byte received.

Type 2 – Another style of handshake generates a busy signal only when the printer will not be able to accept more data for a relatively long time. Acknowledge pulses are created after every byte received. When the busy signal is generated after a byte is received, the associated acknowledge pulse does not occur until **after** the busy signal returns to logic zero. (see figure 7).

Type 3 – A third handshake style does not generate acknowledge pulses, but a busy signal is produced after every byte is received.

DATA TRANSFER SIGNAL PINS		
Pin No.	Ground Return Pin No.	Signal
1	19	$\overline{\text{STROBE}}$
2	20	DATA 1
3	21	DATA 2
4	22	DATA 3
5	23	DATA 4
6	24	DATA 5
7	25	DATA 6
8	26	DATA 7
9	27	DATA 8
10	28	$\overline{\text{ACKNLG}}$
11	29	BUSY

TYPICAL AUXILIARY PIN FUNCTIONS	
Pin No.	Signal
12	PAPER OUT
14	$\overline{\text{AUTO LINE FEED}}$
16	LOGIC GROUND
17	CHASSIS GND
30	GROUND RETURN
31	$\overline{\text{RESET PRINTER}}$
32	$\overline{\text{ERROR}}$
33	GROUND RETURN
36	$\overline{\text{SLCT IN}}$

Figure 6. Parallel Printer Interface Pin Functions

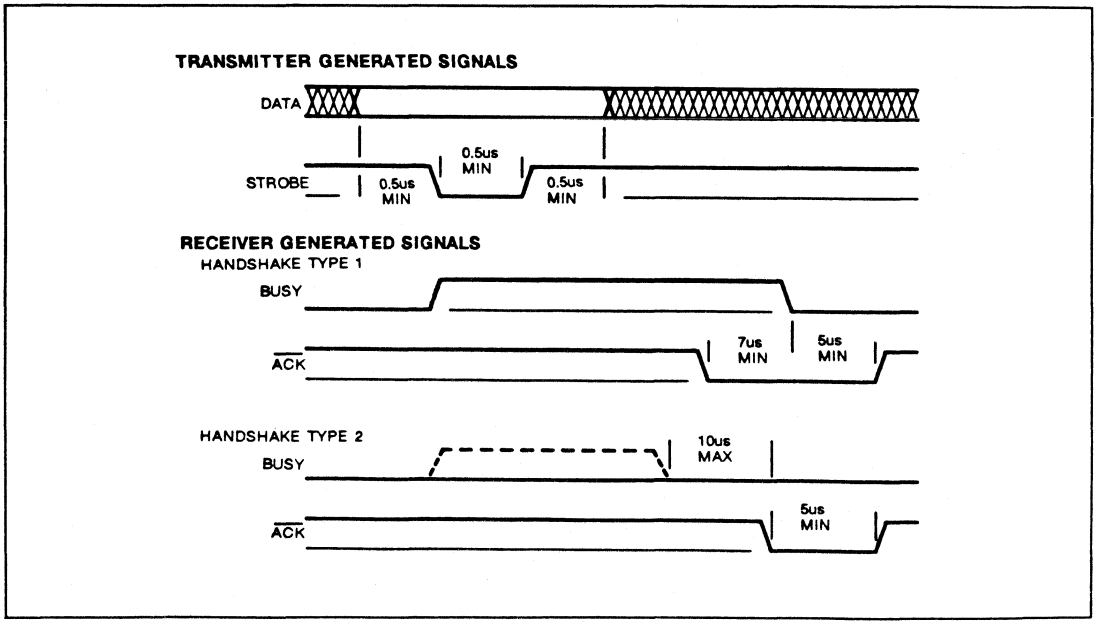


Figure 7. Parallel Printer Interface Signals

PARALLEL PRINTER INTERFACES USING POLLING

TRANSMITTER OPERATION

This application illustrates the flexibility of the port 6 logic in solving an applications problem. We need to be able to handle all types of acknowledge signals that might be received by the transmitter. We will use the ODS pin and output buffer full flag logic to record the receipt of the acknowledge pulse (see figure 8), but not all parallel receivers generate acknowledge pulses. We could poll the busy signal line, but not all receivers generate busy signals for each byte received; so lack of a busy signal does not imply that we can send another byte. We can, however, expect an acknowledge pulse very shortly after the end of a busy signal if one is going to arrive at all. So we can send a new data byte after having received either a positive transition on the acknowledge line, or shortly after receiving a negative edge on the busy line.

The CSR is programmed to the output only mode. In this mode the ODS pin does not control the output drivers but only the output buffer full flag. The flag serves to record the positive transition of the acknowledge signal. The input latch is not used, but the IDS pin is used to set the input buffer full flag. This is used to record the negative transition at the end of the busy signal. Dummy reads by the 83C451 of port 6 will be used to clear the flag. In this example, the AFLAG mode is set only to place the port in the output only mode. The AFLAG pin is not actually used (see figure 10).

The transmitter's CSR (control status register) is programmed to the following mode (see figure 9):

1. CSR bit 6 controls the BFLAG output and therefore the strobe line.
2. The OBF (output buffer full) flag controls the AFLAG* output.
3. The OBF is cleared on the positive edge of the ODS input.
4. The IBF flag is cleared on the negative edge of the IDS strobe.

Note:
With this combination of modes set, port 6 is in the output only mode.

RECEIVER OPERATION

In receiver operation, the IDS input is used to latch in the data transmitted on receipt of the strobe pulse. The receiver's CSR is programmed to allow the following:

1. The input buffer full flag is output through the BFLAG pin and is used as the busy signal to the transmitter.
2. The IBF flag is set and data is latched on the positive edge of IDS.
3. Writing to the CSR bit 4 controls the AFLAG output and therefore the acknowledge line.

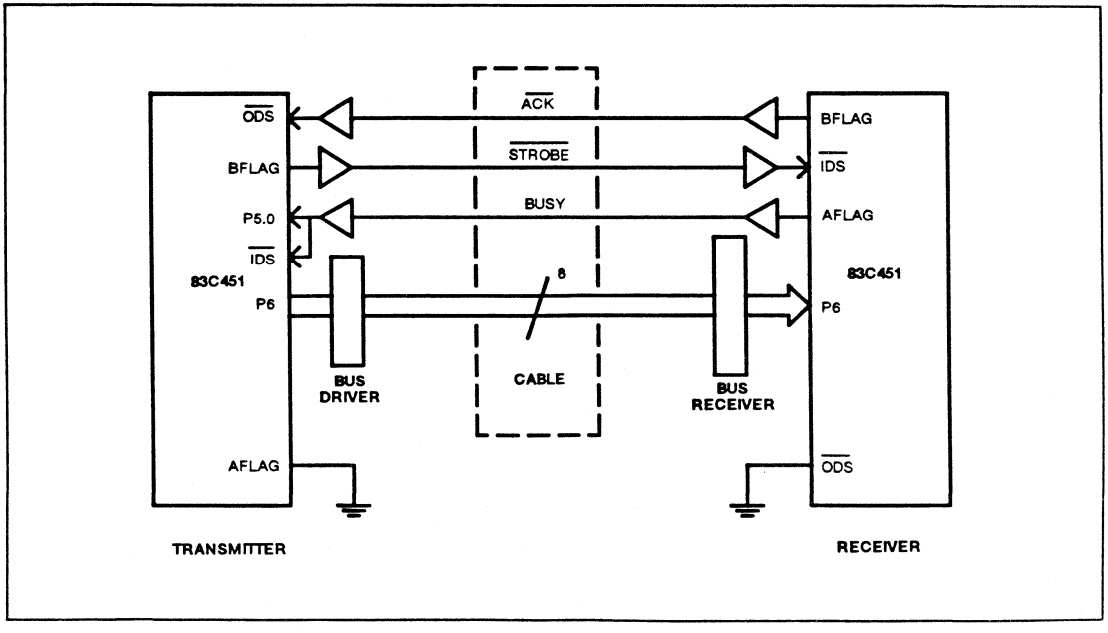


Figure 8. Interconnection for a Parallel Interface Using Polling

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
0	1	1	0	0	1		

Figure 9. CSR Programmed for Polled Transmitter Operation

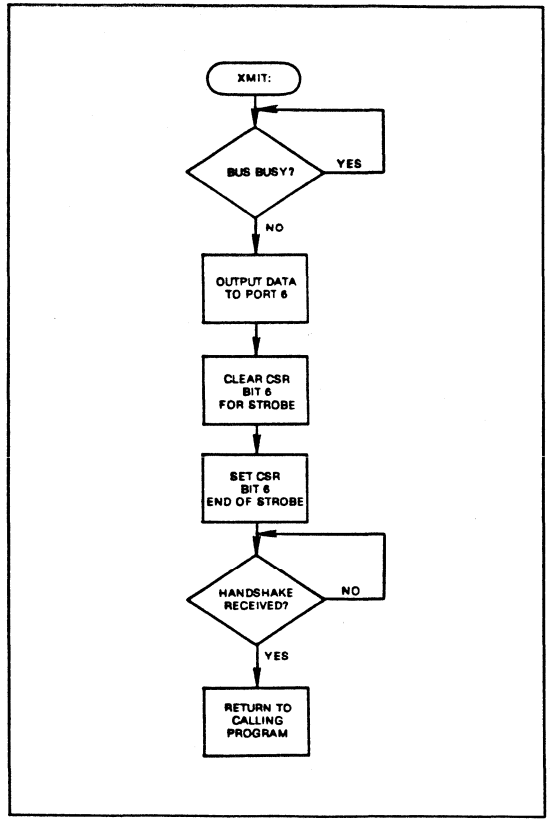


Figure 10. Flow Chart of Polled Parallel Transmitter Operation

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
1	0	0	1	1	1		

Figure 11. CSR Programmed for Polled Parallel Receiver Operation

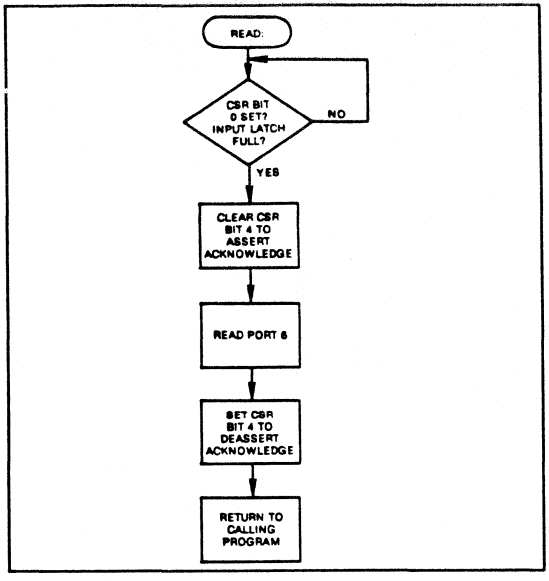


Figure 12. Flow Chart of Polled Parallel Receiver Operation

SOFTWARE EXAMPLES
This polled parallel transmit routine outputs one byte passed to it in the accumulator.

```
P_INIT:  MOV CSR,#064H          ;INITIALIZE PORT 6 OPERATING MODE
P_OUT:   JB P5.0                ;WAIT IF BUSY SIGNAL IS HIGH
         MOV P6,ACC             ;OUTPUT DATA
         MOV R1,P6              ;DUMMY READ TO CLEAR IBF FLAG
         MOV R1,#02H            ;INITIALIZE DELAY COUNTER
         CLEAR CSR.6            ;START STROBE PULSE
         DJNZ R1,$              ;TIME 6 MICROSECOND STROBE PULSE
         SETB CSR.6             ;END STROBE PULSE
WAIT:    JNB CSR.1,OUT          ;EXIT IF ACKNOWLEDGE RCV'D
         JNB CSR.0,WAIT         ;EXIT IF NEGATIVE BUSY EDGE RCV'D
         RET
```

This polled parallel receive routine places one byte in the accumulator each time it is called.

```
P_INIT:  MOV CSR,#09CH          ;INITIALIZE PORT 6 OPERATING MODE
         MOV R7,P6              ;DUMMY READ TO CLEAR IBF FLAG
P_IN:    JNB CSR.0              ;INPUT BUFFER LATCH FULL?
         CLR CSR.4              ;BEGIN ACKNOWLEDGE PULSE
         MOV R7,#02H            ;INITIALIZE DELAY COUNTER
         DJNZ R7,$              ;TIME ACKNOWLEDGE PULSE
         MOV A,P6               ;READ BYTE - CLEAR BUSY SIGNAL
         MOV R7,#02H            ;INITIALIZE DELAY COUNTER
         DJNZ R7,$              ;TIME ACKNOWLEDGE PULSE
         SETB CSR.4             ;END ACKNOWLEDGE PULSE
         RET
```

INTERRUPT DRIVEN PARALLEL PRINTER INTERFACE

TRANSMITTER OPERATION

The transmitter's CSR (control status register) is programmed to the following mode (see figure 14):

1. CSR bit 6 controls the BFLAG output and therefore the strobe line.
2. The OBF (output buffer full) flag controls the AFLAG* output.
3. The OBF is cleared on the positive edge of the ODS (output data strobe) input.
4. The IBF flag is set on the negative edge of the IDS (input data strobe) pin.

Note:
With this combination of AFLAG and BFLAG modes set, port 6 is in the output only mode. The output drivers are always enabled and the ODS input is only used to clear the OBF flag.
INT0 is programmed to be negative edge sensitive and is connected to the OBF flag through the AFLAG pin. The OBF is cleared on the positive edge of ODS. The net result is that INT0 is triggered on the end of the ACK pulse (a positive edge). This signals the transmitter that another byte may be transmitted. The transmitting 83C451 is free to do other tasks prior to this interrupt.

In this routine, figure 15, the main program establishes a buffer in data memory ended by an ASCII end of text character. To begin outputting the buffer the routine PSEND is called. The rest of the buffer is emptied by the interrupt vectors to PSEND1.

For printers which generate acknowledge pulses, output rates of 25K transfers per second are achieved. Timer generated interrupts are used to periodically return program execution to the routine to service non-acknowledging printers and to provide a timeout feature. Non acknowledging printers are serviced at a rate of about 2.5K transfers per second. This maximum rate may be varied by adjusting the timer reload value. As written, the time out procedure attempts to retransmit a byte when the printer has not acknowledged for an excessively long time.

RECEIVER OPERATION

In receiver operation, the IDS input is used to latch in the data transmitted on receipt of the strobe pulse. The receiver's CSR is programmed to allow the following (see figure 16):

1. The input buffer full flag is output through the BFLAG pin and is used as the busy signal to the transmitter. The IBF flag is set and data is latched on the positive edge of IDS.
2. Writing to the CSR bit 4 controls the AFLAG output and therefore the acknowledge line.

The receiver is interrupted on the negative edge of the data strobe. Data is latched in on the positive edge of the strobe pulse (see figure 17). Since the strobe pulse is normally very short there is little time lost between receiving the interrupt and having valid data in the input latch. The receiver is free to do other tasks prior to receiving the INT0 interrupt.

SOFTWARE EXAMPLES

The software for the interrupt driven parallel receiver is similar to the polled receiver example. However, after an interrupt is received, this routine checks to confirm that data has been latched by the positive edge of the strobe pulse before proceeding with the routine.

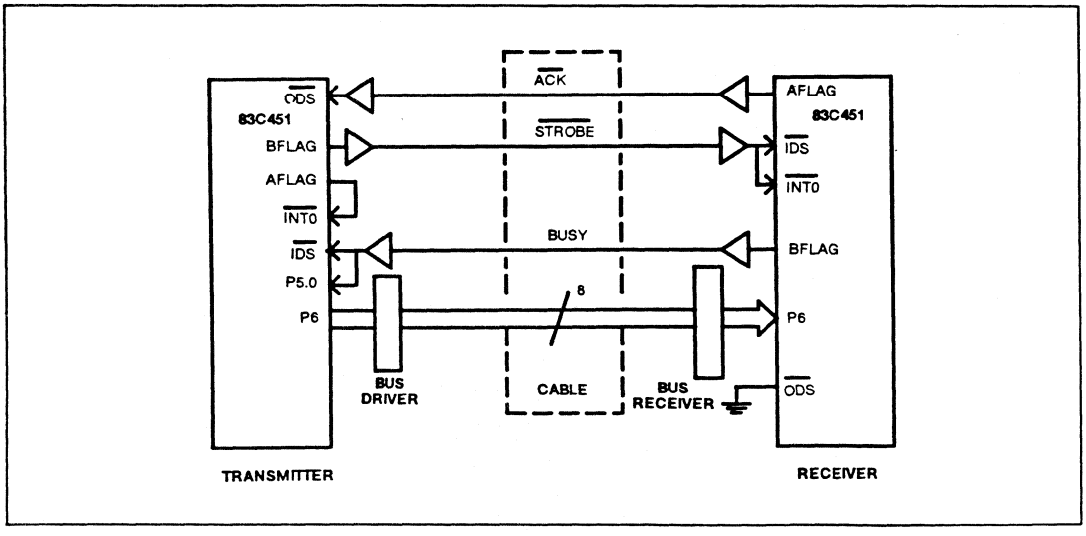


Figure 13. Interrupt Driven Parallel Interfaces Using 83C451 Controllers

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
0	1	1	0	0	1		

Figure 14. CSR Programming for Use as an Interrupt Driven Parallel Transmitter

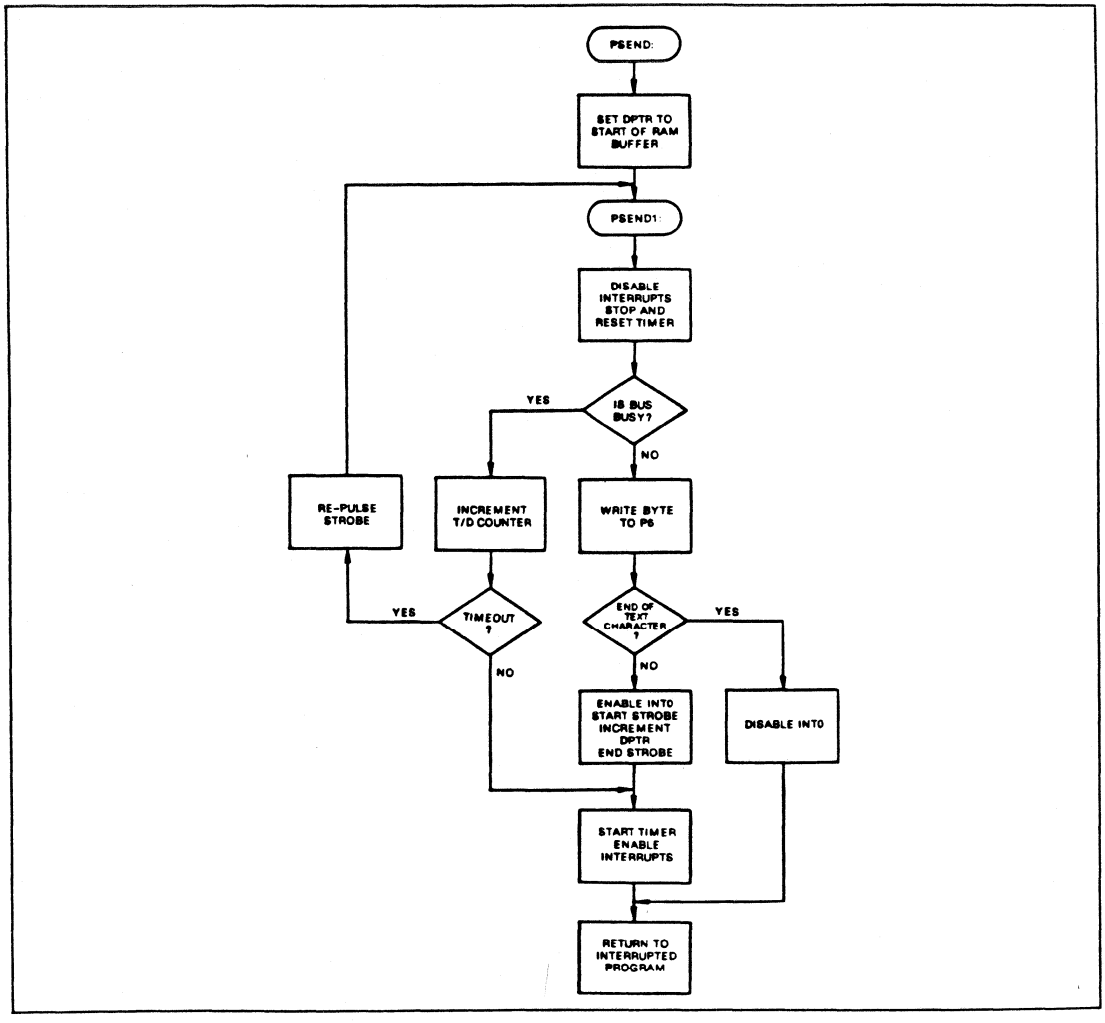


Figure 15. Flow Chart for an Interrupt Driven Parallel Transmitter

CSR 7	CSR 6	CSR 5	CSR 4	CSR 3	CSR 2	CSR 1	CSR 0
MB1	MB0	MA1	MA2	OBFC	IDSM	OBF	IBF
1	0	0	1	1	0		

Figure 16. CSR Programming for Use as an Interrupt Driven Parallel Receiver

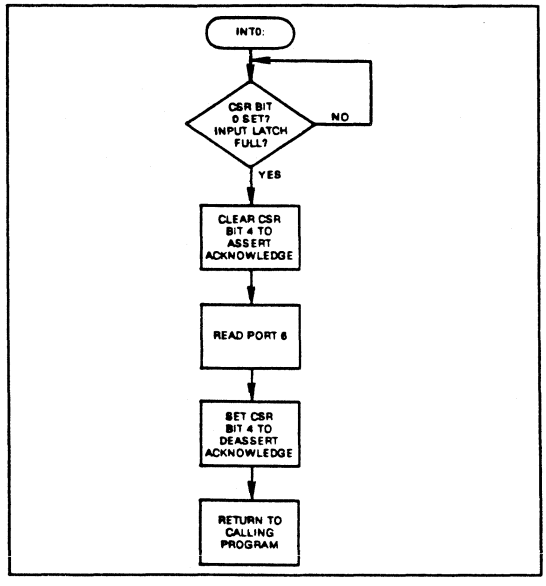


Figure 17. Flow Chart of Interrupt Driven Parallel Receiver Operation

SOFTWARE EXAMPLES

The software for the interrupt driven parallel receiver is similar to the polled receiver example. However, after an interrupt is received, this routine checks to confirm that data has been latched by the positive edge of the strobe pulse before proceeding with the routine.

```
INIT:    MOV CSR,#090H          ;INITIALIZE CSR
         SETB EX0               ;ENABLE INTERRUPT 0
         SETB IT0               ;SET NEG EDGE TRIGGERED INTERRUPTS
         SETB EA                ;ENABLE ALL INTERRUPTS

ORG EXTI0                       ;INTERRUPT 0 VECTOR
         JMP RCVR

RCVR: JNB CSR.0,$               ;CONFIRM DATA LATCHED
         CLR CSR.4              ;START ACKNOWLEDGE PULSE
         MOV R7,#02H            ;INITIALIZE THE DELAY COUNTER
         DJNZ R7,$              ;TIME ACK PULSE
         MOV A,P6               ;READ BYTE - RESET BUSY LINE
         MOV R7,#02H            ;INITIALIZE THE DELAY COUNTER
         DJNZ R7,$              ;TIME ACK PULSE
         SETB CSR.4             ;END ACK PULSE
         RETI
```

This is the software for the interrupt driven parallel transmitter example.

```
;  XMIT ROUTINE DRIVEN BY ACK PULSE GENERATED INTERRUPTS,OR TIMER GENERATED INTERRUPTS
;  FOR NON ACKNOWLEDGING PRINTERS. READS DATA BUFFER IN EXTERNAL RAM STARTING AT 100H
;  AND READING UNTIL 04H IS FOUND.

   ORG RESET
   JMP 26H
   ORG TIMER0
   JMP PSEND1
   ORG EXTI0
   JMP PSEND1
   ORG 26H
        MOV CSR, #064H          ;PORT 6 MODE
        MOV TMOD, #002H         ;CONFIGURE TIMER 0 TO 16 BITS
        SETB TO0                ;INT0 IS EDGE TRIGGERED
        SETB EA                 ;ENABLE INTERRUPTS
PSEND:  MOV DPTR, #0100H        ;SET DPTR TO START OF TEXT
                                ;BUFFER
PSEND1: CLR EA                  ;DISABLE INTERRUPTS AND STOP
                                ;TIMER
        CLR TR0                 ;IF ENABLED
        CLR ET0
        MOV R7, 00H             ;CLEAR TIMEOUT COUNTER
        MOV R6,00H
        MOV TH0, #-4            ;SET TIMER INTERRUPT PERIOD
        MOV TL0, #00H
        JB 0C8H,BB              ;BUS BUSY
        MOV ACC, #00H           ;CLEAR ACCUMULATOR
        MOVX A, @DPTR           ;RETRIEVE FIRST BYTE
        MOV P6, ACC             ;OUTPUT FIRST BYTE
        CJNE A, #004H, CONT1    ;LOOK FOR END OF TEXT
        JMP EOTB
CONT1:  SETB EX0                ;ENABLE INT0
        CLR 0EEH                ;START STROBE PULSE
        INC DPTR
        MOV ACC, DPH            ;LOOK FOR PHYSICAL END OF
        JB ACC. 2, EOTB         ;TEXT BUFFER
        SETB 0EEH               ;END STROBE PULSE
        JMP CONT
EOTB:   CLR EX0                 ;END OF TEXT FOUND, DISABLE
                                ;INT0
        SETB 0EEH
        SETB EA
        RETI
BB:     INC R7                  ;COUNT TIMER TIMEOUTS ON
                                ;BUS BUSY
        CJNE R7, #00H, CONT     ;LOOK FOR OVERFLOW
        INC R6                  ;COUNT OVERFLOWS
        CJNE R6, #10H, CONT     ;TIMEOUT APPROX 5 SEC
        JMP TO
CONT:   SETB TR0                ;ENABLE TIMER INTERRUPT
        SETB ET0                ;START TIMER
        SETB EA
        RETI
```

```
TO:     CLR 0C9H                ;SEND NEW STROBE PULSE IN
                                ;RESPONSE TO TIMEOUT
        NOP
        NOP
        MOV R6, #00H            ;RESET TO COUNTER
        MOV R7, #00H
        SETB 0C9H               ;END OF STROBE PULSE
        JMP PSEND1
```

3.2 The PCB80/83C552 microcontroller

CONTENTS – THE PCB80/83C552 MICROCONTROLLER

		page
1.0	DESCRIPTION	3.2-5
2.0	FEATURES	3.2-5
3.0	FUNCTIONAL DESCRIPTION	3.2-12
3.1	Memory organization	3.2-12
3.2	I/O facilities	3.2-15
3.2.1	Pulse width modulated outputs	3.2-17
3.2.2	Analogue input pins	3.2-18
3.3	Timer/event counters	3.2-20
3.3.1	Timer T2	3.2-20
3.3.2	Watchdog timer T3	3.2-25
3.4	Serial I/O	3.2-26
3.5	Idle and Power-down operation	3.2-30
3.6	Interrupt system	3.2-32
3.7	Oscillator circuitry	3.2-37
3.8	Reset circuitry	3.2-37
3.8.1	Power-on reset	3.2-39

1.0 DESCRIPTION

The PCB83C552 single-chip 8-bit microcontroller is manufactured in an advanced CMOS process and is a derivative of the PCB80C51 microcontroller family. The PCB83C552 has the same instruction set as the PCB80C51. Two versions of the derivative exist although the generic term "PCB83C552" is used to refer to both family members:

- PCB83C552: 8 K bytes mask-programmable ROM, 256 bytes RAM
- PCB80C552: ROM-less version of the PCB83C552

This I/O intensive device provides architectural enhancements to function as a controller in the field of automotive electronics, specifically engine management and gear box control.

The PCB83C552 contains a non-volatile 8 K x 8 read-only program memory, a volatile 256 x 8 read/write data memory, six 8-bit I/O ports, two 16-bit timer/event counters (identical to the timers of the 80C51), an additional 16-bit timer coupled to capture and compare latches, a fifteen-source, two-priority-level, nested interrupt structure, an 8-input ADC, a dual DAC pulse width modulated interface, two serial interfaces (UART and I^2C-bus), a 'watchdog' timer and on-chip oscillator and timing circuits. For systems that require extra capability, the PCB83C552 can be expanded using standard TTL compatible memories and logic.

The device also functions as an arithmetic processor having facilities for both binary and BCD arithmetic plus bit-handling capabilities. The instruction set consists of over 100 instructions: 49 one-byte, 45 two-byte and 17 three-byte. With a 12 MHz crystal, 58% of the instructions are executed in 1 µs and 40% in 2 µs. Multiply and divide instructions require 4 µs.

2.0 Features

- 80C51 Central processing unit
- 8 K x 8 ROM, expandable externally to 64 K bytes
- 256 x 8 RAM, expandable externally to 64 K bytes
- Two standard 16-bit timer/counters
- An additional 16-bit timer/counter coupled to four capture registers and three compare registers
- A 10-bit ADC with 8 multiplexed analogue inputs
- Two 8-bit resolution, Pulse Width Modulated outputs
- Five 8-bit I/O ports plus one 8-bit input port shared with analogue inputs
- I^2C-bus serial I/O port with byte orientated master and slave functions
- Full-duplex UART compatible with the standard PCB80C51
- On-chip watchdog timer

PACKAGE OUTLINES

PCB83C552: 68-lead PLCC, plastic, leaded-chip-carrier (SOT-188A).

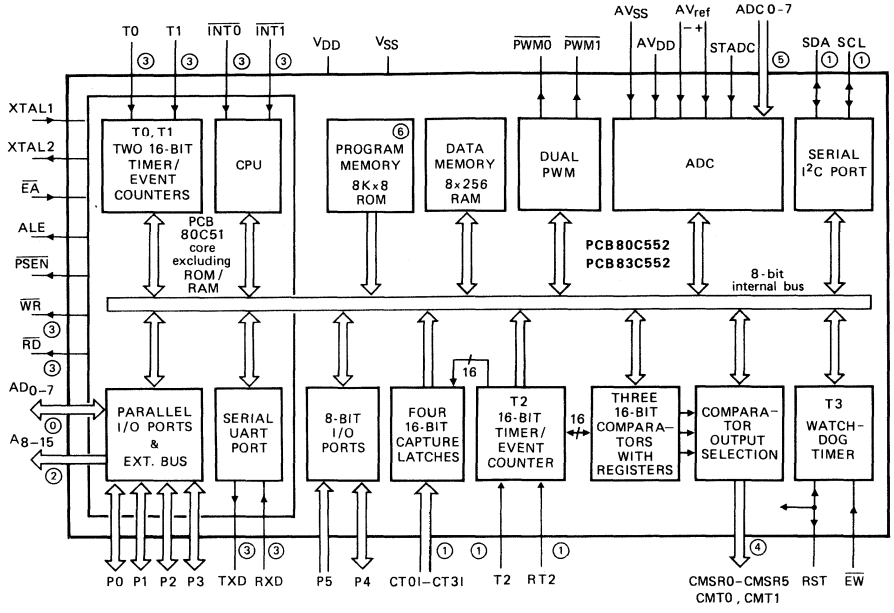


Fig. 1 Block diagram.

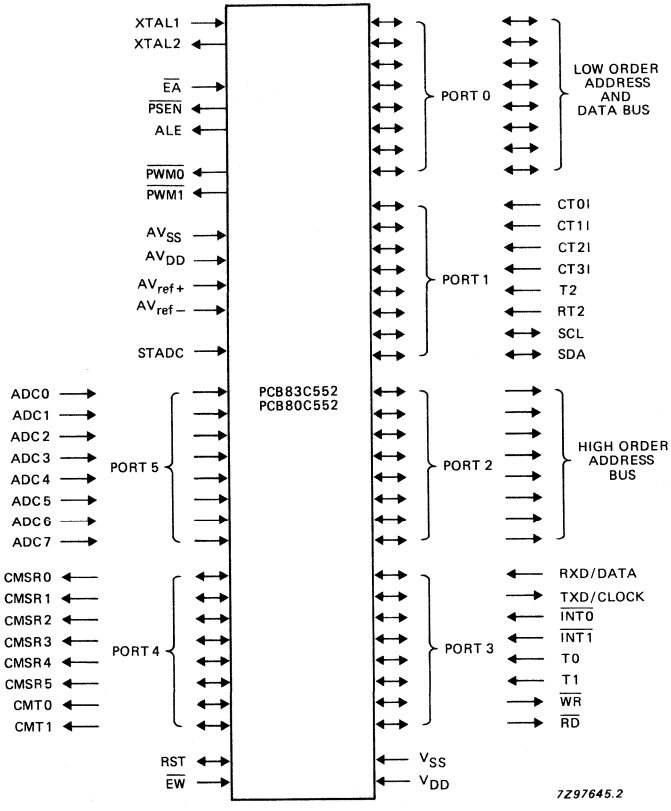


Fig. 2 Functional diagram.

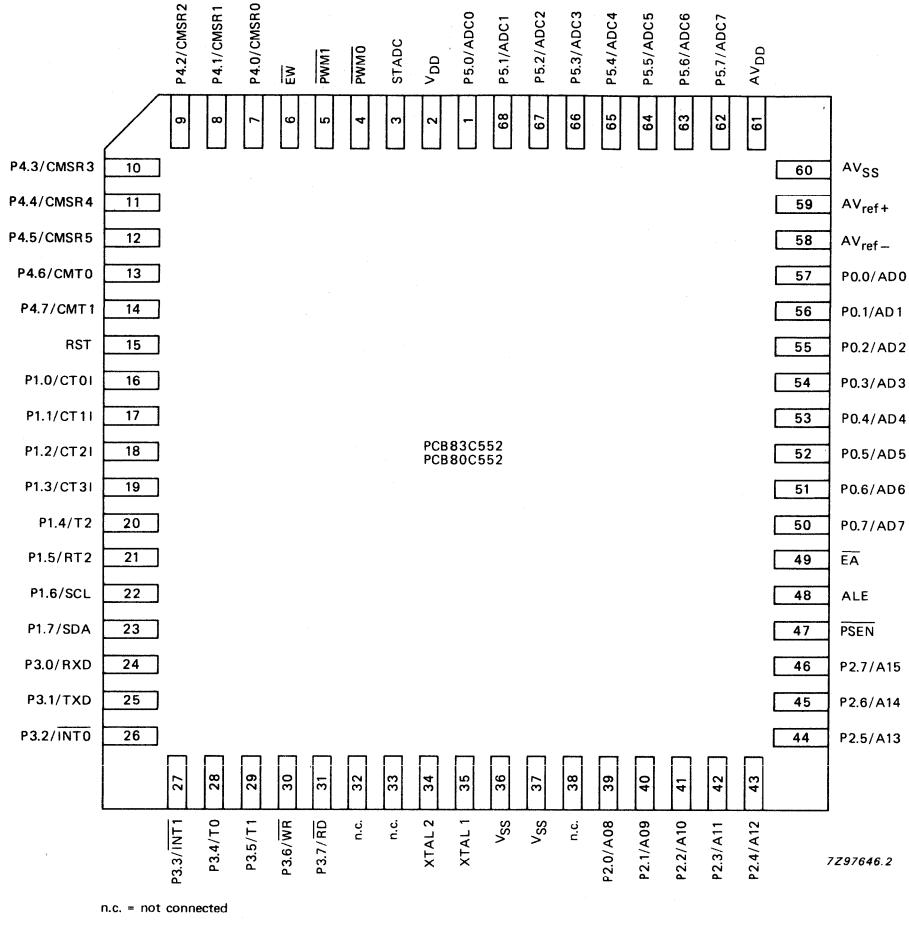


Fig. 3 Pinning diagram for PCB83C552.

2 V_{DD} — Digital power supply: + 5 V power supply pin during normal operation, Idle mode and Power-down mode

3 STADC — Start ADC operation: Input starting analogue to digital conversion (ADC operation can also be started by software)

4 $\overline{PWM0}$ — Pulse width modulation output 0

5 $\overline{PWM1}$ — Pulse width modulation output 1

6 $\overline{EW}$ — Enable watchdog timer: Enable for T3 watchdog timer and disable Power-down mode

Port 4:

7-14 P4.0-P4.7 — 8-Bit quasi-bidirectional I/O port

Port pin	Alternative function	
P4.0	CMSR0	Timer T2: compare and set/reset outputs on a match with timer T2
P4.1	CMSR1	
P4.2	CMSR2	
P4.3	CMSR3	
P4.4	CMSR4	
P4.5	CMSR5	
P4.6	CMT0	Timer T2: compare and toggle outputs on a match with timer T2
P4.7	CMT1	

15 RST — Reset: Input to reset the PCB83C552. It also provides a reset pulse as output when timer T3 overflows.

Port 1

16-23 P1.0-P1.7 — 8-Bit quasi-bidirectional I/O port

Port pin	Alternative function
P1.0	CT0I — Capture timer input signals for timer T2
P1.1	CT1I
P1.2	CT2I
P1.3	CT3I
P1.4	T2 : T2 event input
P1.5	RT2 : T2 timer reset signal. Rising edge triggered
P1.6	SCL : Serial port clock line I^2C-bus
P1.7	SDA : Serial port data line I^2C-bus

Port 3

24-31 P3.0-P3.7 — 8-Bit quasi-bidirectional I/O port

Port pin	Alternative function
P3.0	RXD : Serial input port
P3.1	TXD : Serial output port
P3.2	$\overline{INT0}$: External interrupt
P3.3	$\overline{INT1}$: External interrupt
P3.4	T0 : Timer 0 external input
P3.5	T1 : Timer 1 external input
P3.6	WR : External data memory write strobe
P3.7	RD : External data memory read strobe

32,33 Not connected

34 XTAL2 Crystal input 2: output of the inverting amplifier that forms the oscillator. Left open-circuit when an external oscillator is used.

35 XTAL1 Crystal input 1: input to the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator clock signal when an external oscillator is used.

36,37 V_{SS} Two digital ground pins

38 Not connected

Port 2

39- P2.0- 8-Bit quasi-bidirectional I/O port
46 P2.7

Port pin	Alternative function
P2.0-P2.7	High-order address byte for external memory (A08-A15)

47 $\overline{PSEN}$ Program store enable: active LOW read strobe to external program memory

48 ALE Address latch enable: latches the low byte of the address during accesses to external memory. It is activated every six oscillator periods. During an external data memory access one ALE pulse is skipped. ALE can drive up to 8 LSTTL inputs and handles CMOS inputs without an external pull-up.

49 $\overline{EA}$ External access: When $\overline{EA}$ is held at TTL level HIGH, the CPU executes out of the internal program ROM, provided the program counter is less than 8192. When $\overline{EA}$ is held at TTL LOW level, the CPU executes out of external program memory. $\overline{EA}$ is not allowed to float.

Port 0

50- P0.7- 8-Bit binary I/O port
57 P0.0

Port pin	Alternative function
P0.7-P0.0	Multiplexed low-order address and data bus of external memory (AD7 -AD0)

58 AV_{ref}- Low end of analogue to digital conversion reference resistor

59 AV_{ref}+ HIGH end of analogue to digital conversion reference resistor

60 AV_{SS} Analogue ground

61 AV_{DD} Analogue power supply

Port 5

62- P5.7- 8-Bit input port

68,1 P5.0

Port pin	Alternative Function
P5.0-P5.7	Eight input channels to ADC (ADC0-ADC7)

N.B. To avoid a 'latch-up' effect at power-on, the voltage on any pin at any time must not be higher or lower than V_{DD} + 0,5 V or V_{SS} - 0,5 V respectively.

3.0 FUNCTIONAL DESCRIPTION

Because the PCB83C552 has the same core CPU as the MAB8051, only the additional facilities will be described herein. For more exact details on the functional operation of the PCB83C552, see the preceding MAB8051 section. For the electrical specification of the PCB83C552 see the data sheet.

General

The PCB83C552 is a stand-alone high-performance microcontroller designed for use in real-time applications such as instrumentation, industrial control and specific automotive control applications.

The device provides in addition to the 80C51 standard functions, a number of dedicated hardware functions for these applications.

The PCB83C552 has two software selectable modes of reduced activity for further power reduction -Idle and Power-down. The Idle mode freezes the CPU while allowing the RAM, timers, serial ports and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator causing all other chip functions to be inoperative.

3.1 Memory organization

The central processing unit (CPU) manipulates operands in three memory spaces: these are the 64 K-byte external data memory, 256-byte internal data memory and the 64 K-byte internal and external program memory. The internal data memory address space is sub-divided into the 256-byte internal data RAM and 128-byte Special Function Register (SFR) address spaces, as shown in Fig. 4. Figures 5(a) and (b) show the Special Function Register memory map. Internal RAM locations 0-127 are directly and indirectly addressable. Internal RAM locations 128-255 are only indirectly addressable. The special function register locations 128 - 255 are only directly addressable.

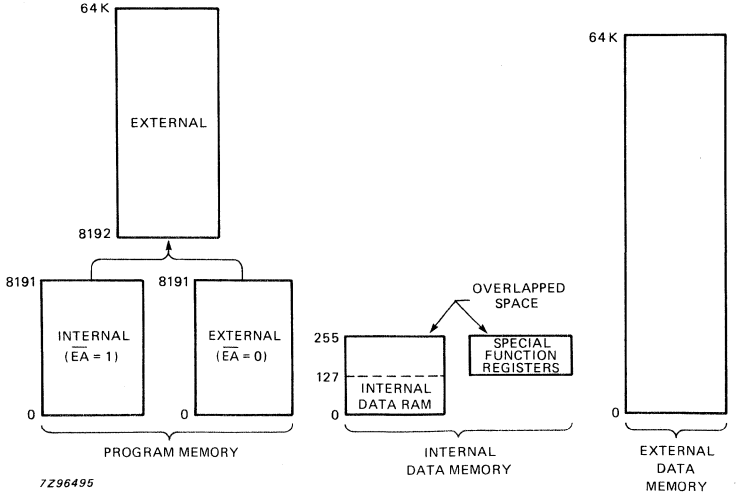
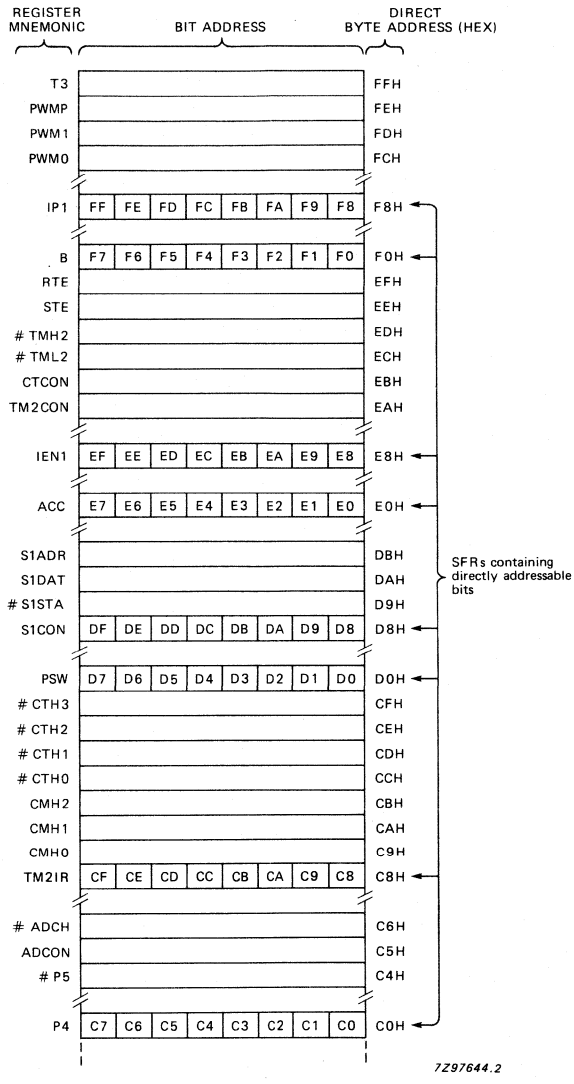
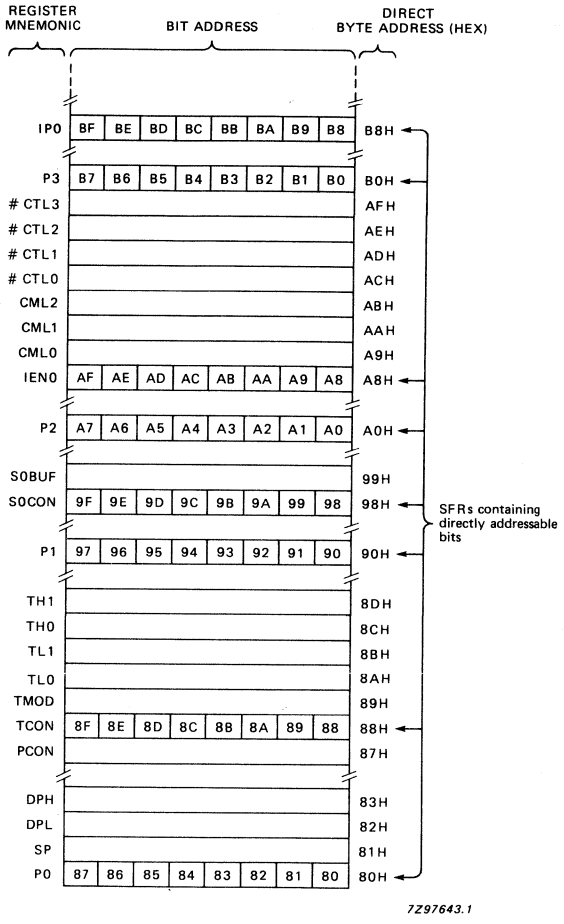


Fig. 4 Memory map.



# denotes read only registers

Fig. 5(a) Special Function Register memory map (continued in Fig. 5(b)).



# denotes read only registers

Fig. 5(b) Special Function Register memory map (continued from Fig. 5(a)).

3.2 I/O facilities

The PCB83C552 has six 8-bit ports. Ports 0 - 3 are the same as in the 80C51, with the exception of the additional functions of Port 1. The parallel I/O function of port 4 is equal to that of Ports 1, 2 and 3. Port 5 has a parallel input port function, but has no function as an output port. Port lines P1.7 and P1.6 may be selected as the SDA and SCL lines of serial port SIO1 (I^2C). Because the I^2C-bus may be active while the device is disconnected from V_{DD}, these pins are provided with open drain drivers. Pins P1.7 and P1.6 do not have pull-up devices when used as ports.

Ports 0, 1, 2, 3, 4, and 5 perform the following alternative functions:

- Port 0: provides the multiplexed low-order address and data bus used for expanding the PCB83C552 with standard memories and peripherals.

- Port 1: Port 1 is used for a number of special functions;

  - 4 capture inputs
  - external counter input
  - external counter reset input
  - SCL and SDA for the I^2C interface

Bits whose alternate function is not used may be used as normal bidirectional I/O pins.

- Port 2: provides the high-order address bus when expanding the PCB83C552 with external program memory and/or external data memory.

- Port 3: pins can be configured individually to provide:

  - external interrupt request inputs
  - counter inputs
  - serial port receiver input and transmitter output
  - control signals to READ and WRITE external data memory

The generation or use of a Port 3 pin as an alternative function is carried out automatically by the PCB83C552 provided the associated Special Function Register bit is set high.

- Port 4: can be configured to provide signals indicating a match between timer counter T2 and its compare registers.

- Port 5: may be used in conjunction with the ADC interface. Unused analogue inputs can be used as digital inputs. As port 5 lines may be used as inputs to the ADC, these digital inputs have an inherent hysteresis to prevent the input logic from drawing too much current from the power lines when driven by analogue signals. Channel to channel crosstalk should be considered when both digital and analogue signals are simultaneously input to Port 5 (see D.C. characteristics).

All ports are bidirectional with the exception of Port 5 which is an input port

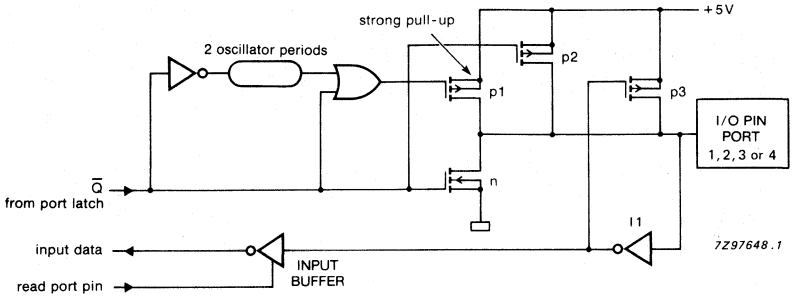


Fig. 6 I/O buffers in the PCB83C552 (Ports 2, 3, 4 and P1.0 to P1.5).

In addition to the standard 8-bit ports, the I/O facilities of the PCB83C552 also include a number of special I/O lines:

3.2.1 Pulse width modulated outputs

Two pulse width modulated output channels are provided with the PCB83C552. These channels output pulses of programmable length and interval. The repetition frequency is defined by an 8-bit prescaler PWMP which generates the clock for the counter. Both the prescaler and counter are common to both PWM channels. The 8-bit counter counts modulo 255 i.e. from 0 to 254 inclusive. The value of the 8-bit counter is compared to the contents of two registers: PWM0 and PWM1. Provided the contents of either of these registers is greater than the counter value, the output of $\overline{PWM0}$ or $\overline{PWM1}$ is set LOW. If the contents of these registers are equal to, or less than the counter value, the output will be HIGH. The pulse-width-ratio is therefore defined by the contents of the registers PWM0 and PWM1. The pulse-width-ratio is in the range of 0 to 255/255and may be programmed in increments of 1/255.

The repetition frequency f_{PWM}, at the PWMnN outputs is given by:

$$f_{PWM} = \frac{f_{OSC}}{2x(1 + PWMP)x\ 255}$$

This gives a repetition frequency range of 92 Hz to 23,5 kHz (f_{osc} = 12 MHz).

By loading the PWM registers with either 00H or FFH, the PWM outputs can be retained at a constant HIGH or LOW level respectively. When loading FFH to the PWM registers, the 8-bit counter will never actually reach this value.

Both PWMn output pins are driven by push-pull drivers, and are not shared with any other function.

Prescaler frequency control register PWMP

	7	6	5	4	3	2	1	0
PWMP(FEH)	MSB							LSB

Bit	Function
PWMP.0-7	Prescaler division factor = (PWMP) + 1

Pulse width registers PWM0 and PWM1

	7	6	5	4	3	2	1	0
PWM0(FCH) PWM1(FDH)	MSB							LSB

BIT	Function
PWM0.0-7 PWM1.0-7	Low/high ratio of PWMn signals = $\frac{(PWMn)}{255 - (PWMn)}$

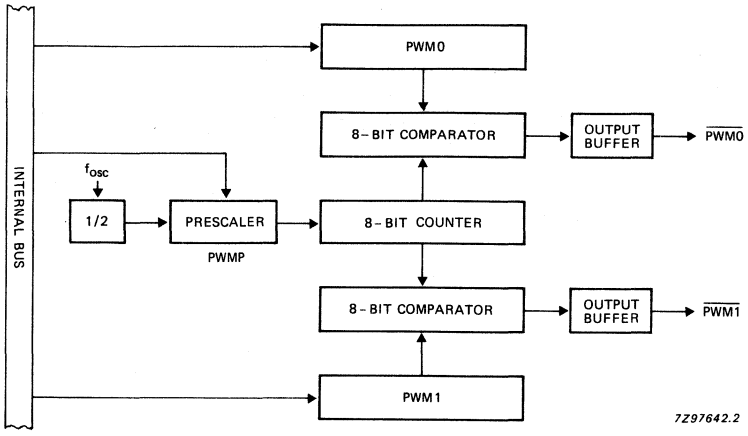


Fig. 7 Functional diagram of pulse width modulated outputs.

3.2.2 Analogue input pins

The analogue input circuitry consists of an 8-input analogue multiplexer and an ADC with 10 bit resolution. The analogue reference voltage and analogue power supplies are connected via separate input pins. The conversion takes 50 machine cycles i.e. 50 μs at 12 MHz oscillator frequency.

The ADC is controlled using the ADCON control register. Input channels are selected by the analogue multiplexer, care of ADCON register bits 0-2.

ADC control register ADCON

	7	6	5	4	3	2	1	0
ADCON (C5H)	ADC.1	ADC.0	ADEX	ADCI	ADCS	AADR2	AADR1	AADR0

Bit	Symbol	Function
ADCON.7	ADC.1	Bit 1 of ADC converted value
ADCON.6	ADC.0	Bit 0 of ADC converted value
ADCON.5	ADEX	Enable external start of conversion by STADC 0 = Conversion cannot be started externally by STADC 1 = Conversion can be started externally by STADC
ADCON.4	ADCI	ADC interrupt flag: This flag is set when an A/D conversion result is ready to be read. An interrupt is invoked if this is enabled. The flag must be cleared by software. It cannot be set by software.
ADCON.3	ADCS	ADC start and status: Setting this bit starts an ADC conversion. It may be set by software or by the external

signal STADC. The ADC logic ensures that this signal is HIGH while the ADC is busy. On completion of the conversion, ADCS is reset at the same time the interrupt flag ADCI is set. ADCS can not be reset by software.

ADCI	ADCS	OPERATION
0	0	ADC not busy, a conversion can be started
0	1	ADC busy, start of a new conversion is blocked
1	0	Conversion completed, start of a new conversion is blocked
1	1	Not possible

ADCON.2
ADCON.1
ADCON.0

Analogue input select: this binary coded address selects one of the eight analogue port bits of P5 to be input to the converter. It can only be changed when ADCI and ADCS are both LOW. AADR2 is the most significant bit (100 selects the ADC4 analogue input channel).

The completion of the 10 bit ADC conversion is flagged by ADCI in the ADCON register and the result is stored in special function register ADCH (upper 8 bits) and the 2 LSBs in register ADCON.

An ADC conversion in progress is unaffected by an external or software ADC start. The result of a completed conversion remains unaffected provided ADCI = logic 1. While ADCS = logic 1 or ADCI = logic 1, a new ADC START will be blocked and consequently lost. An ADC conversion already in progress is aborted when the Idle or Power-down mode is entered. The result of a completed conversion (ADCI = 1) remains unaffected when entering the Idle mode.

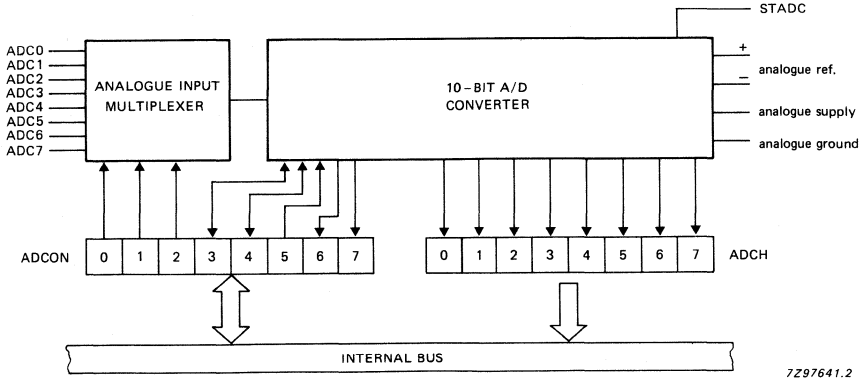


Fig. 8 Functional diagram of analogue input.

3.3 Timer/event counters

The PCB83C552 contains three 16-bit timer/event counters: Timer 0, Timer 1 and Timer T2 and one 8-bit timer, T3. Timer 0 and Timer 1 may be programmed to carry out the following functions:

- Measure time intervals and pulse durations
- Count events
- Generate interrupt requests

Timer 0 and Timer 1 can be programmed independently to operate in three modes:

- Mode 0: 8-bit timer or 8-bit counter each with divide-by-32 prescaler
- Mode 1: 16-bit time-interval or event counter
- Mode 2: 8-bit time-interval or event counter with automatic reload upon overflow

Timer 0 can be programmed to operate in an additional mode as follows:

- Mode 3: one 8-bit time-interval or event counter and one 8-bit time-interval counter

When Timer 0 is in Mode 3, Timer 1 can be programmed to operate in Modes 0, 1 or 2 but cannot set an interrupt request flag or generate an interrupt. However the overflow from Timer 1 can be used to pulse the serial port transmission-rate generator.

The frequency handling range of these counters with a 12 MHz crystal is as follows:

- In the timer function, the timer is incremented at a frequency of 1 MHz - a division by 12 of the oscillator frequency
- 0 Hz to an upper limit of 0,5 MHz when programmed for external inputs

Both internal and external inputs can be gated to the counter by a second external source for directly measuring pulse durations.

The counters are started and stopped under software control. Each one sets its interrupt request flag when it overflows from all logic 1s to all logic 0s (or automatic reload value), with the exception of mode 3 as previously described.

3.3.1 Timer T2

Timer T2 is a 16 bit timer/counter which has, coupled to it, capture and compare facilities. The operational diagram is shown in Fig. 9.

The 16 bit timer/counter is clocked via a prescaler with a programmable division factor of 1, 2, 4 or 8. The input of the prescaler is clocked with 1/12 of the oscillator frequency, or with positive edges on the T2 input or it is switched to the off position. This prescaler is cleared if its division factor or its input source is changed, or if the timer/counter is reset. T2 is readable 'on the fly', but posseses no extra read latches; this means that software precautions have to be taken against misinterpretation in overflow from least to most significant byte during read. T2 is not loadable and is reset by the RST signal or at the positive edge of the input signal RT2, if enabled. In the Idle mode the timer/counter and prescaler are reset and halted.

T2 is connected to four 16-bit capture registers: CT0, CT1, CT2 and CT3. These registers are loaded with the contents of T2 and an interrupt is requested upon receipt of the input signals CT0I, CT1I, CT2I or CT3I. These input signals are shared with port 1. Using the capture register CTCON, these inputs may capture on a positive edge, a negative edge or on either a positive or negative edge.

The contents of the compare registers CM0, CM1 and CM2 are continually compared with the counter value of Timer T2. When a match is found an interrupt may be invoked. Using the match signal of CM0, the controller sets bits 0-5 of Port 4, if the corresponding bits of the set enable register STE are logic 1.

Considering a match with CM1, if the corresponding bits of the reset/toggle enable register RTE are logic 1, then the controller will use the match signal to reset bits 0-5 of port 4. Bits 6 and 7 of port 4 may be 'toggled' by the signal that indicates a match between Timer T2 and CM2 if the corresponding bits of RTE are logic 1. CM0, CM1 and CM2 are reset by the RST signal.

Port 4 can be read and written by software without affecting the toggle, set and reset signals. At byte overflow of the least significant byte, or at a 16-bit overflow of the timer/counter, an interrupt sharing the same interrupt vector is requested. Either or both of these overflows can be programmed to request an interrupt.

All interrupt flags must be reset by software.

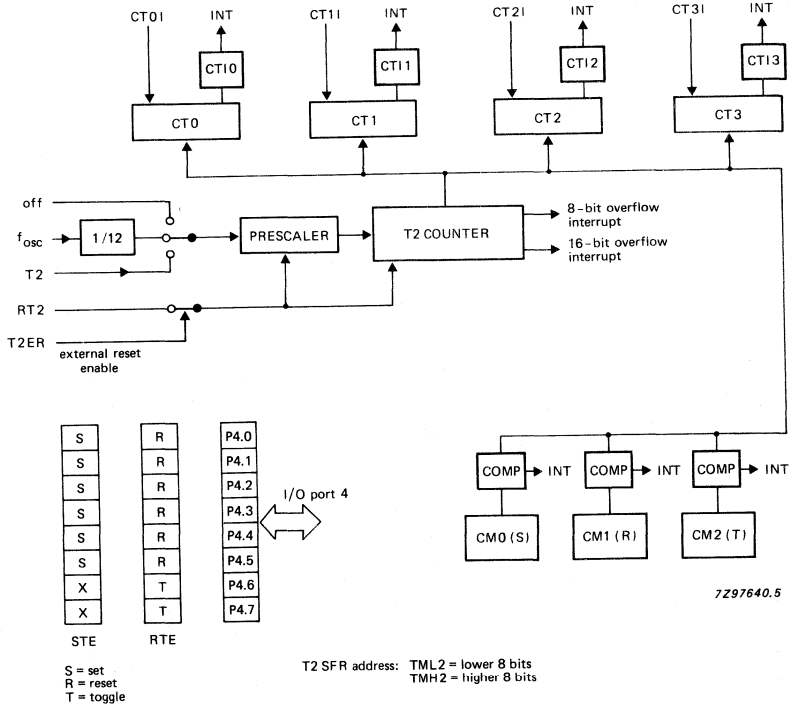


Fig. 9 Block diagram of Timer T2 configuration.

Counter control register TM2CON

	7	6	5	4	3	2	1	0
TM2CON (EAH)	T2IS1	T2IS0	T2ER	T2B0	T2P1	T2P0	T2MS1	T2MS0
	MSB							LSB

Bit	Symbol	Function
TM2CON.7	T2IS1	Enable 16 bit overflow interrupt
TM2CON.6	T2IS0	Enable byte overflow interrupt
TM2CON.5	T2ER	Timer 2 external reset enable
TM2CON.4	T2B0	Timer 2 byte overflow interrupt flag
TM2CON.3	T2P1	Timer 2 prescaler select
TM2CON.2	T2P0	

T2P1	T2P0	T2 clock
0	0	clock source
0	1	clock source / 2
1	0	clock source / 4
1	1	clock source / 8

Bit	Symbol	Function
TM2CON.1	T3MS0	Timer 2 mode select
TM2CON.0	T2MS1	

T2MS1	T2MS0	Mode Selected
0	0	Timer T2 is halted
0	1	T2 clock source = $f_{osc}/12$
1	0	Test mode; do not use
1	1	T2 clock source = pin T2

Capture control register CTCON

	7	6	5	4	3	2	1	0
CTCON (EBH)	CTN3	CTP3	CTN2	CTP2	CTN1	CTP1	CTN0	CTP0
	MSB							LSB

Bit	Symbol	Capture/Interrupt on:
CTCON.7	CTN3	CT3I negative edge
CTCON.6	CTP3	CT3I positive edge
CTCON.5	CTN2	CT2I negative edge
CTCON.4	CTP2	CT2I positive edge
CTCON.3	CTN1	CT1I negative edge
CTCON.2	CTP1	CT1I positive edge
CTCON.1	CTN0	CT0I negative edge
CTCON.0	CTP0	CT0I positive edge

Timer interrupt flag register TM2IR

	7	6	5	4	3	2	1	0
TM2IR (C8H)	T2OV	CMI2	CMI1	CMI0	CTI3	CTI2	CTI1	CTI0
	MSB							LSB

Bit	Symbol	Function
TM2IR.7	T2OV	T2 16-bit overflow interrupt flag
TM2IR.6	CMI2	CM2 interrupt flag
TM2IR.5	CMI1	CM1 interrupt flag
TM2IR.4	CMI0	CM0 interrupt flag
TM2IR.3	CTI3	CT3 interrupt flag
TM2IR.2	CTI2	CT2 interrupt flag
TM2IR.1	CTI1	CT1 interrupt flag
TM2IR.0	CTI0	CT0 interrupt flag

Interrupt enable register IEN1 is used to enable/disable Timer 2 interrupts. Interrupt priority register IP1 is used to determine the timer interrupt priority.

The set enable register STE

STE	7	6	5	4	3	2	1	0
STE (EEH)	-	-	SP45	SP44	SP43	SP42	SP41	SP40
	MSB							LSB

Bit	Symbol	Function
STE.7	-	not used
STE.6	-	not used
STE.5	SP45	if 1 then P4.5 is set on a match between CM0 and T2
STE.4	SP44	if 1 then P4.4 is set on a match between CM0 and T2
STE.3	SP43	if 1 then P4.3 is set on a match between CM0 and T2
STE.2	SP42	if 1 then P4.2 is set on a match between CM0 and T2
STE.1	SP41	if 1 then P4.1 is set on a match between CM0 and T2
STE.0	SP40	if 1 then P4.0 is set on a match between CM0 and T2

If STE.n is "0" then P4.n is not affected by a match between CM0 and T2

The reset/toggle enable register RTE

	7	6	5	4	3	2	1	0
RTE (EFH)	TP47	TP46	RP45	RP44	RP43	RP42	RP41	RP40
	MSB							LSB

Bit	Symbol	Function
RTE.7	TP47	if 1 then P4.7 toggles on a match between CM2 and T2
RTE.6	TP46	if 1 then P4.6 toggles on a match between CM2 and T2
RTE.5	RP45	if 1 then P4.5 is reset on a match between CM1 and T2
RTE.4	RP44	if 1 then P4.4 is reset on a match between CM1 and T2
RTE.3	RP43	if 1 then P4.3 is reset on a match between CM1 and T2
RTE.2	RP42	if 1 then P4.2 is reset on a match between CM1 and T2
RTE.1	RP41	if 1 then P4.1 is reset on a match between CM1 and T2
RTE.0	RP40	if 1 then P4.0 is reset on a match between CM1 and T2

If RTE.n is "0" then P4.n is not affected by a match between CM1 and T2 or CM2 and T2

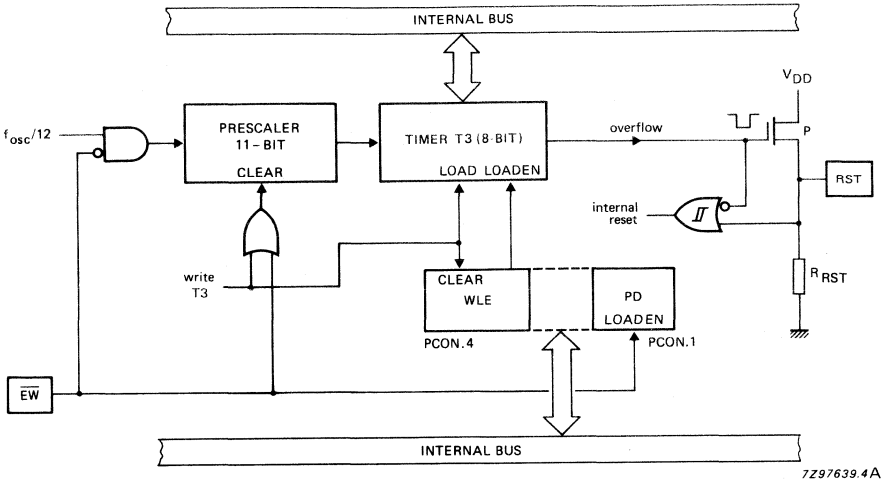


Fig. 10. Functional diagram of T3 watchdog timer.

3.3.2 T3 The watchdog timer (see Fig. 10)

In addition to Timer T2 and the standard timers, a watchdog timer consisting of an 11-bit prescaler and an 8-bit timer are also incorporated.

The timer is incremented every 2 ms, derived from the oscillator frequency of 12 MHz by the following:

$$f_{timer} = \frac{f_{osc}}{12 \times 2048}$$

When a timer overflow occurs, the microcontroller is reset and a reset output pulse is generated at pin RST. To prevent a system reset the timer must be reloaded in time by the application software. If the processor suffers a hardware/software malfunction, the software will fail to reload the timer. This failure to reload the timer will produce a reset upon overflow thus preventing the processor running out of control.

The watchdog timer can only be reloaded if the condition flag WLE = PCON.4 has been previously set by software.

At the moment the counter is loaded the condition flag is automatically cleared.

The time interval between the timer's reloading and occurence of a reset, is dependent upon the reloaded value. This may range from 2 ms to 0,5 s at an oscillator frequency of 12 MHz.

In the Idle state the watchdog timer and reset circuitry remain active.

The watchdog timer is controlled by the watchdog enable pin ($\overline{EW}$). A logic 0 enables the watchdog timer and disables the Power-down mode. A logic 1 disables and resets the watchdog timer and enables the Power-down mode.

3.4 Serial I/O (see Fig. 11)

The PCB83C552 is equipped with two independent serial ports. SIO0 is the full duplex UART port and is identical to the serial port of the PCB80C51 (see PCB80C51 data sheet).

Serial port SIO1 supports the I^2C-bus, the function of which is controlled by the S1CON register. S1STA is the status register whose contents may also be used as a vector to various service routines. S1DAT is the data shift register and S1ADR the slave address register. The least significant bit of S1ADR enables/disables general call address recognition.

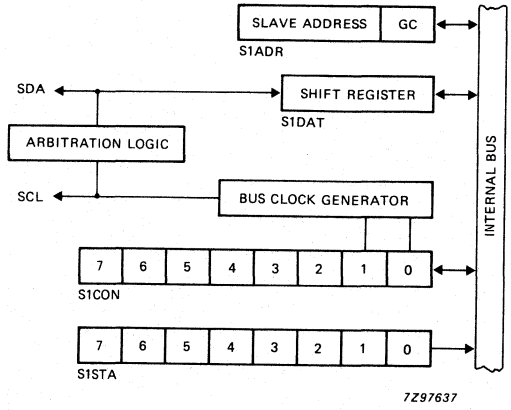


Fig. 11 Block diagram of I^2C serial I/O.

The I^2C serial I/O has complete autonomy in byte handling and operates in 4 modes:

1. Master transmitter
2. Master receiver
3. Slave transmitter
4. Slave receiver

Slave address recognition is performed by on-chip hardware.

The I^2C-bus consists of two lines: a data line (SDA) and a clock line (SCL). These lines also function as the I/O port lines on P1.7 and P1.6. The system is unique because data transport, clock generation, address recognition and bus control arbitration are all controlled by hardware.

Serial control register S1CON

	7	6	5	4	3	2	1	0
S1CON (D8H)	X	ENS1	STA	STO	SI	AA	CR1	CR0

Bits CR1 and CR0 determine the clock frequency that is generated in the master mode of operation. Table 1 displays the clock rate when using a 12 MHz crystal.

Table 1 Clock rate when using a 12 MHz crystal

CR1/CR0	bit frequency	f_{osc} divided by
0 0	12,5 kHz	960
0 1	100 kHz	120
1 0	200 kHz	60 ($f_{osc} < 6$ MHz meeting I^2C)
1 1	62,5 - 0,5 kHz	96 x (256 – reload value Timer 1) (reload value range: 0-254 in mode 2)

AA

Assert acknowledge bit. When this bit is set, an acknowledge is returned after any one of the following conditions:

- Own slave address is received
- General call address is received (S1ADR.0 = logic 1)
- A data byte is received, while the device is programmed to be a master receiver
- A data byte is received, while the device is a selected slave receiver

When this bit is reset, no acknowledge is returned. Consequently, no interrupt is requested when the own address or general call address is received.

SI

SIO1 interrupt flag. This flag is set, and an interrupt request is generated, after any of the following events occur:

- A START condition is generated in MST mode
- The own slave address has been received during AA = logic 1
- The general call address has been received while S1ADR.0 and AA = logic 1
- A data byte has been received or transmitted in MST mode (even if arbitration is lost)
- A data byte has been received or transmitted as selected slave
- A STOP or START condition is received as selected slave receiver or transmitter

STO

STOP flag. When in master mode, and this bit is set a STOP condition is generated. A STOP condition detected on the I^2C-bus clears this bit. This bit may also be set in slave mode in order to recover from an error condition. Then no STOP condition is generated to the I^2C-bus, but the hardware releases the SDA and SCL lines and switches to the not selected slave receiver mode. The STOP flag is cleared by the hardware.

STA

START flag. When this bit is set in slave mode, the hardware checks the I^2C-bus and generates a START condition if the bus is free or after the bus becomes free. If the device is operating in master mode it will generate a repeated START condition.

ENS1

0 = Serial I/O Disabled and reset. P1.6 and P1.7 I/O port function with open drain

1 = Serial I/O Enabled. Output Ports P1.6 and P1.7 must be set to logic 1

Serial Status Register S1STA (S1STA is a read-only register)

	7	6	5	4	3	2	1	0
S1STA (D9H)	SC4	SC3	SC2	SC1	SC0	0	0	0

S1STA.3 - S1STA.7 hold a status code. S1STA.0 - S1STA.2 are held LOW. The contents of the status register may be used as a vector to a service routine. This optimizes the response time of the software and consequently that of the I^2C-bus.

Abbreviations used:

SLA	:	7-bit slave address
R	:	Read bit
W	:	Write bit
ACK	:	Acknowledgement (acknowledge bit = logic 0)
$\overline{ACK}$	:	Not acknowledgement (acknowledge bit = logic 1)
DATA	:	8-bit data byte to or from I^2C-bus
MST	:	Master
SLV	:	Slave
TRX	:	Transmitter
REC	:	Receiver

The following is a list of the status codes:

MST/TRX mode

S1STA value

- 08H - A START condition has been transmitted
- 10H - A repeated START condition has been transmitted
- 18H - SLA and W have been transmitted, ACK has been received
- 20H - SLA and W have been transmited, $\overline{ACK}$ received
- 28H - DATA of S1DAT has been transmitted, ACK received
- 30H - DATA of S1DAT has been transmitted, $\overline{ACK}$ received
- 38H - Arbitration lost in SLA, R/W or DATA

MST/REC mode

S1STA value

- 38H - Arbitration lost while returning $\overline{ACK}$
- 40H - SLA and R have been transmitted, ACK received
- 48H - SLA and R have been transmitted, $\overline{ACK}$ received
- 50H - DATA has been received, ACK returned
- 58H - DATA has been received, $\overline{ACK}$ returned

SLV/REC mode

S1STA value

60H - Own SLA and W have been received, ACK returned
68H - Arbitration lost in SLA, R/W as MST. Own SLA and W have been received, ACK returned
70H - General CALL has been received, ACK returned
78H - Arbitration lost in SLA, R/W as MST. General call has been received
80H - Previously addressed with own SLA. DATA byte received, ACK returned
88H - Previously addressed with own SLA. DATA byte received $\overline{\text{ACK}}$ returned
90H - Previously addressed with general call. DATA byte has been received ACK has been returned
98H - Previously addressed with general call. DATA byte has been received, $\overline{\text{ACK}}$ has been returned
A0H - A STOP condition or repeated START condition has been received while still addressed as SLV/REC or SLV/TRX

SLV/TRX mode

S1STA value

A8H - Own SLA and R have been received, ACK returned
B0H - Arbitration lost in SLA, R/W as MST. Own SLA and R have been received, ACK returned
B8H - DATA byte has been transmitted, ACK received
C0H - DATA byte has been transmitted, $\overline{\text{ACK}}$ received
C8H - Last DATA byte has been transmitted (AA = logic 0), ACK received.

Miscellaneous

S1STA value

00H - Bus error during MST mode or selected SLV mode, due to an erroneous START or STOP condition

The data shift register S1DAT

S1DAT (DAH)	7	6	5	4	3	2	1	0

This register contains the serial data to be transmitted or data which has just been received. Bit 7 is transmitted or received first; i.e. data is shifted from right to left.

Address register S1ADR

	slave address							GC
S1ADR (DBH)	7	6	5	4	3	2	1	0

S1ADR.0, GC : 0 = general call address is not recognised
1 = general call address is recognised.

S1ADR.7-1 : own slave address

This 8-bit register may be loaded with the 7-bit slave address to which the controller will respond when programmed as a slave receiver/transmitter. The LSB (GC) is used to determine whether the general call address is recognized.

3.5 Idle and Power-down operation (see Fig. 12).

Idle mode operation permits the interrupt, serial ports and timer blocks to continue to function while the CPU is halted. The following functions are also switched off when the processor enters the IDLE mode.

Timer T2	(stopped and reset)
PWM0, PWM1	(reset, output = HIGH)
ADC	(aborted if in progress)

The following functions remain active during Idle mode. These functions may generate an interrupt or reset and thus end the Idle mode.

Timer 0, Timer 1
Timer T3
SIO0, SIO1
External interrupt

The Power-down operation freezes the oscillator. The Power-down mode can only be activated by setting the PD bit in the PCON register. The PD bit can only be set if the EWN input is HIGH.

Idle mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM and all other registers maintain their data during Idle mode. The status of the external pins during Idle mode is shown in Table 2.

There are two ways to terminate the Idle mode:

Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware terminating Idle mode. The interrupt is serviced, and following return from interrupt instruction RETI, the next instruction to be executed will be the one which follows the instruction that wrote a logic 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an interrupt, the service routine can examine the status of the flags bits.
The second way of terminating the Idle mode is with an external hardware reset, or an internal reset caused by an overflow of Timer T3. Since the oscillator is still running, the hardware reset is required to be active for two machine cycles (24 oscillator periods) to complete the reset operation.

Power-down mode

The instruction that sets PCON.1 is the last executed prior to going into the Power-down mode. Once in Power-down mode, the oscillator is stopped. Only the contents of the on-chip RAM are preserved. The Special Function Registers are not saved. A hardware reset is the only way of exiting the Power-down mode.

In the Power-down mode, V_{DD} may be reduced to minimize circuit power consumption. The voltage must not be reduced until the Power-down mode is entered, but must be restored before the hardware reset is applied which will free the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

The status of the external pins during Power-down mode is shown in Table 2. If the Power-down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a logic 1, the port pin is held HIGH during the Power-down mode by the strong pull-up transistor p1 (see Fig. 6).

Table 2 Status of the external pins during Idle and Power-down modes

mode	memory	ALE	$\overline{PSEN}$	Port 0	Port 1	Port 2	Port 3	Port 4	$\overline{PWM0}/\overline{PWM1}$
Idle (1)	internal	1	1	port data	port data	port data	port data	port data	HIGH
Idle (1)	external	1	1	floating	port data	address	port data	port data	HIGH
Power-down	internal	0	0	port data	port data	port data	port data	port data	HIGH
Power-down	external	0	0	floating	port data	port data	port data	port data	HIGH

Power control register (PCON)

These special modes are activated by software via the Special Function Register PCON. Its hardware address is 87H. PCON is not bit addressable.

	7	6	5	4	3	2	1	0
PCON (87H)	SMOD	-	-	WLE	GF1	GF0	PD	IDL

Bit	Symbol	Function
PCON.7	SMOD	Double Baud rate bit. When set to logic 1 the baud rate is doubled when the serial port SIOO is being used in modes 1, 2 or 3.
PCON.6	-	(reserved)
PCON.5	-	(reserved)
PCON.4	WLE	Watchdog load enable. This flag must be set by software prior to loading T3 (watchdog timer). It is cleared when T3 is loaded.
PCON.3	GF1	General-purpose flag bit
PCON.2	GF0	General-purpose flag bit
PCON.1	PD	Power-down bit. Setting this bit activates Power-down mode. It can only be set if input EW is high.
PCON.0	IDL	Idle mode bit. Setting this bit activates the Idle mode.

If logic 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XX00000).

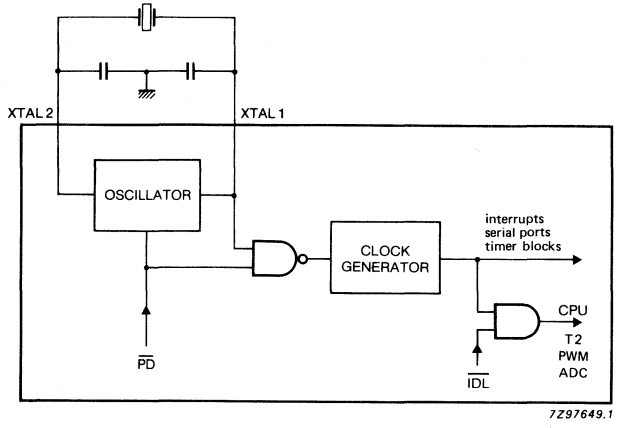


Fig. 12 Internal Idle and Power-down clock configuration.

3.6 Interrupt system (see Fig. 13)

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution a multiple-source, two-priority-level, nested interrupt system is provided. Interrupt response latency is from 3 μs to 7 μs when using a 12 MHz crystal. The PCB83C552 acknowledges interrupt requests from fifteen sources as follows:

- $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$: externally via pins 26 and 27 respectively
- Timer 0 and Timer 1: from the two internal counters
- Timer T2 (8 separate interrupts): 4 capture interrupts, 3 compare interrupts and an overflow interrupt
- ADC end-of-conversion interrupt
- I^2C serial I/O port interrupt
- UART serial I/O port interrupt.

Each interrupt vectors to a separate location in program memory for its service program. Each source can be individually enabled or disabled by a corresponding bit in the IEN0 or IEN1 register, moreover each interrupt may be programmed to a high or low priority level using a corresponding bit in the IP0 or IP1 register. Also all enabled sources can be globally disabled or enabled. Both external interrupts can be programmed to be level-activated or transition-activated; an active LOW level allows "wire-ORing" of several interrupt sources to one input pin.

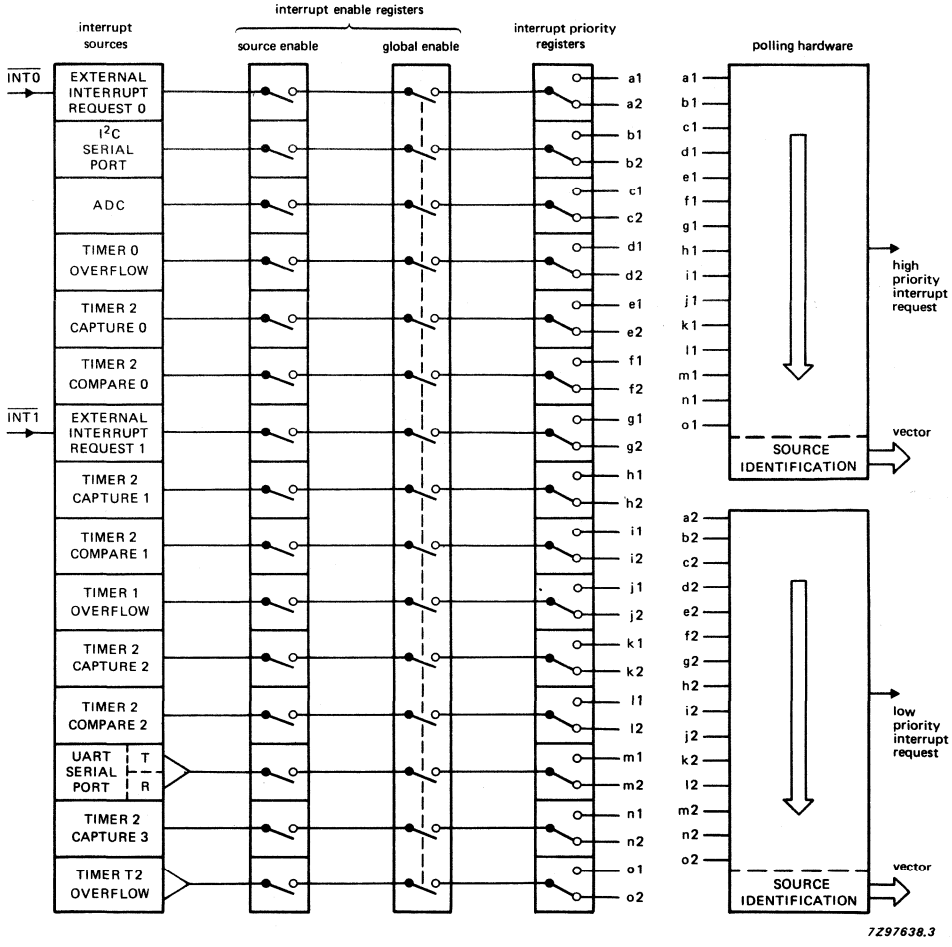


Fig. 13 Interrupt system.

Interrupt enable registers

	7	6	5	4	3	2	1	0
IEN0 (A8H)	EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0

Bit	Symbol	Function
IEN0.7	EA	General enable/disable control. 0 = No interrupt is enabled 1 = Any individually enabled interrupt will be accepted
IEN0.6	EAD	Enable ADC interrupt
IEN0.5	ES1	Enable SIO1 (I^2C) interrupt
IEN0.4	ES	Enable SIO0 (UART) interrupt
IEN0.3	ET1	Enable Timer 1 interrupt
IEN0.2	EX1	Enable External 1 interrupt
IEN0.1	ET0	Enable Timer 0 interrupt
IEN0.0	EX0	Enable External 0 interrupt

	7	6	5	4	3	2	1	0
IEN1 (E8H)	ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0

Bit	Symbol	Function
IEN1.7	ET2	Enable T2 overflow interrupt(s)
IEN1.6	ECM2	Enable T2 comparator 2 interrupt
IEN1.5	ECM1	Enable T2 comparator 1 interrupt
IEN1.4	ECM0	Enable T2 comparator 0 interrupt
IEN1.3	ECT3	Enable T2 capture register 3 interrupt
IEN1.2	ECT2	Enable T2 capture register 2 interrupt
IEN1.1	ECT1	Enable T2 capture register 1 interrupt
IEN1.0	ECT0	Enable T2 capture register 0 interrupt

where 0 = interrupt disabled
and 1 = interrupt enabled

Interrupt priority registers

	7	6	5	4	3	2	1	0
IP0 (B8H)	-	PAD	PS1	PS0	PT1	PX1	PT0	PX0

Bit	Symbol	Function
IP.7	-	Unused
IP.6	PAD	ADC interrupt priority level
IP.5	PS1	SIO1 (I^2C) interrupt priority level
IP.4	PS	SIO0 (UART) interrupt priority level
IP.3	PT1	Timer 1 interrupt priority level
IP.2	PX1	External interrupt 1 priority level
IP.1	PT0	Timer 0 interrupt priority level
IP.0	PX0	External interrupt 0 priority level

	7	6	5	4	3	2	1	0
IP1 (F8H)	PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0

Bit	Symbol	Function
IP1.7	PT2	T2 overflow interrupt(s) priority level
IP1.6	PCM2	T2 comparator 2 priority interrupt level
IP1.5	PCM1	T2 comparator 1 priority interrupt level
IP1.4	PCM0	T2 comparator 0 priority interrupt level
IP1.3	PCT3	T2 capture register 3 priority interrupt level
IP1.2	PCT2	T2 capture register 2 priority interrupt level
IP1.1	PCT1	T2 capture register 1 priority interrupt level
IP1.0	PCT0	T2 capture register 0 priority interrupt level

Interrupt priority levels are as follows : 0 - low priority
1 - high priority

Table 3 shows the interrupt vectors. The vector indicates the ROM location where the appropriate interrupt service routine starts.

Table 3 Interrupt vectors

Source		Vector
External 0	X0	0003H
Timer 0 overflow	T0	000BH
External 1	X1	0013H
Timer 1 overflow	T1	001BH
Serial I/O 0 (UART)	S0	0023H
Serial I/O 1 (I^2C)	S1	002BH
T2 capture 0	CT0	0033H
T2 capture 1	CT1	003BH
T2 capture 2	CT2	0043H
T2 capture 3	CT3	004BH
ADC completion	ADC	0053H
T2 compare 0	CM0	005BH
T2 compare 1	CM1	0063H
T2 compare 2	CM2	006BH
T2 overflow	T2	0073H

Interrupt priority

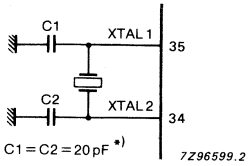
Each interrupt source can be either high priority or low priority. If both levels are requested simultaneously, the processor will branch to the high priority vector. If there are simultaneous requests from sources of the same priority, then interrupts will be serviced in the following order:

X0, S1, ADC, T0, CT0, CM0, X1, CT1, CM1, T1, CT2, CM2, S0, CT3, T2

A low priority interrupt can only be interrupted by high priority interrupt. A high priority interrupt can not be interrupted.

3.7 Oscillator circuitry

The oscillator circuitry of the PCB83C552 is a single-stage inverting amplifier in a Pierce oscillator configuration. The circuitry between XTAL 1 and XTAL 2 is basically an inverter biased to the transfer point. Either a crystal or ceramic resonator can be used as the feedback element to complete the oscillator circuitry. Both are operated in parallel resonance. XTAL 1 (pin 35) is the high gain amplfier input, and XTAL 2 (pin 34) is the output (see Fig. 14). To drive the PCB83C552 externally, XTAL 1 is driven from an external source and XTAL 2 left open-circuit (see Fig. 15).



*) preliminary value

Fig. 14 PCB83C552 oscillator circuit.

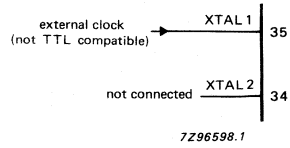


Fig. 15 Driving the PCB83C552 from an external source.

3.8 Reset circuitry (see Fig. 16)

The reset circuitry for the PCB83C552 is connected to the reset pin RST. A Schmitt trigger is used at the input for noise rejection. The output of the Schmitt trigger is sampled by the reset ciruitry every machine cycle.

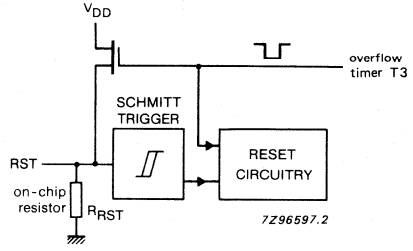


Fig. 16 On-chip reset configuration.

A reset is accomplished by holding the RST pin HIGH for at least two machine cycles (24 oscillator periods). The CPU responds by executing an internal reset. During reset ALE and $\overline{PSEN}$ output a HIGH level. In order to perform a correct reset, this level must not be affected by external elements.

Also with the PCB83C552, the RST line can be pulled HIGH internally by a pull-up transistor activated by the watchdog timer T3. The length of the output pulse from T3 is 3 machine cycles. A pulse of such short duration is necessary in order to recover from a processor or system fault as fast as possible.

N.B. It can be seen, that the short reset pulse from Timer T3 cannot discharge the power-on reset capacitor (see Fig. 17). Consequently, when the watchdog timer is also used to reset external devices this capacitor arrangement should not be connected to the RST pin, and an extra circuit should be used to perform the power-on reset operation. It should be remembered that a Timer T3 overflow, if enabled, will force a reset condition to the PCB83C552 by an internal connection, whether the output RST is tied LOW or not.

The internal reset is executed during the second cycle in which RST is HIGH and is repeated every cycle until RST goes LOW. It leaves the internal registers as follows:

Register	Content
ACC	0000 0000
ADCON	XX00 0000
ADCH	XXXX XXXX
B	0000 0000
CML0 - CML2	0000 0000
CMH0 - CMH2	0000 0000
CTCON	0000 0000
CTL0 - CTL3	XXXX XXXX
CTH0 - CTH3	XXXX XXXX
DPL	0000 0000
DPH	0000 0000
IEN0	0000 0000
IEN1	0000 0000
IP0	X000 0000
IP1	0000 0000
PCH	0000 0000
PCL	0000 0000
PCON	0XX0 0000
PSW	0000 0000
PWM0	0000 0000
PWM1	0000 0000
PWMP	0000 0000
P0 - P4	1111 1111
P5	XXXX XXXX
RTE	0000 0000
S0BUF	XXXX XXXX
S0CON	0000 0000
S1ADR	0000 0000
S1CON	X000 0000
S1DAT	0000 0000
S1STA	1111 1000
SP	0000 0111
STE	XX00 0000
TCON	0000 0000
TH0, TH1	0000 0000
TMH2	0000 0000
TL0, TL1	0000 0000
TML2	0000 0000
TMOD	0000 0000
TM2CON	0000 0000
TM2IR	0000 0000
T3	0000 0000

The internal RAM is not affected by reset. When V_{DD} is turned on, the RAM content is indeterminate.

3.8.1 Power-on reset (see Fig. 17)

When V_{DD} is turned on, and provided its rise-time does not exceed 10 ms, an automatic reset can be obtained by connecting the RST pin to V_{DD} via a 2,2 µF capacitor. When the power is switched on, the voltage on the RST pin is equal to V_{DD} minus the capacitor voltage, and decreases from V_{DD} as the capacitor charges through the internal resistor (R_{RST}) to ground. The larger the capacitor, the more slowly V_{RST} decreases. V_{RST} must remain above the lower threshold of the Schmitt trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

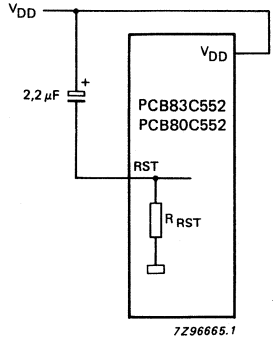


Fig. 17 Power-on reset.

3.3 The PCB80/83C652 microcontroller

CONTENTS – THE PCB83C652 MICROCONTROLLER

		page
1.0	DESCRIPTION	3.3-5
2.0	FEATURES	3.3-5
3.0	PACKAGE OUTLINES	3.3-5
	3.1. Pin Designation	3.3-8
4.0	FUNCTIONAL DESCRIPTION	3.3-11
	4.1 Memory organization	3.3-11
	4.2 I/O facilities	3.3-13
	4.3 Idle and Power-down operation	3.3-14
	4.3.1 Power control register (PCON)	3.3-15
	4.3.2 Idle mode	3.3-16
	4.3.3 Power-down mode	3.3-17
	4.4 Interrupt system	3.3-17
	4.4.1 Interrupt enable register	3.3-19
	4.4.2 Interrupt priority register	3.3-19
	4.4.3 Interrupt vectors	3.3-20
	4.4.4 Interrupt priority	3.3-20
	4.5 Oscillator circuitry	3.3-21
	4.6 Reset circuitry	3.3-21
	4.6.1 Power-on reset	3.3-23

1.0 DESCRIPTION

The PCB83C652 single-chip 8-bit microcontroller is manufactured in an advanced CMOS process and is a derivative of the 80C51 microcontroller family. PCB83C652 has the same instruction set as the 80C51. Two versions of the derivative exist although the generic term "PCB83C652" is used to refer to both family members:

- PCB80C652: ROM-less version of the PCB83C652
- PCB83C652: 8 K bytes mask-programmable ROM, 256 bytes RAM

This device provides architectural enhancements that make it applicable in a variety of applications including general control systems.

The PCB83C652 contains a non-volatile 8 K x 8 read-only program memory (not ROM-less version), a volatile 256 x 8 read/write data memory; four 8-bit I/O ports; two 16-bit timer/event counters (identical to the timers of the 80C51), a multi-source, two-priority-level, nested interrupt structure; an I^2C interface and a UART, and on-chip oscillator and timing circuits. For systems that require extra capability, the PCB83C652 can be expanded using standard TTL compatible memories and logic.

The device also functions as an arithmetic processor having facilities for both binary and BCD arithmetic plus bit-handling capabilities. The instruction set consists of over 100 instructions; 49 one-byte, 45 two-byte and 17 three-byte. With a 12 MHz crystal, 58% of the instructions are executed in 1 µs and 40% in 2 µs. Multiply and divide instructions require 4 µs.

2.0 FEATURES

- 80C51 Central processing unit
- 8 K x 8 ROM, expandable externally to 64 K bytes
- 256 x 8 RAM, expandable externally to 64 K bytes
- Two standard 16-bit timer/counters
- Four 8-bit I/O ports
- I^2C-bus serial I/O port with byte orientated master and slave functions
- Full-duplex UART facilities
- In preparation: extended temperature range, frequency range; 1,2 - 12 MHz

3.0 PACKAGE OUTLINE

PCB83C652P;PCB80C652P: 40-lead DIL; plastic (SOT-129).
PCB83C652;PCB80C652WP: 44-lead plastic leaded-chip-carrier (PLCC); (SOT-187A).

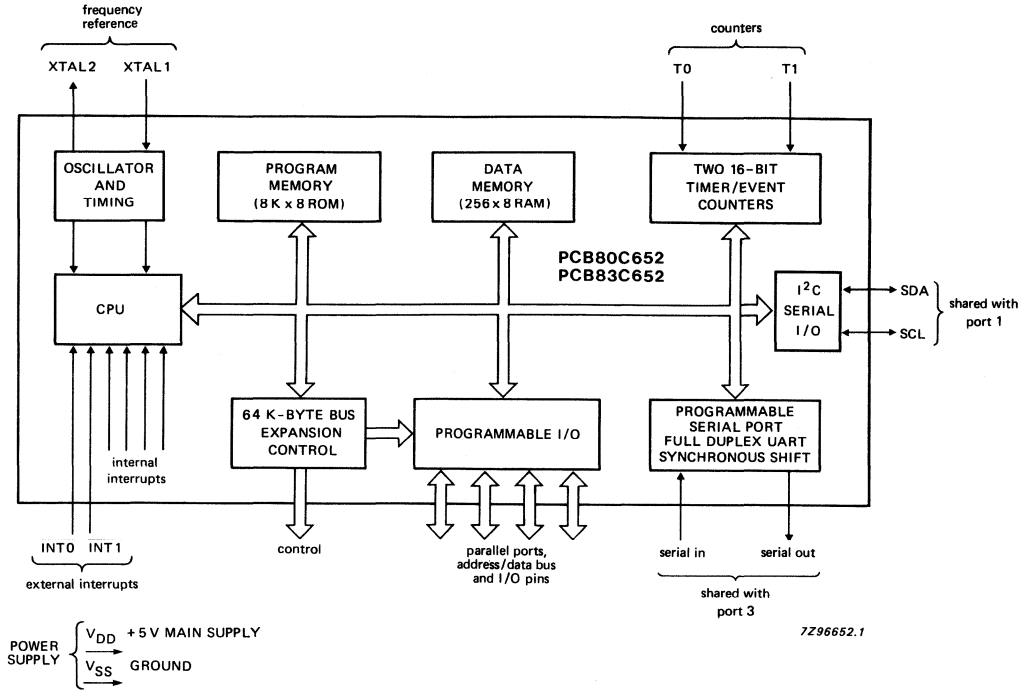


Fig. 1 Block diagram.

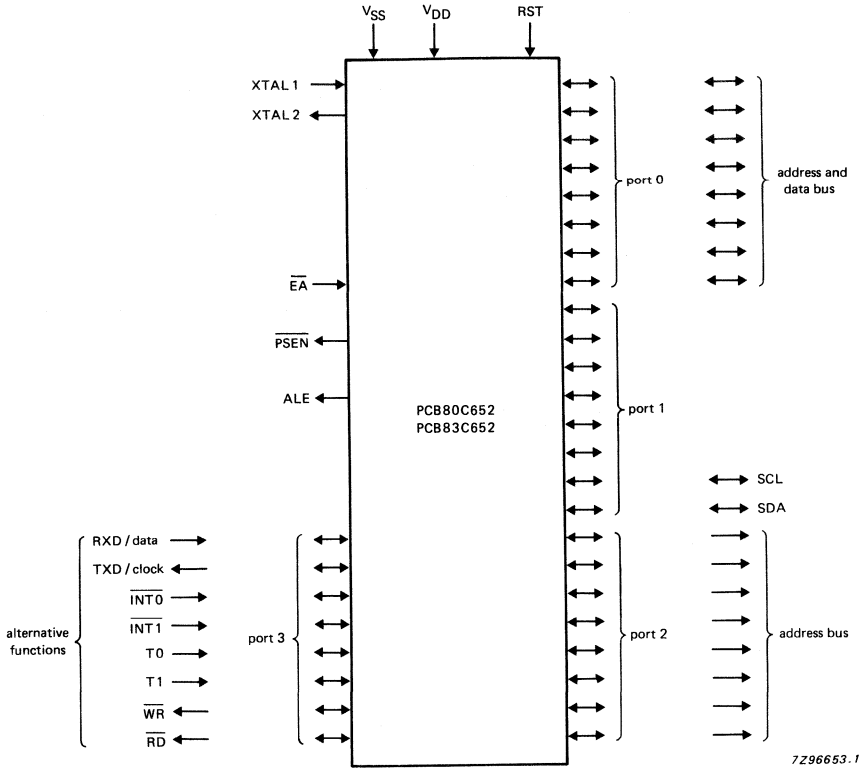


Fig. 2 Functional diagram.

3.1 Pin designation

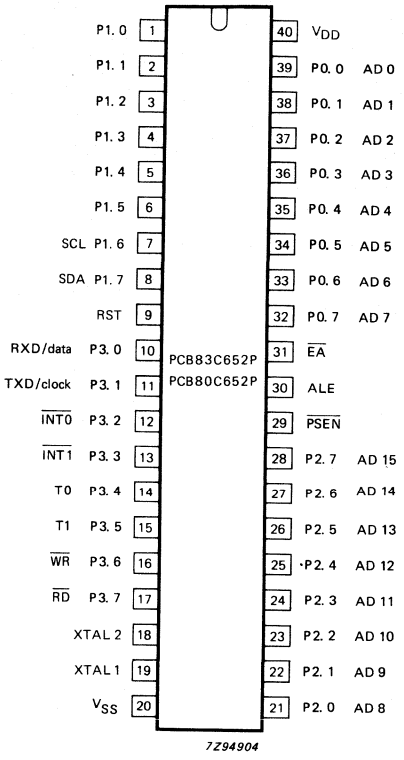


Fig. 3a Pinning diagram for PCB83C652P;PCB80C652P.

Pin designation

1-8 P1.0-P1.7 Port 1: 8-bit quasi-bidirectional I/O port. Port 1 can sink/source one TTL (= 4 LS TTL) input. The port can drive CMOS inputs without external pull-ups, except P1.6/P1.7 which have open drain outputs.

Port pin	Alternative function
P1.6	SCL : I^2C-bus serial port clock line
P1.7	SDA : I^2C-bus serial port data line

9 RST: a high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pull-down permits Power-On reset using only a capacitor connected to V_{DD}.

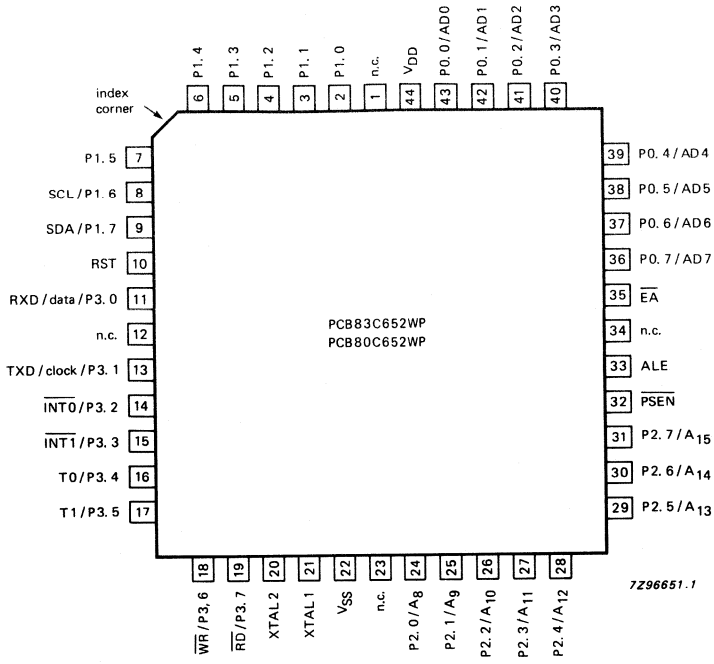


Fig. 3b Pinning diagram for PCB83C652WP;PCB80C652WP.

10-17 P3.0-P3.7 Port 3: 8-bit quasi-bidirectional I/O port with internal pull-ups. It also serves the following alternative functions:

Port pin	Alternative function
P3.0	RXD/data: serial port receiver data input (asynchronous) or data input/output (synchronous)
P3.1	TXD/clock: serial port transmitter data output (asynchronous) or clock output (synchronous)
P3.2	$\overline{INT0}$: external interrupt 0 or gate control input for timer/event counter 0
P3.3	$\overline{INT1}$: external interrupt 1 or gate control input for timer/event counter 1
P3.4	T0: external input for timer/event counter 0
P3.5	T1: external input for timer/event counter 1
P3.6	$\overline{WR}$: external data memory write strobe
P3.7	$\overline{RD}$: external data memory read strobe

		The generation or use of a port 3 pin in its alternative function is carried out automatically by the PCB83C652 provided the associated Special Function Register bit is set high. Port 3 can sink/source one TTL (= 4 LSTTL) input. It can drive CMOS inputs without external pull-ups.
18	XTAL 2	Crystal input 2: output of the inverting amplifier that forms the oscillator. Left open-circuit when an external oscillator clock is used (see figures 9 and 10).
19	XTAL 1	Crystal input 1: input to the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator clock is used (see figures 9 and 10).
20	V_{SS}	Ground: circuit ground potential.
21-28	P2.0-P2.7	Port 2: 8-bit quasi-bidirectional I/O port with internal pull-ups. During access to external memories (RAM/ROM) that use 16-bit addresses (MOVX @DPTR) Port 2 emits the high order address byte. When external RAM is accessed with an 8-bit address (MOVX @Ri) Port 2 emits the contents of the P2 special function register. Port 2 can sink/source one TTL input. It can drive CMOS inputs without external pull-ups.
29	$\overline{PSEN}$	Program Store Enable output: read strobe to the external Program Memory via port 0 and 2. It is activated twice each machine cycle during fetches from external Program Memory. When executing out of external Program Memory two activations of $\overline{PSEN}$ are skipped during each access to external Data Memory. $\overline{PSEN}$ is not activated (remains HIGH) during no fetches from external Program Memory. $\overline{PSEN}$ can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pull-up.
30	ALE	Address Latch Enable output: latches the low byte of the address during accesses to external memory in normal operation. It is activated every six oscillator periods except during an external data memory access. ALE can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pull-up.
31	$\overline{EA}$	External Access input: When $\overline{EA}$ is held at a TTL high level the CPU executes out of the internal Program Memory (ROM) provided the program counter is less than 8192. When $\overline{EA}$ is held at a TTL low level, the CPU executes out of external Program Memory via Port 0 and Port 2 $\overline{EA}$ is not allowed to float.
32-39	P0.7-P0.0	Port 0: 8-bit open drain bidirectional I/O port. Port 0 can sink/source eight LSTTL inputs. It is also the multiplexed low-order address and data bus during access to external memory (during these accesses it activates internal pull-ups)
40	V_{DD}	Power supply: + 5 V power supply pin during normal operation, Idle mode and Power-down mode.

To avoid a 'latch-up' effect at power-on, the voltage on any pin at any time must not be higher or lower than V_{DD} + 0,5 V or V_{SS} - 0,5 respectively.

4.0 FUNCTIONAL DESCRIPTION

As the PCB83C652 contains facilities which are standard to both the PCB80C51 and PCB83C552, only the differences to the above devices are described herein. For details on any of the omitted features, see the PCB80C552 and PCB80C51 sections.

The PCB83C652 is a stand-alone high-performance microcontroller designed for use in real-time applications such as instrumentation and industrial control.

The device provides, in addition to the 80C51 standard functions; an I^2C-bus interface. Aside to expandability via the parallel bus, functions may also be expanded using the I^2C-bus utilizing the complete line of our I^2C clips family.

The PCB83C652 is a control-oriented CPU with on-chip program and data memory. It can be extended with external program and data memory up to 64 Kbytes. For systems requiring extra capability, the PCB83C652 can be expanded using standard memories and peripherals.

The PCB83C652 has two software selectable modes of reduced activity for further power reduction -Idle and Power-down. The Idle mode freezes the CPU while allowing the RAM, timers, serial ports and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator causing all other chip functions to be inoperative.

4.1 Memory organization

The central processing unit (CPU) manipulates operands in three memory spaces; these are the 64 K-byte external data memory, 256-byte internal data memory and the 64 K-byte internal and external program memory. The internal data memory address space is sub-divided into the 256-byte internal data RAM and 128-byte Special Function Register (SFR) address spaces, as shown in Fig. 4. Fig. 5 overleaf, shows the Special Function Register memory map. Internal RAM locations 0-127 are directly and indirectly addressable. Internal RAM locations 128-255 are only indirectly addressable as internal data RAM. The special function register locations 128 - 255 are only directly addressable.

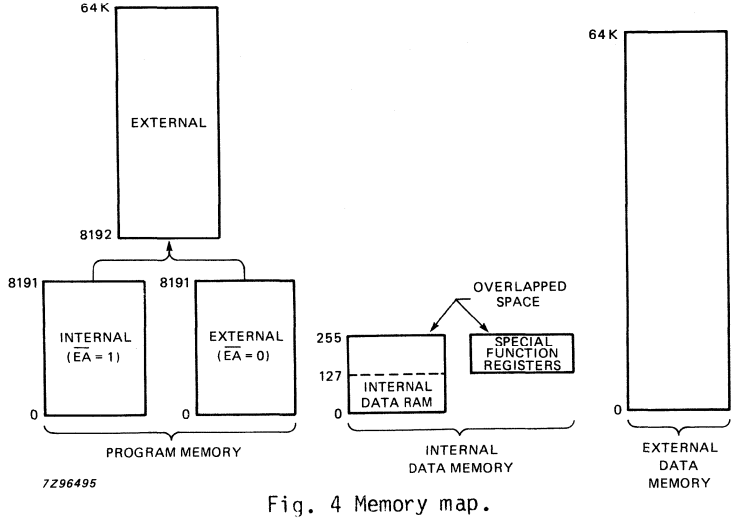
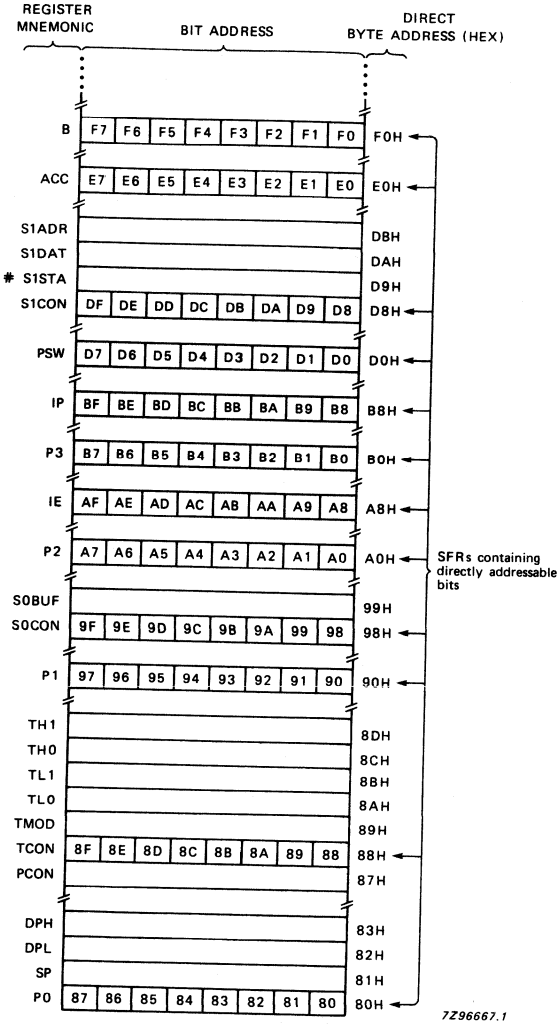


Fig. 4 Memory map.



S1STA is a read-only register.

Fig. 5 SFR memory map.

The internal data RAM contains four register banks (each with eight registers), 128 addressable bits, and the stack. The stack depth is limited by the available internal data RAM and its location is determined by the 8-bit stack pointer. All registers except the program counter and the four 8-register banks reside in the special function register address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, interrupt system registers, timers and serial port registers. There are 128 addressable bit locations in the SFR address space.

The PCB83C652 contains 256 bytes of internal data RAM and 24 special function registers. It provides a non-paged program memory address space to accommodate relocatable code. Conditional branches are performed relative to the program counter. The register-indirect jump permits branching relative to a 16-bit base register with an offset provided by an 8-bit index register. 16-bit jumps and calls permit branching to any location in the contiguous 64 K program memory address space.

4.2 I/O facilities

The PCB83C652 has four 8-bit ports. Ports 0 - 3 are the same as in the 80C51, with the exception of the additional functions of port 1. Port lines P1.7 and P1.6 may be selected as the SDA and SCL lines of serial port SIO 1 (I^2C). Because the I^2C-bus may be active while the device is disconnected from V_{DD}, these pins are provided with open drain drivers.

N.B. Therefore pins P1.7 and P1.6 do not have pull-up devices when used as ports.

Ports 0, 1, 2 and 3 perform the following alternative functions:

- Port 0; provides the multiplexed low-order address and data bus used for expanding the PCB83C652 with standard memories and peripherals.

- Port 1; Port 1 is partly used for the I^2C-bus;

  - SCL and SDA for the I^2C interface, P1.6 and P1.7 respectively.

Bits whose alternate function is not used may be used as normal bidirectional I/O pins.

- Port 2; provides the high-order address bus when expanding the PCB83C652 with external program memory and/or external data memory.

- Port 3; pins can be configured individually to provide:-
  - external interrupt request inputs
  - counter inputs
  - serial port receiver input and transmitter output
  - control signals to READ and WRITE external data memory

The generation or use of a Port 3 pin as an alternative function is carried out automatically by the PCB83C652 provided the associated Special Function Register bit is set HIGH.

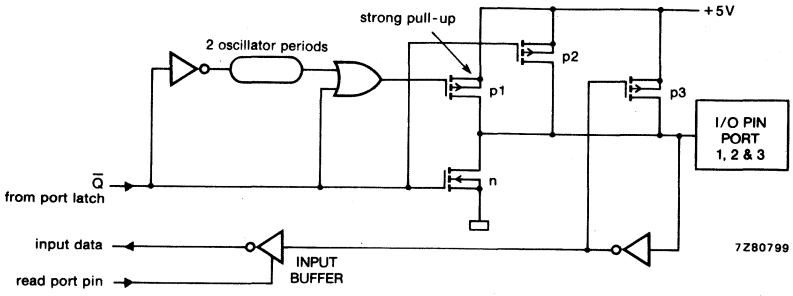


Fig. 6 I/O buffers in the PCB83C652 (Ports 1, 2 and 3).

4.3 Idle and Power-down operation (see Fig. 7)

Idle mode operation permits the interrupt, serial ports and timer blocks to continue to function while the clock to the CPU is halted.

The following functions remain active during IDLE mode. These functions may generate an interrupt or reset and thus end the IDLE mode:

Timer 0, Timer 1
SIO 0, SIO 1
External interrupt

The Power-down operation freezes the oscillator. The Power-down mode can only be activated by setting the PD bit in the PCON register.

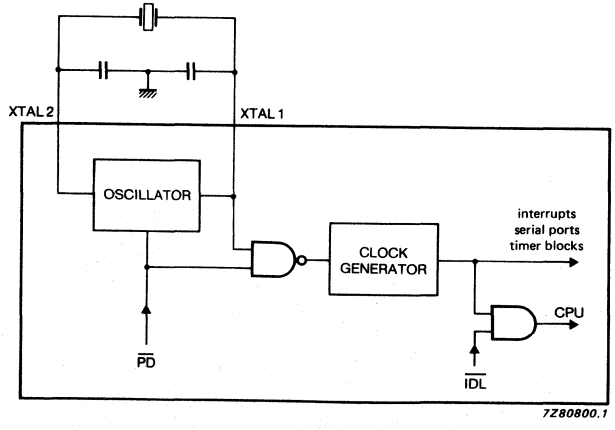


Fig. 7 Internal Idle and Power-down clock configuration.

4.3.1 Power control register (PCON)

These special modes are activated by software via the Special Function Register PCON. Its hardware address is 87H. PCON is not bit addressable.

PCON

SMOD	X	X	X	GF1	GF0	PD	IDL
7	6	5	4	3	2	1	0

Bit		Definition
SMOD	PCON.7	Double Baud rate bit when set to logic 1 the baud rate is doubled when the serial port SIO 0 is being used in modes 1, 2 or 3.
	PCON.6	(reserved)
	PCON.5	(reserved)
	PCON.4	(reserved)
GF1	PCON.3	general-purpose flag bit
GF0	PCON.2	general-purpose flag bit
PD	PCON.1	Power-down bit; setting this bit activates power-down mode.
IDL	PCON.0	Idle mode bit; setting this bit activates the idle mode operation.

If logic 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

4.3.2 Idle mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM and all other registers maintain their data during Idle mode. The status of the external pins during Idle mode is shown in Table 1.

There are two ways to terminate the Idle mode:-

Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware terminating Idle mode. The interrupt is serviced, and following return from interrupt instruction RETI, the next instruction to be executed will be the one which follows the instruction that wrote a logic 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an interrupt, the service routine can examine the status of the flags bits.

The second way of terminating the Idle mode is with an external hardware reset. Since the oscillator is still running, the hardware reset is required to be active for two machine cycles (24 oscillator periods) to complete the reset operation.

4.3.3 Power-down mode

The instruction that sets PCON.1 is the last executed prior to going into the Power-down mode. Once in Power-down mode, the oscillator is stopped. Only the contents of the on-chip RAM are preserved. The Special Function Registers are not saved. A hardware reset is the only way of exiting the Power-down mode.

In the Power-down mode, V_{DD} may be reduced to minimize circuit power consumption. The voltage must not be reduced until the Power-down mode is entered, but must be restored before the hardware reset is applied which will free the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

The status of the external pins during Power-down mode is shown in Table 1. If the Power-down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a logic 1, the port pin is held HIGH during the Power-down mode by the strong pull-up transistor p1 (see Fig. 6).

Table 1 Status of the external pins during Idle and Power-down modes.

mode	memory	ALE	$\overline{PSEN}$	Port 0	Port 1	Port 2	Port 3
Idle	internal	1	1	port data	port data	port data	port data
Idle	external	1	1	floating	port data	address	port data
Power-down	internal	0	0	port data	port data	port data	port data
Power-down	external	0	0	floating	port data	port data	port data

Note: Port 1.7 and 1.6 if selected, function as SDA and SCL respectively in the Idle mode.

4.4 Interrupt system (see Fig. 8)

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution a multiple-source, two-priority-level, nested interrupt system is provided. Interrupt response latency is from 3 µs to 8 µs when using a 12 MHz crystal. The PCB83C652 acknowledges interrupt requests from six sources as follows:

- $\overline{INT0}$ and $\overline{INT1}$; externally via pins 14 and 15 respectively
- Timer 0 and Timer 1; from the two internal counters
- I^2C serial I/O port interrupt
- UART serial I/O port interrupt.

Each interrupt vectors to a separate location in program memory for its service program. Each source can be individually enabled or disabled by a corresponding bit in the IE register, moreover each interrupt may be programmed to a high or low priority level using a corresponding bit in the IP register. Also all enabled sources can be globally disabled or enabled. Both external interrupts can be programmed to be level-activated or transition-activated, and an active LOW level allows "wire-ORing" of several interrupt sources to the input pin.

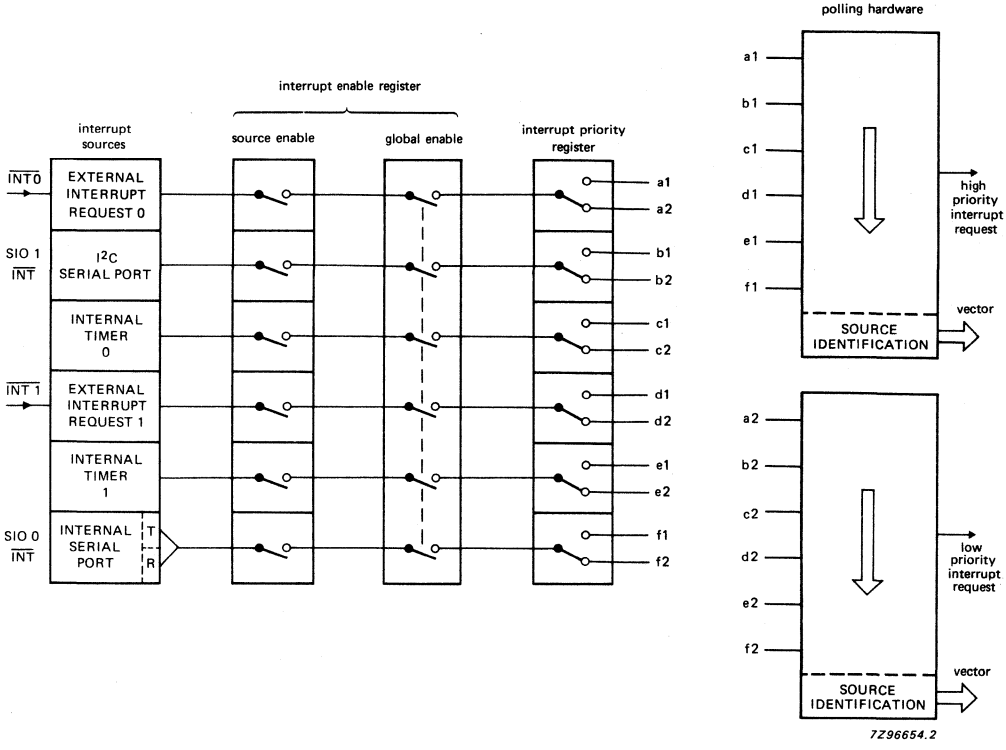


Fig. 8 Interrupt system.

4.4.1 Interrupt enable register

IE

EA	-	ES1	ES0	ET1	EX1	ET0	EX0
7	6	5	4	3	2	1	0

Bit		Function
IE.7	EA	General enable/disable control. 0 = No interrupt is enabled 1 = Any individually enabled interrupt will be accepted
IE.6	-	Unused
IE.5	ES1	Enable SIO 1 (I^2C) interrupt
IE.4	ES0	Enable SIO 0 (UART) interrupt
IE.3	ET1	Enable Timer 1 interrupt
IE.2	EX1	Enable External 1 interrupt
IE.1	ET0	Enable Timer 0 interrupt
IE.0	EX0	Enable External 0 interrupt

Where 0 = Interrupt disabled
1 = Interrupt enabled

4.4.2 Interrupt priority register

IP

X	X	PS1	PS0	PT1	PX1	PT0	PX0
7	6	5	4	3	2	1	0

Bit		Function
IP.7	-	Unused
IP.6	-	Unused
IP.5	PS1	SIO 1 (I^2C) interrupt priority level
IP.4	PS0	SIO 0 (UART) interrupt priority level
IP.3	PT1	Timer 1 interrupt priority level
IP.2	PX1	External interrupt 1 priority level
IP.1	PT0	Timer 0 interrupt priority level
IP.0	PX0	External interrupt 0 priority level

Interrupt priority levels are as follows: 0 = low priority
1 = high priority

4.4.3 Interrupt vectors

Table 2 shows the interrupt vectors. The vectors indicate the ROM location where the appropriate interrupt service routine starts.

Table 2 Interrupt vectors

Source		Vector
External 0	X0	0003H
Timer 0 overflow	T0	000BH
External 1	X1	0013H
Timer 1 overflow	T1	001BH
Serial I/O 0 (UART)	S0	0023H
Serial I/O 1 (I^2C)	S1	002BH

4.4.4 Interrupt priority

Each interrupt source can be provided by software with two priority levels; high and low. If both priorities are requested simultaneously, the processor will branch to the high priority vector. If there are simultaneous requests from sources of the same priority, then interrupts will be serviced in the following order left to right:

X0 S1 T0 X1 T1 S0

A low priority interrupt routine can only be interrupted by a high priority interrupt. A high priority interrupt cannot be interrupted.

4.5 Oscillator circuitry

The oscillator circuitry of the PCB83C652 is a single-stage inverting amplifier in a Pierce oscillator configuration. The circuitry between XTAL 1 and XTAL 2 is basically an inverter biased to the transfer point. Either a crystal or ceramic resonator can be used as the feedback element to complete the oscillator circuitry. Both are operated in parallel resonance. XTAL 1 is the high gain amplfier input, and XTAL 2 is the output (see Fig. 9). To drive the PCB83C652 externally, XTAL 1 is driven from an external source and XTAL 2 left open-circuit (see Fig. 10).

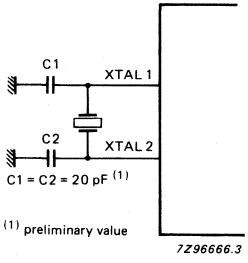


Fig. 9 PCB83C652 oscillator circuit.

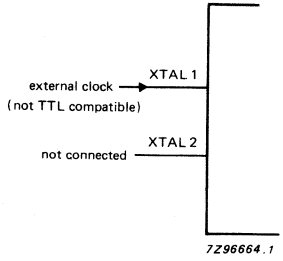


Fig. 10 Driving the PCB83C652 from an external source.

4.6 Reset circuitry (see Fig. 11)

The reset circuitry for the PCB83C652 is connected to the reset pin RST. A Schmitt trigger is used at the input for noise rejection. The output of the Schmitt trigger is sampled by the reset circuitry every machine cycle.

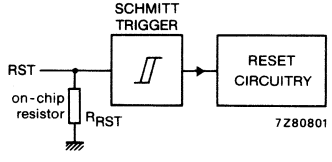


Fig. 11 Reset configuration at RST.

A reset is accomplished by holding the RST pin HIGH for at least two machine cycles (24 oscillator periods). The CPU responds by executing an internal reset. During reset ALE and $\overline{PSEN}$ output a HIGH level. In order to perform a correct reset, this level must not be affected by external elements.

The internal reset is executed during the second cycle in which RST is HIGH and is repeated every cycle until RST goes LOW. It leaves the internal registers as follows:

Register	Content
ACC	0000 0000
B	0000 0000
DPL	0000 0000
DPH	0000 0000
IE	0X00 0000
IP	XX00 0000
PCH	0000 0000
PCL	0000 0000
PCON	0XXX 0000
PSW	0000 0000
P0 - P3	1111 1111
S0BUF	XXXX XXXX
S0CON	0000 0000
S1ADR	0000 0000
S1CON	X000 0000
S1DAT	0000 0000
S1STA	1111 1000
SP	0000 0111
TCON	0000 0000
TH0, TH1	0000 0000
TL0, TL1	0000 0000
TMOD	0000 0000

The internal RAM is not affected by reset. When V_{DD} is turned on, the RAM content is indeterminate.

4.6.1 Power-on reset (see Fig. 12)

When V_{DD} is turned on, an automatic reset can be obtained by connecting the RST pin to V_{DD} via a 2.2 µF capacitor. When the power is switched on, the voltage on the RST pin is the difference between V_{DD} and the capacitor voltage, and decreases from V_{DD} as the capacitor charges through the internal resistor (R_{RST}) to ground. The larger the capacitor, the more slowly V_{RST} decreases. V_{RST} must remain above the lower threshold of the Schmitt trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

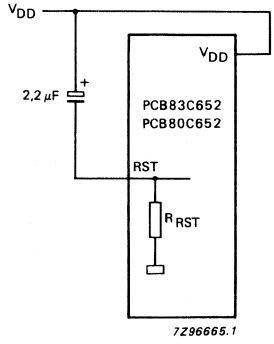


Fig. 12 Power-on reset.

4. The MAB84X1 microcontroller family

CONTENTS – THE MAB84X1 MICROCONTROLLER FAMILY

		page
1.0	DESCRIPTION	4-5
2.0	FEATURES	4-5
3.0	PACKAGE OUTLINES	4-6
3.1	Pin designation MAB8401B "piggy-back" version bottom pinning, MAB8411, MAB8421, MAB8441 and MAB8461	4-6
3.2	MAB8401B (top pinning)	4-7
4.0	BOND-OUT VERSION MAB8401WP	4-8
4.1	Pad designation	4-8
5.0	FUNCTIONAL DESCRIPTION	4-14
5.1	Program memory (ROM)	4-14
5.2	Data memory (RAM)	4-14
5.3	Input/Output	4-17
5.3.1	Parallel ports	4-17
5.3.2	Serial I/O	4-19
5.4	Interrupts	4-20
5.4.1	Interrupt logic	4-20
5.4.2	Interrupt program examples	4-22
5.5	High current outputs	4-24
5.6	Test inputs T1 and T0/$\overline{\text{INT}}$	4-24
5.6.1	Test input T1	4-24
5.6.2	Test input T0/$\overline{\text{INT}}$	4-25
5.7	Oscillator and clock	4-25
5.8	Timer/event counter	4-26
5.9	Program status word	4-27
5.10	Program counter	4-28
5.11	Central processing unit	4-29
5.12	Reset	4-30
5.13	Instruction set	4-31
6.0	MAB8401WP $\overline{\text{HALT}}$ FUNCTION	4-39

		page
7.0	GENERATION OF CLOCK SIGNALS	4-40
7.1	General PCB layout	4-40
7.2	Crystal clock generator	4-40
7.3	Use of ceramic resonators for clock generation	4-42
7.4	LC oscillators for clock generation	4-42
7.5	Single crystal oscillator for multiple device use	4-43
7.5.1	Central oscillator	4-43
7.5.2	One MAB84X1 driving another separate MAB84X1	4-44
7.5.3	Buffered on-chip oscillators	4-44
8.0	TIMING	4-47
8.1	Clock adjustment	4-47
8.2	Timing requirements for the P23 and SCLK input signals	4-49
8.3	Timing requirements for the P23 and SCLK output signals	4-50

1.0 DESCRIPTION

The MAB84X1 family of single-chip 8-bit microcontrollers are manufactured in N-MOS. The family consists of 5 devices:

- MAB 8401 - 128 RAM bytes, external program memory, with 8-bit LED-driver (10 mA), emulation of MAB/F8422/42 possible
- MAB/MAF 8411 - 1 K ROM/ 64 RAM bytes plus 8-bit LED driver
- MAB/MAF 8421 - 2 K ROM/ 64 RAM bytes plus 8-bit LED-driver
- MAB/MAF 8441 - 4 K ROM/128 RAM bytes plus 8-bit LED-driver
- MAB/MAF 8461 - 6 K ROM/128 RAM bytes plus 8-bit LED-driver

Each type has 20 quasi-bidirectional I/O port lines, two serial I/O lines, three single-level vectored interrupts (ext. interrupt, serial I/O, timer/evnt), an 8-bit timer-event counter and an on-board clock oscillator and clock circuits. Two 20-pin versions, MAB/F8422 and MAB/F8442 are also available.

This microcontroller family is an efficient controller as well as an arithmetic processor. The instruction set is based on that of the MAB8048. The microcontrollers have extensive bit handling ability and facilities for both binary and BCD arithmetic.

2.0 FEATURES

- 8-bit: CPU, ROM, RAM, I/O in a single 28-lead DIL package (not MAB8401B)
- 1 K, 2 K, 4 K or 6 K ROM bytes
- 64 or 128 RAM bytes
- 20 quasi-bidirectional I/O port lines
- Two test inputs: may be used for interrupts, conditional jumps/branches or, zero voltage cross-over detection
- Single-level vectored interrupts: external, timer/event counter, serial I/O
- Serial I/O supported by I^2C serial communication bus for single master and multi-master systems (serial I/O data via an existing port line and clock via a dedicated line)
- 8-bit programmable timer/event counter
- Internal oscillator
- Over 80 instructions (based on MAB8048). All instructions 1 or 2 cycles
- Single 5 V supply (± 10%)
- Operating temperature range:

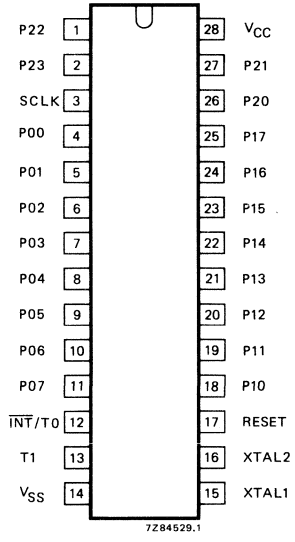
0 to + 70 °C	MAB84X1	family
-40 to + 85 °C	MAF84X1	family
-40 to + 110 °C	MAF84AX1	family

extended temperature range not available for MAB8401

3.0 PACKAGE OUTLINES

MAB/MAF8401B: 28-lead 'Piggy-back' package (with up to 28-pin EPROM on top)
MAB/MAF8401WP: 68-lead plastic leaded chip-carrier (PLCC) (SOT-188)
MAB/MAF8411/21/41/61P: 28-lead DIL; plastic (SOT-117)
MAF84A11/21/41/61P: 28-lead DIL; plastic (int. heat spreader) (SOT-117)
MAB8411/21/41/61T: 28-lead mini-pack; plastic (SO-28; SOT-136)

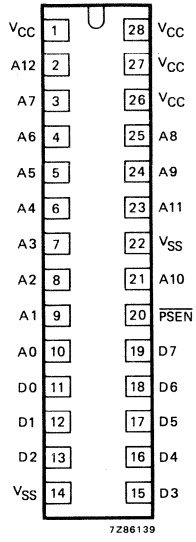
3.1 Fig. 1. PIN DESIGNATION MAB8411/21/41/61 also MAB8401B "piggyback" bottom pinning, (for top pinning see Fig. 2).



PIN DESIGNATION

designation	pin no.	function
V_{SS}	14	Ground
V_{CC}	28	Power supply, + 5 V.
P00-P07	4-11	Port 0. 8-bit quasi-bidirectional I/O port.
P10-P17	18-25	Port 1. 8-bit quasi-bidirectional I/O port.
P20-P23	26,27,1,2	Port 2. 4-bit quasi-bidirectional I/O port; P23 is the serial data input/output in serial I/O mode.
SCLK	3	Bidirectional clock for serial I/O
$\overline{INT}$/T0	12	External interrupt input (negative going edge). Testable using JT0 and JNT0.
T1	13	Input pin testable using the JT1, JNT1 instructions. Can be designated the event counter input, using the SRTR CNT instruction. Also allows zero cross-over sensing of slowly moving A.C. inputs.
RESET	17	Input used to initialize the processor (active HIGH).
XTAL1	15	Connection to timing component (crystal) which determines the frequency of the internal oscillator. Also the input for an external clock source.
XTAL2	16	Connection of the other side of timing component.

3.2 MAB8401B (top pinning)



PIN DESIGNATION

designation	pin no.	function
V_{SS}	14,22	Ground
V_{CC}	1,26-28	Power supply, + 5 V.
A0-A12	10-3,25,24, 21,23,2	Address outputs
D0-D7	11-13,15-19	Data input
$\overline{PSEN}$	20	Program store enable

Fig. 2 Pinning diagram for MAB8401B 'Piggy-back' version top pinning (for bottom pinning see Fig. 1); to access a 2732 or 2764 EPROM.

Note: Access times for ROMS/RAMS to be below 1 μs.

Note: 1) A 2732 EPROM has to be inserted with pin 1 in the pin 3 location of the MAB8401B top pinning

2) A 2716 EPROM may be used if its Vpp pin (pin 23 on MAB8401B top pinning) is disconnected from A11 and connected to V_{CC}.

When emulating MAB8401 family software using the MAB8401B, care must be taken not to exceed the memory capacity of the final product. Special attention should be paid to examples involving indirect addressing.

4.0 BOND-OUT VERSION MAB8401WP

The bond-out version is a microcontroller that contains no on-chip ROM, but has address, data and control lines brought-out to access an external ROM or EPROM. Therefore, this version has more pins than the standard microcontrollers with on-chip ROM. It has all the features of the other members of the MAB84X1 family, including emulation facilities for the MAB/F8422/42 (20-pin version). It can address 8 K-bytes of ROM. The RAM has 128 bytes.

The PLCC (plastic leaded chip carrier) is designed for use in SMD (surface mounted device) circuitry. Fig. 3. shows the pad designation.

4.1 PAD DESIGNATION MAB8401WP

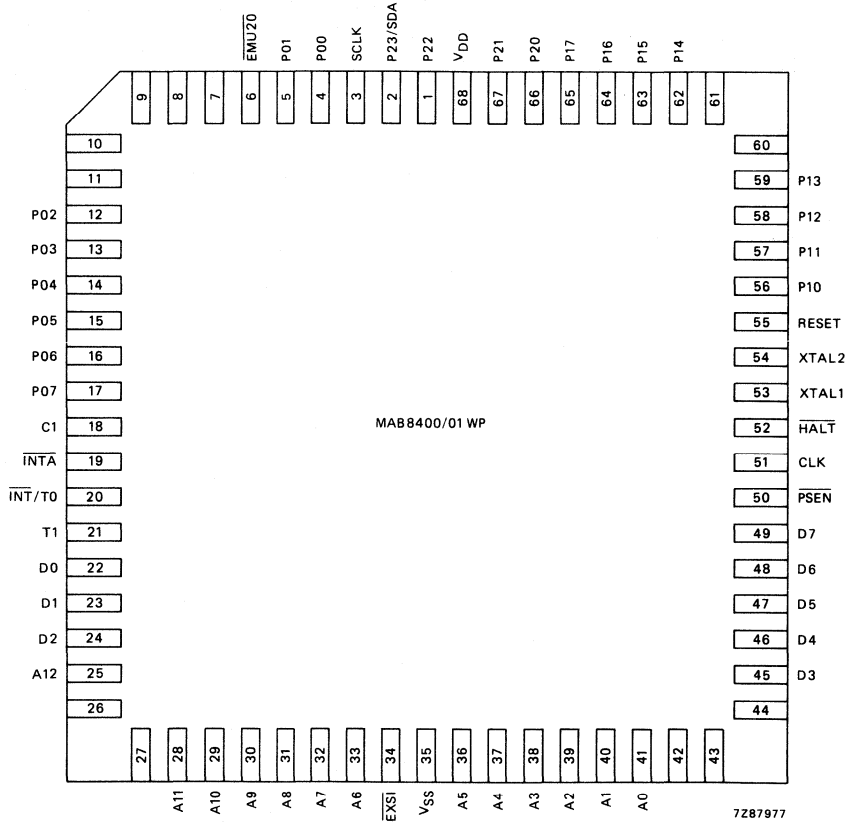


Fig. 3 MAB8401WP PLCC pad designation.

CHIP CARRIER DESIGNATION

4.1 Pad Designation

designation	pad N^{o}	function
V_{SS}	35	Ground
V_{CC}	68	Power supply: +5V
P00-P07	4-5, 12-17	Port 0: 8-bit quasi-bidirectional
P10-P17	56-59, 62-65	Port 1: 8-bit quasi-bidirectional
P20-P22	66, 67, 1	Port 2: 4-bit quasi-bidirectional
P23/SDA	2	P23 serial data I/O line
SCLK	3	Bidirectional clock for serial I/O
$\overline{INT}$/T0	20	External interrupt input (sensitive on negative going edge): testable using JT0 or JNT0 instructions.
T1	21	T1 input pin: testable using the JT1 or JNT1 instructions. It can be designated as event counter input using the STRT CNT. It can also be used to detect zero cross-over of slowly moving A.C. inputs.
RESET	55	Input to initialize the processor (active HIGH)
XTAL1	53	Connection to timing component: (e.g. crystal) that determines the frequency of the internal oscillator. It is also the input for an external clock source.
XTAL2	54	Connection to other side of timing component
$\overline{EXSI}$	34	External serial I/O interrupt (active LOW): for emulation of MAB/F8422/42 (MAB84X1 back bias)
A0-A12	41-36, 33-28,25	Program memory address outputs: A0 = LSB, A12 = MSB. Address output changes on Ø3 of TS8.
D0-D7	22-24,45-49	Data input lines: used for reading external program memory, D0 = LSB, D7 = MSB.
SCLK	51	Clock output buffered from XTAL2: On the positive going edge the internal Ø clock goes high.

Pad designation (continued)

$\overline{PSEN}$	50	Program store enable: This signal is used for enabling the external EPROM (e.g. on the 'piggy-back' version). For emulation, it enables the emulation memory and indicates machine cycles. Active LOW during TS9, TS10 of each machine cycle and TS1 of the following machine cycle.
$\overline{C1}$	18	Cycle 1 indication output (active LOW): During emulation, this signal indicates the op-code fetch cycle (useful for external instruction decoding, real-time trace). Active from start of TS10 of the cycle preceeding cycle 1, until the start of TS10 of cycle 1,
$\overline{HALT}$	52	Halt input (active LOW): If activated the current instruction is finished and the microcontroller stops execution ($\overline{HALT}$ mode). The next program counter address is available on the address bus. Program counter and timer/event counter are no longer updated. The serial I/O finishes the current transmit/receive operation and goes into the idle state. Interrupts are not sampled in the $\overline{HALT}$ mode, they are only sampled when the microcontroller is running. Interrupt routines may be single-stepped as a normal program.
$\overline{INTA}$	19	Interrupt acknowledge output (active LOW): It indicates any interrupt acception. Active from start of TS8 of the interrupted cycle, until the start of TS7 of the second cycle of the internally forced 'CALL vector address' instruction. During $\overline{INTA}$ active, the address bus shows the address that has been saved in the stack (return address) the $\overline{C1}$ output indicates opcode fetch cycles as if a user CALL was executed.
$\overline{EMU20}$	6	Emulate 20-pin version MAB/F8422/42 (active LOW): MAB8401 not connected.

The diagram below; Fig. 3(a), shows the connection of EPROM to 'piggy-back' package MAB8401B.

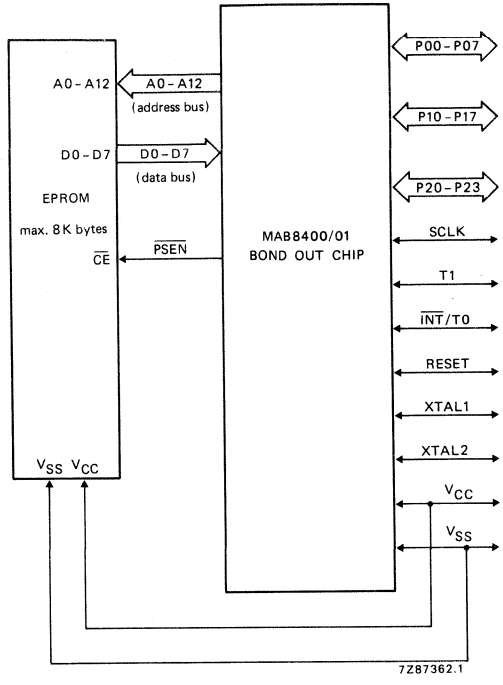


Fig. 3(a) connection of EPROM to MAB8401B.

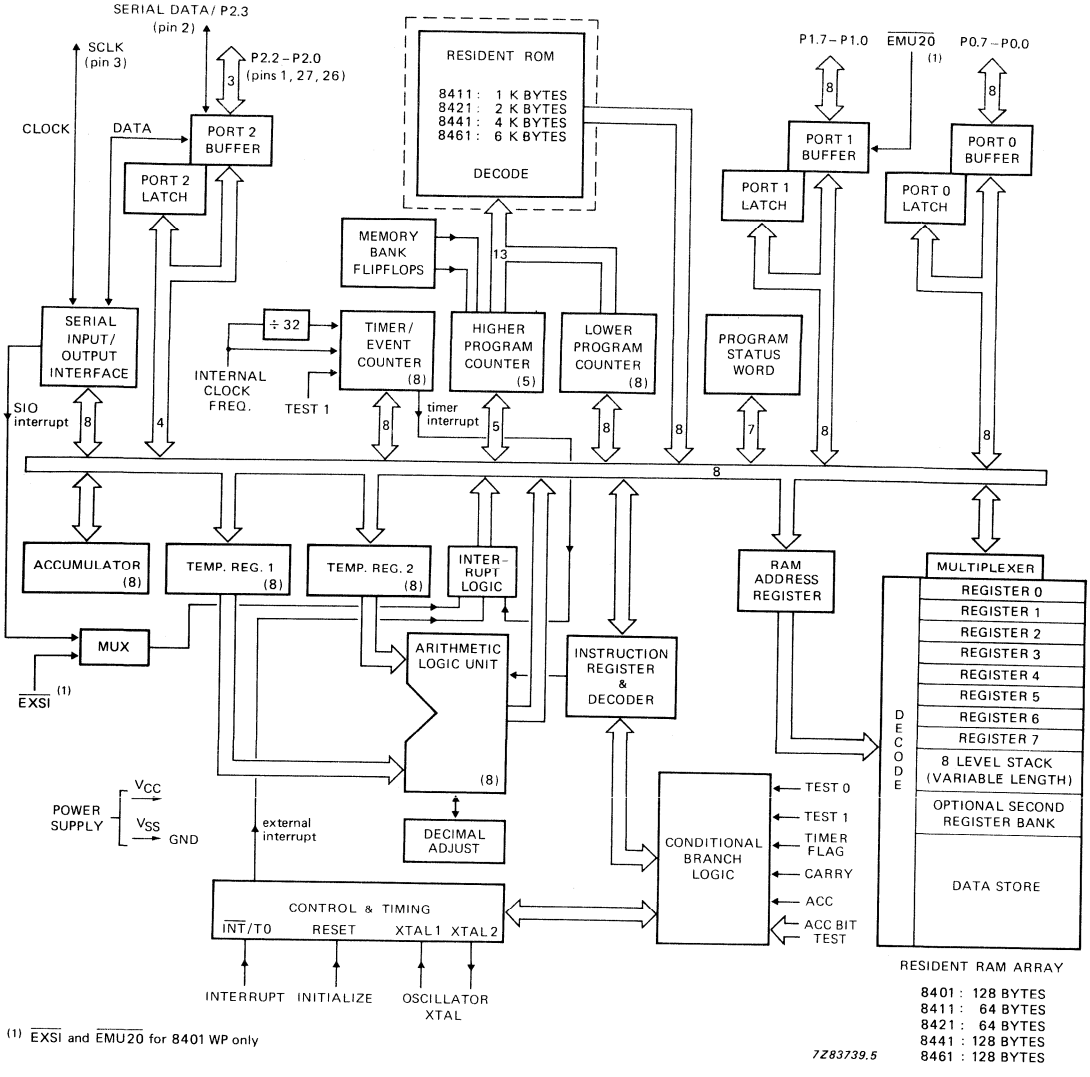
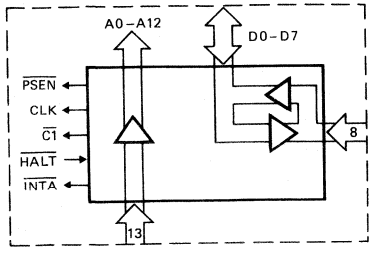
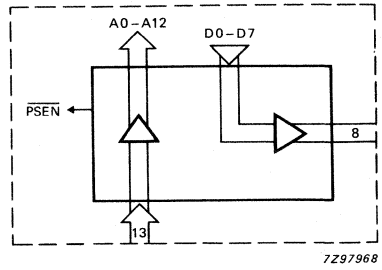


Fig. 4 Block diagram of the MAB84X1 family.



(a)

Fig. 4(a) Replacement of dotted part in Fig. 4 showing the MAB8401WP bond-out version.



(b)

Fig. 4(b) Replacement of dotted part in Fig. 4 for MAB8401B ('Piggy-back').

5.0 FUNCTIONAL DESCRIPTION

5.1 Program memory (ROM)

The program memory consists of 1 K, 2 K, 4 K or 6 K bytes (8-bit words), which are addressed by the program counter. The memory is mask-programmed at production. Because the MAB84X1 family offers a range of ROM capacities to suit the application, ROM expansion is not required. Fig. 5 shows the program memory map.

Four program memory locations are of special importance:

location 0 - contains the first instruction to be executed after the processor is initialized (RESET)
location 3 - contains the vector of an external interrupt service subroutine
location 5 - contains the vector of a serial I/O interrupt service subroutine
location 7 - contains the vector of a timer/event counter interrupt service subroutine

Program memory is arranged in banks of 2 K bytes, organised in pages of 256 bytes. These are selected by SEL MB instructions. Only the unconditional branch instructions (JMP and CALL) can cause jumps over page boundaries. Memory bank boundaries can be crossed only by using the same unconditional branch instructions after the appropriate memory bank has been selected. A CALL instruction can transfer control to a subroutine on any page, RET and RETR instructions can transfer control from a subroutine back to the main program. Note, if a memory bank change is necersary, the required bank must be selected prior to a CALL or JMP instruction.

5.2 Data memory (RAM)

Data memory consists of 64 or 128 bytes (8-bit words). All locations are indirectly addressable using RAM pointer registers; up to 16 designated locations are directly addressable. Memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Fig. 6 shows the data memory map.

Location 0 to 7 are designated as working registers, directly addressable by the direct register instructions. Because these registers are easily addressed and require the minimum instruction bytes to manipulate their contents, they are used to store frequently accessed intermediate results. This bank of registers can be selected by the SEL RB0 instruction.

Executing the select register bank instruction SEL RB1, designates locations 24 to 31 as the working registers, replacing locations 0 to 7. These are also directly addressable. This second bank of working registers may be used as an extension of the first or reserved for use during interrupt service subroutines, saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first two locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations.

All RAM locations make efficient program loop counters when used with the decrement register and test instruction DJNZ.

Locations 8 to 23 may be designed as an 8-level program counter stack (2 locations per level), or as general purpose RAM. The program counter stack (Fig. 7) enables the processor to keep track of the return addresses and status generated by interrupts or CALL instructions by storing the contents of the program counter prior to servicing the subroutine. A 3-bit stack pointer determines which of the program counter stack's eight register pairs will be loaded with the next return address generated.

The stack pointer, when initialized to 000 by RESET, points to RAM locations 8 and 9. On the first subroutine CALL or interrupt, the contents of the program counter and bits 4, 6 and 7 of the program status word (PSW) are transferred to locations 8 and 9. The stack pointer increments by one and points to locations 10 and 11 ready for another call. Because an address may be up to 13 bits long, two bytes must be used to store each address.

At the end of a subroutine, which is signalled by a return instruction (RET), the stack pointer decrements by one and the contents of the register pair on top of the stack are transferred to the program counter. The saved PSW bits are transferred to the PSW only by the RETR instruction.

If not all 8 levels of subroutine and interrupt nesting are used, the unused portion of the stack may be used as indirectly addressable RAM. Locations 32 to 127 may be used for storage of program variables or data.

Nesting of subroutines within subroutines may continue until overflow of the stack. If an overflow does occur, the deepest address (location 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The value of the saved contents of the program counter is different for an interrupt CALL compared to a normal CALL to subroutine. With an interrupt CALL, the program counter return address is saved; with a subroutine CALL, the saved program counter value is one less than the program counter return address.

Figs. 5 and 6 show the program memory and data memory maps. Fig. 7 illustrates the structure of the program counter stack.

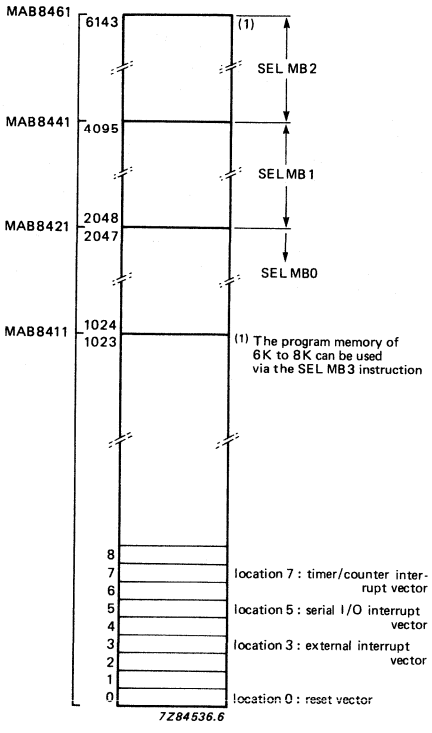


Fig. 5 The program memory map

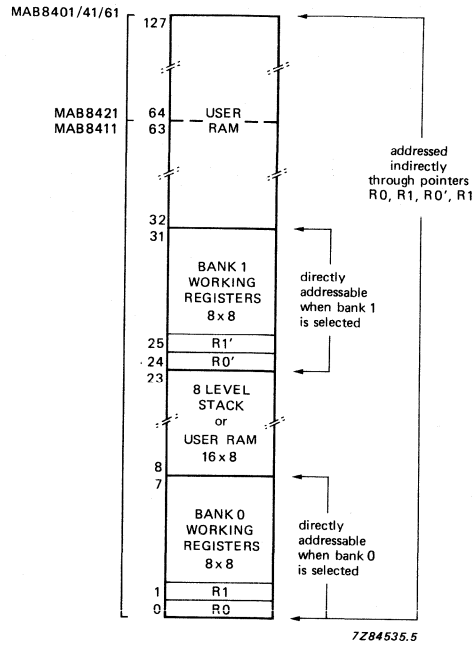


Fig. 6 The data memory map

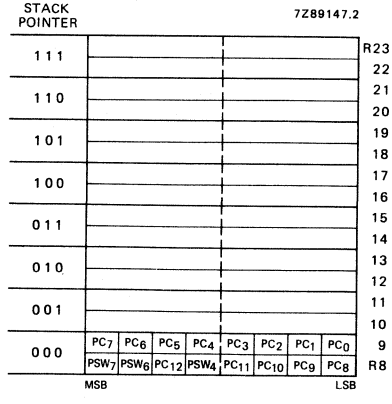


Fig. 7 Program counter stack

5.3 Input/Output

The MAB84X1 family has 23 I/O lines arranged as:

- two parallel ports of 8 lines (P00 to P07, P10 to P17). Each line of Port 1 can sink 10 mA.
- one parallel port of 4 lines (P20 to P23)
- a serial I/O consisting of a data line shared with a parallel port line (P23) and a separate clock line SCLK,
- one external interrupt and test input ($\overline{INT}$/T0): when used as a test input it can be tested by the conditional branch instructions JT0 and JNT0.
- one test input (T1), which can alter program sequences when tested by conditional jump instructions JT1 and JNT1; T1 can also be used as an input to the timer/event counter, or used for zero-cross detection.

5.3.1 Parallel ports

Output data written to a port is latched and remains unchanged until rewritten. Input data is not latched and so must remain present until read by an input instruction.

Input lines are fully TTL compatible, output lines can drive one standard TTL load *. Fig. 9(c) shows the quasi-bidirectional I/O interface with push-pull output with pull-up transistor. Each line is continuously pulled up to +5 V through this high impedance (TR3 cut off). When a 0 is written to the line, the low impedance of TR1 overcomes the pull-up and provides TTL current sinking capability. When a 1 is written, TR2 is momentarily switched on to give fast pull-up. One state of a machine cycle later, the 1 level is still maintained by the high impedance pull-up, so that the line can be used as an input line. This can be done by software control. After RESET, all I/O lines are in the input mode (i.e. TR1 is high impedance). The 1 on these lines can then be easily pulled down to a '0' by CMOS or TTL circuits.

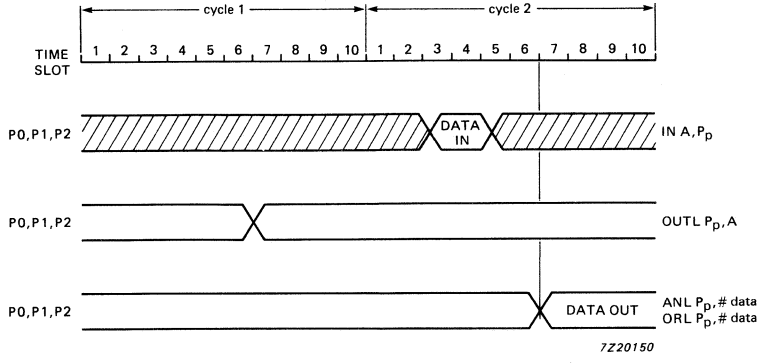
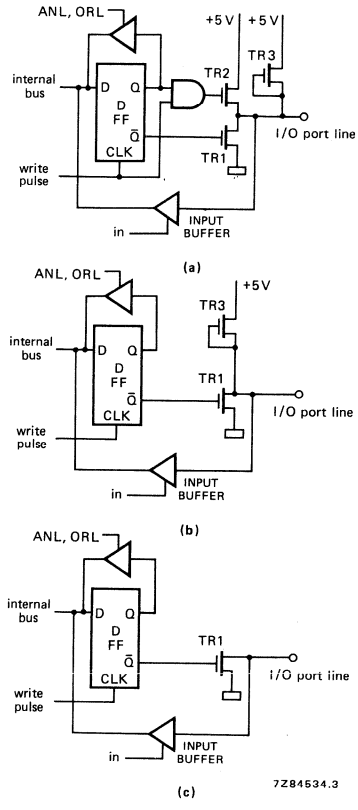


Fig. 8 Shows the timing diagram for all ports using IN, OUTL, ANL and ORL instructions. For the OUTL instruction data changes on time slot 7 of cycle 1. For the ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

* Port 1 can sink up to 8 LEDs.



Option 1

Option 2

Option 3

Fig. 9 Quasi-bidirectional I/O interface with port options:

(a) open drain output without pull-up transistor, (b) open drain output with pull-up transistor, (c) push-pull output with pull-up transistor.

Note: Port options of MAB8401B/WP are;

All ports except P23 = option 3 only
Pin T1 and P23 = option 1 only

(Please state these port options in the order entry forms)

5.3.2 Serial I/O

The design of the MAB84X1 family serial I/O system allows any number of MAB84X1 family devices to be interconnected by the two-line serial I^2C-bus. The interface lends the ability to two or more devices to communicate without interrupting the operation of any other devices on the bus. This is an outstanding attribute of the system. It is achieved by allocating a specific 7-bit address to each device and providing a system whereby a device reacts only to message prefixed with its own address or the 'general call' address. Address recognition is performed by the interface hardware so that operation of the microcontroller need only be interrupted when a valid address has been received. Such a system saves significant processing time and memory space compared with a conventional microcontroller employing a software serial interface.

When the addressing facility is not required, for instance in a system with only two microcontrollers on the serial bus, direct data transfer without addressing can be performed. In multi-master systems, an automatically invoked arbitration procedure prevents two or more devices from continuing simultaneous transmission.

For the two 20 pin versions, the MAB8422/42, a separate chapter of this user manual addresses itself specifically to these two chips.

A more detailed description of serial I/O is contained in a separate section of this manual.

5.4 Interrupts

When the external interrupt is enabled, a HIGH-to-LOW transition on the $\overline{INT}$/TO input initiates an external interrupt subroutine which causes a call to program memory location 3 following completion of the current instruction. The serial I/O interrupt when enabled, invokes a call to location 5. A timer/event counter overflow interrupt, when enabled, invokes a call to location 7.

External interrupts are always latched, even when the external interrupt is disabled. Therefore, keyboard or sensor interrupt requests are not lost when the processor must first perform some necessary functions whilst the external interrupt is disabled. When an interrupt subroutine starts, the program counter contents and bits 4, 6 and 7 of the PSW will be saved in the program counter stack. Accumulator contents have to be saved by user software. Interrupt acknowledgement can be carried out by software via port pins. All interrupt routines must reside in memory bank 0.

The interrupt system is single-level; once an interrupt is detected, further interrupt requests are latched but ignored until the execution of a RETR instruction re-enables the interrupt input logic. After executing RETR, the program continues in the main part. When a second interrupt occurs during the running of the first routine the computer will enter the second routine after having executed one instruction in the main program. If 2 or 3 interrupts occur simultaneously, their priority is: (1) external, (2) serial I/O, (3) timer/ event counter,

Another external interrupt can be created by enabling the timer/event counter interrupt loading FFH into the counter (one less than overflow) and enabling the event counter mode. A LOW to HIGH transition on the T1 input will then initiate an interrupt subroutine and cause a call to the timer/counter interrupt vector location 7.

5.4.1 Interrupt logic

The external interrupt of the MA884X1 is active on the negative edge; a HIGH to LOW transition on the $\overline{INT}$/TO input will be latched in a 'digital filter/latch' (3-bit shift register in which the negative interrupt is stored; see interrupt logic diagram Fig. 10) and when enabled, initiates an external interrupt service routine. The interrupt activity remains latched in the 'digital filter' until it is taken over by the 'External Interrupt Flag', which in turn resets the digital filter. This flag can only be set when external interrupts are enabled.

The 'External Interrupt Flag' when set, causes the current instruction to end, and, if INT has been asserted before time slot 7 of the last cycle of the current instruction, forces a subroutine CALL to program memory location 3. If asserted after this time, the next instruction will be performed first. During this forced CALL the 'External Interrupt Flag' is reset, which enables new interrupts to be latched in the digital filter/latch'. Also the 'Interrupt in Progress Flag' is set, which causes other interrupts to be ignored and latched until the RETR instruction is executed. These (latched) interrupts will start an interrupt routine after the program has returned to the main part.

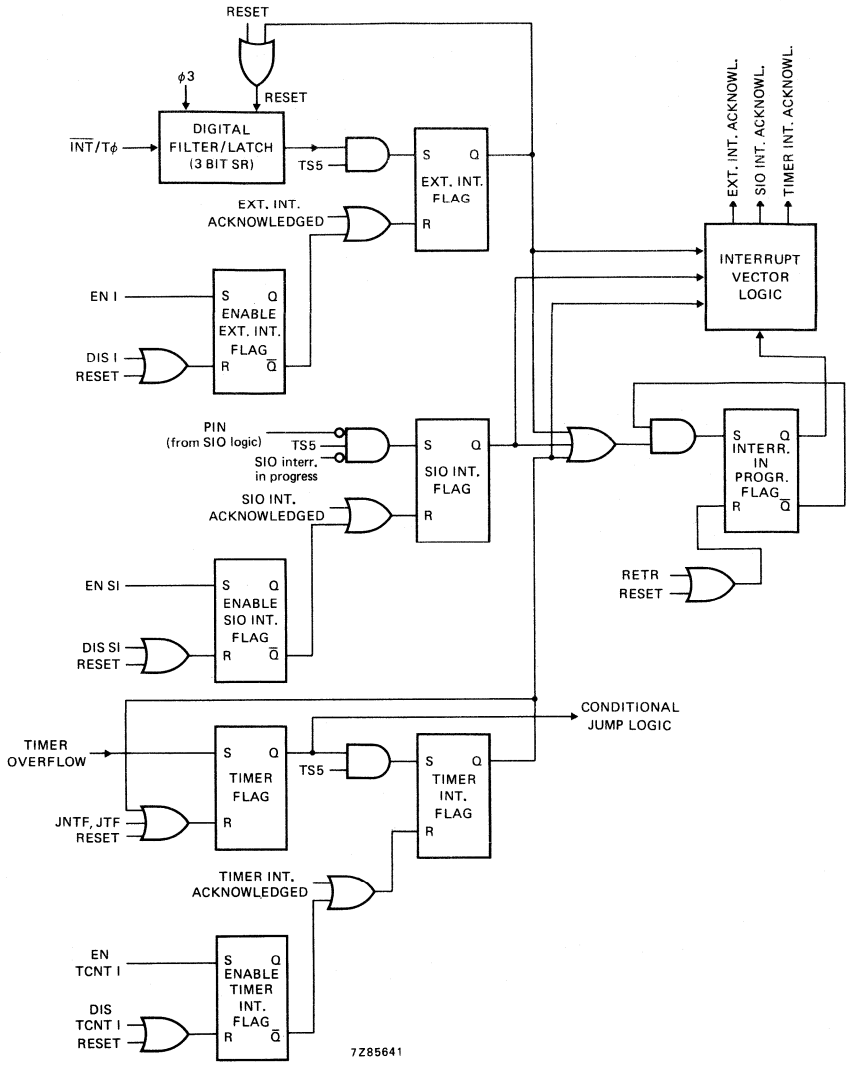


Fig. 10 Interrupt logic.

Note:

1. $\overline{\text{INT}}$/T0 negative edge is always latched in the digital filter/latch.

2. Interrupt is ensured when $\overline{\text{INT}}$/T0 is HIGH for longer than (4 oscillator periods) followed by LOW of longer than (7 oscillator periods).

3. A DIS I instruction within, or immediately following an interrupt service routine clears a pending external interrupt. A DIS TCNT I within an interrupt service routine clears a pending timer/counter interrupt.

4. When the interrupt in progress flag is set, further interrupts are latched but ignored, until RETR is executed.

5. When the timer/counter interrupt flag has been enabled, the JNTF and JTF instructions will always find the flag at 0. This is because the timer flag is only set between TS3 and TS5 of any cycle.

5.4.2 Interrupt program examples

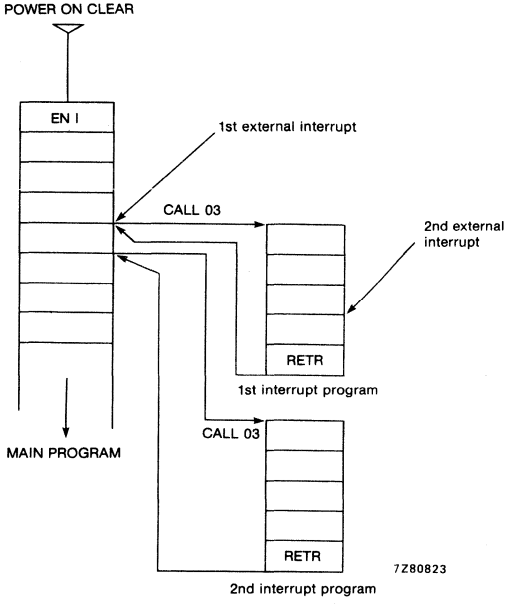


Fig. 11 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

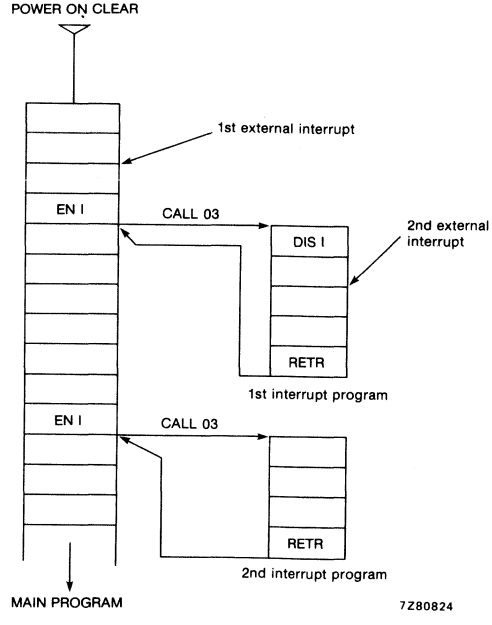


Fig. 12 Second external interr -upt is postponed until enabled again

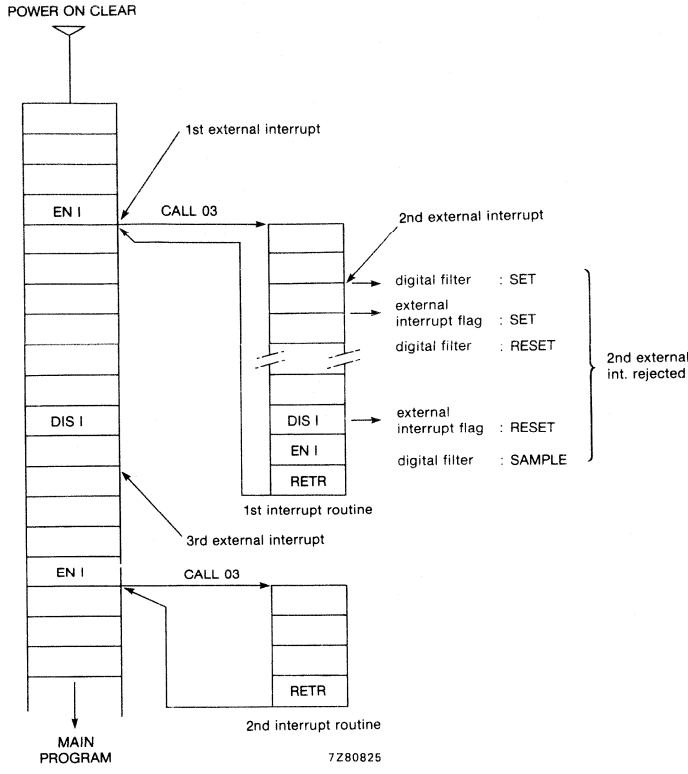


Fig. 13 Clearing of previous external interrupt

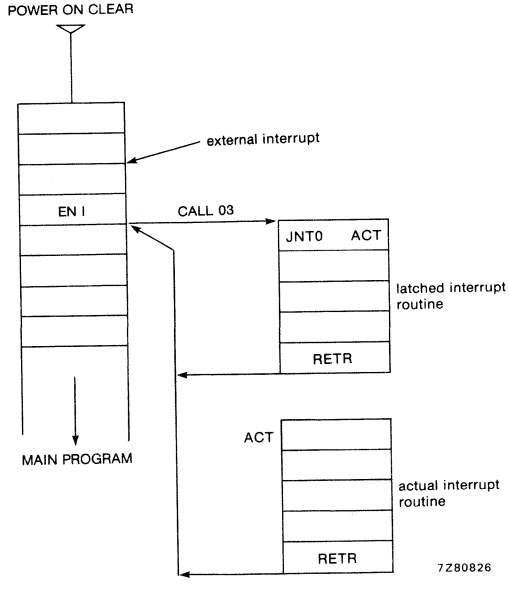


Fig. 14 Detection of previous and actual external interrupt.

To ensure latching in the 'digital filter/latch', the external interrupt should be HIGH for at least 1,33 time slots and LOW for at least 2,33 time slots (1 time slot = 3 clock cycles). The digital filter, therefore acts as a noise supression filter.

The maximum rate at which interrupt edges can be repeated and latched is 4 machine cycles.

Note that $\overline{INT}$/T0 edges are always latched, even when the external interrupts are disabled.

The SIO interrupt circuit is similar to the external interrupt circuit. Here the 'PIN' bit (SIO status word S1) can initiate service routines.

Timer interrupt routines can be started by the overflow of the timer, causing the timer flag to be set. This, when allowed also sets the 'Timer Interrupt Flag', which then in turn resets the timer flag.

5.5 High current outputs

Some pins are provided which can sink higher currents (typical values):

- P23 (serial data), pin 2 — 5 mA at 0,45 V; Open drain
- SCLK, pin 3 — 5 mA at 0,45 V; Open drain
- P10 - P17 — 10 mA at 1,0 V

Note: MAB8401B/WP, P10, P11 — 7 mA at 2,5 V

5.6 Test inputs T1 and T0/$\overline{INT}$

5.6.1 Test input T1

The T1 input line can be used as:

- a test for branch instructions,
- an input for zero voltage cross-over detection,
- an external input to the event counter.

A pull-up resistor can be provided as ROM mask option. This is useful when the input is from a switch or a standard TTL output.

When T1 is used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels respectively. The T1 input has a self-biasing circuit which can detect when an A.C. signal crosses zero (±135 mV sensitivity, max. freq 1 KHz, when coupled with a 1 µF capacitor). The maximum input voltage is 3 V (peak to peak), so maximum voltage ratings are avoided. A LOW-to-HIGH transisition on the T1 input increments the timer/event counter. An overflow of the timer/event counter sets the timer flag.

Zero cross-over detection when used in conjunction with the timer/event counter interrupt, is useful in thyristor control of power equipment. In this case, the port option must be open drain (option 1).

The operation of T1 as an input to the event counter is described in section 5.8.

5.6.2 Test input TO/$\overline{INT}$

The TO/$\overline{INT}$ input may be used as:

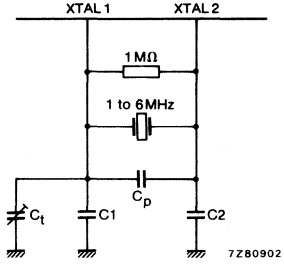
- an external interrupt
- an input level that may be tested by the conditional jump instructions JTO and JNTO.

When used as an external interrupt input, the interrupt signal must be held LOW for a minimum of 7 clock periods and HIGH for a minimum of 4 clock periods. The interrupt is detected on its negative going edge. The interrupt facility is discussed in more detail in chapter 5.4.

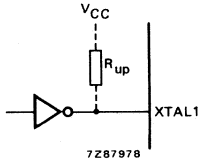
5.7 Oscillator and clock

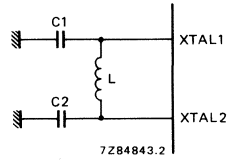
A crystal, ceramic resonator, inductor or resistor connected between XTAL1 and XTAL2 (see Fig. 15) determines the frequency of the internal oscillator. An externally generated clock signal can also be applied to XTAL1 as the frequency reference. A machine cycle consists of 10 states, each state being 3 oscillator periods. A 6 MHz crystal gives a 5,0 us machine cycle.

The MAB84X1 family has dynamic logic; for adequate refreshing the oscillator frequency must be at least 1 MHz (see clock adjustment Chapter 8.1).



1. Crystal – AT-cut
2. Ceramic resonator
   C1 = C2 = 27 pF
   C1 may be trimmed





LC oscillator timing

frequency	C1 = C2	L
3,0 MHz	33 pF	100 μH
4,0 MHz	33 pF	56 μH
4,4 MHz	33 pF	47 μH
5,0 MHz	33 pF	33 μH
6,0 MHz	33 pF	22 μH

Drive XTAL1
Leave XTAL2 open

Driver may be high-speed CMOS or any TTL
t_r, $t_f < 10$ ns

Fig. 15 Connections for clock adjustment.

For a detailed specification on clock adjustment see section 8.0.

5.8 Timer/event counter

An internal 8-bit binary up-counter is provided. This can count external events, modulo-32 machine cycles, or machine cycles directly (Fig. 16). Table 1 shows the instructions that control the counter and prescaler and the functions performed.

Table 1 Timer/event counter control

Function	timer mode modulo-1, module-32*	counter mode
CLEAR	MOV T,A (A) = 0 or RESET	MOV T,A (A) = 0 or RESET
PRESET	MOV T,A	MOV T,A
START	STRT T	STRT CNT
STOP	STOP TCNT or RESET	STOP TCNT or RESET
TEST	JTF/JNTF	JTF/JNTF
READ**	MOV A,T	MOV A,T

* With prescaler select, PS = 0, the timer counts modulo-32 machine cycles, with PS =1 it counts modulo-1 cycles (prescaler not used); prescaler cleared with STRT T, prescaler not readable.
** READ does not disturb the counting process.

When used as a timer, the input to the counter is either the overflow or input from a 5-bit prescaler. When used as an event counter, LOW to HIGH transitions on T1 (pin 13) are counted. The maximum rate at which the counter may be incremented is once every machine cycle (147,7 kHz for a 6,77 µs machine cycle). When the counter overflows, the timer flag is set. The flag can be tested and reset using both the JTF (jump if timer flag = 1) and JNTF instruction. Overflow generates an interrupt to the processor when the timer/event counter interrupt is enabled.

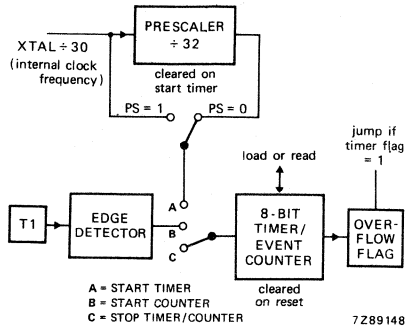


Fig. 16 Timer event counter.

5.9 Program status word

The program status word (PSW) is an 8-bit word (1 byte) in the CPU which stores information about the current status of the microcontroller (Fig.17). The PSW bits are:

bits 0, 1 and 2 - stack pointer bits (SP_0, SP_1, SP_2),
bit 3 - prescaler select (PS); 0 = modulo-32; 1 = modulo-1 (no prescaling),
bit 4 - working register bank select (RBS); 0 = register bank 0; 1 = register bank 1,
bit 5 - not used, set to (1),
bit 6 - auxiliary carry (AC); half carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A,
bit 7 - carry (CY); the carry flag indicates that the previous operation has resulted in an overflow of the accumulator.

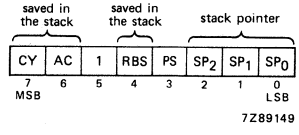


Fig. 17 Program status word.

All bits can be read using the MOV A,PSW instruction. Bits 7 and 6 are set and cleared by CPU operation. Bit 4 can be changed by a SEL RB instruction, bit 3 by the MOV PSW,A instruction, and bits 0, 1 and 2 by the CALL, RET or RETR instructions. Bits 7, 6 and 4 are stored in the program counter stack during subroutines and interrupt calls. These bits are restored in the PSW using the RETR (return and restore) instruction which must be used at the end of everyinterrupt, but which may be used in place of RET at the end of a normal subroutine. The RET instruction has no restore feature and must not be used at the end of an interrupt.

5.10 Program counter

A 13-bit program counter is used so that up to 8 K bytes of ROM can be addressed. Fig. 18 shows the arrangement of the bits. During an interrupt subroutine PC_{11} and PC_{12} are forced to 0. All 13 bits are saved in the stack during CALL and interrupt routines.

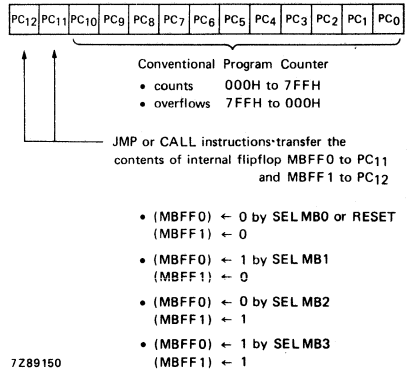


Fig. 18 Program counter.

5.11 Central processing unit

The MAB84X1 family has arithmetic, logic and branch capabilities. The DA A, SWAP A, and XCHD instructions simplify BCD arithmetic and bit handling. The MOVP A,@A instruction permits table look up from the current ROM page.

The conditional branch logic within the processor enables several conditions, to be tested by the user's program, internal and external to the processor. Table 2 lists the conditional jump instructions used to change program sequence. The DJNZ instruction decrements a designated register or data memory location and branches if the contents are not zero. This instruction is useful for looping control. The JMPP@A instruction allows multiway branches to destinations the address of which are pointed to by the accumulator.

Table 2 Conditional branches

test	jump condition	jump instruction
accumulator	0 or non-zero	JZ,JNZ
accumulator bit test	1	JB0 to JB7
carry flag	0 or 1	JNC,JC
timer overflow	0 or 1	JNTF,JTF
test input T0	0 or 1	JNT0,JT0
test input T1	0 or 1	JNT1,JT1
register	non-zero	DJNZ

5.12 Reset

A positive-going signal to the RESET input:

- sets the program counter to zero,
- selects location 0 of memory bank 0, and register bank 0,
- sets the stack pointer to zero (000); pointing to RAM address 8,
- disables the interrupts (external, timer and serial I/O),
- stops the timer/event counter, then sets it to zero,
- sets the timer prescaler to modulo-32,
- resets the timer flag,
- sets all ports to logic '1' (input model),
- sets the serial I/O to slave receiver mode and disables the serial I/O

The external power-on-reset circuit should consist of a 1 μF capacitor connected between V_{CC} and the RESET pin. A diode may be added between the RESET pin and ground to ensure a reset if the supply voltage falls momentarily.

RESET has to be active HIGH for at least 2 machine cycles after the power supply and clock have stabilised.

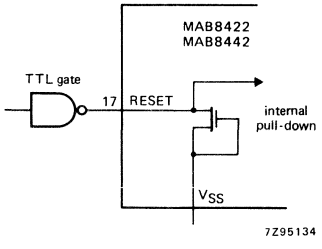


Fig. 19 External reset.

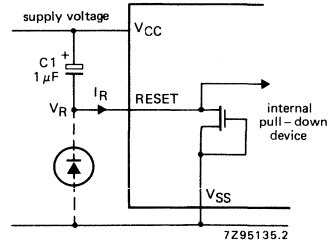


Fig. 20(a) Power-on reset.

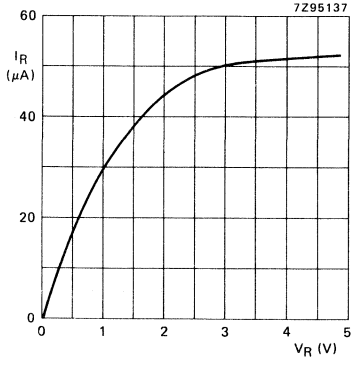


Fig. 20(b) Typical input characteristics.

5.13 Instruction set

The MAB84X1 family instruction set consists of over 80 one and two byte instructions based on the MAB8048 instruction set. New instructions include those for serial I/O operation and memory bank selection. Program code efficiency is high because all RAM locations on a 256-byte page require only a single byte address.

Table 5 shows the instruction set of the MAB84X1 family; Table 6 shows the instruction map. The following symbols and abbreviations are used:

Symbol	description
A	accumulator
AC	auxiliary carry flag
addr	program memory address
Bb	bit designation (b = 0-7)
RBS	register bank select
CLK	clock signal
C	carry (bit CY)
CNT	event counter
D	mnemonic for 4-bit word (nibble)
data	8-bit number or expression
DBF	program memory bank flip-flop
F0,F1	flags 0 and 1
I	interrupt
INT	external interrupt
MB	memory bank
MBFF	memory bank flip-flop
P	mnemonic for 'in-page' operation
PC	program counter
Pp	port designation (p = 0,1,2)
PSW	program status word
RB	register bank
Rr	register designation (r = 0-7)
SP	stack pointer
T	timer
TS8	time slot 8 etc
TF	timer flag
T1	test 1 input
T0	test 0 input
#	immediate data prefix
@	indirect address prefix
(X)	contents of X
((X))	contents of location addressed by X
←	is replaced by
↔	is exchanged with

Table 3 shows the MAB84X1 family instructions (including the five instructions for serial I/O operation) absent from the MAB8048 instruction set.

Table 3

serial I/O	register	control	conditional branch

MOV A,S_n
MOV S_n,A
MOV S_n,#data
EN SI
DIS SI
	DEC@Rr
	DJNZ@Rr,addr
	OUTL P0,A
		SEL MB2
		SEL MB3
			JNTF addr

Table 4 shows the MAB8048 instructions absent from the MAB84X1 family instruction set

Table 4

data moves	flags	branch	control

MOVX A,@R
MOVX @R,A
MOVP3 A,@A
MOVD A,P
MOVD P,A
ANLD P,A
ORLD P,A
	CLR F0
	CPL F0
	CLR F1
	CPL F1
		*JNI addr
		JF0 addr
		JF1 addr
		
		* replaced by JT0, JNT0.
			ENT0 CLK

Difference between the MAB8021, MAB8048 microcontrollers and the MAB84X1 family:

	8021	8048	84X1 family
ROM capacity (bytes)	1 K	1 K	1K,2K,4K,6K ROMless
RAM capacity (bytes)	64	64	64, 128
parallel I/O lines	8+8+4	8+8+8	8+8+4
single inputs	1	3	2
serial I/O	no	no	2-line multi-transmitter
timer	8-bit	8-bit	8-bit
prescaler	mod.32	mod.32	mod.1 & mod.32
machine cycle time (us)	10	2,5	5
for clock (MHz)	3	6	6
instruction set	8021	8048	8048 with omissions; 5 new serial I/O instructions 2 new register inst. 2 new control inst. 1 new branch inst.
interrupts	none	2 ext, tim/evt	3 ext, serial I/O, tim,evt.
N° of pins(DIL)	28	40	28

Table 5 MAB84X1 family instruction set

	mnemonic	opcode (hex.)	bytes/ cycles	description	function		notes
ACCUMULATOR	ADD A, Rr	6*	1/1	Add register contents to A	(A)←(A) + (Rr)	r = 0–7	1
	ADD A, @Rr	60 61	1/1	Add RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) (A)←(A) + ((R1))		1
	ADD A, #data	03 data	2/2	Add immediate data to A	(A)←(A) + data		1
	ADDC A, Rr	7*	1/1	Add carry and register contents to A	(A)←(A) + (Rr) + (C)	r = 0–7	1
	ADDC A, @Rr	70 71	1/1	Add carry and RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) + (C) (A)←(A) + ((R1)) + (C)		1
	ADDC A, #data	13 data	2/2	Add carry and immediate data to A	(A)←(A) + data + (C)		1
	ANL A, Rr	5*	1/1	'AND' Rr with A	(A)←(A) AND (Rr)	r = 0–7	
	ANL A, @Rr	50 51	1/1	'AND' RAM data, addressed by Rr, with A	(A)←(A) AND ((R0)) (A)←(A) AND ((R1))		
	ANL A, #data	53 data	2/2	'AND' immediate data with A	(A)←(A) AND data		
	ORL A, Rr	4*	1/1	'OR' Rr with A	(A)←(A) OR (Rr)	r = 0–7	
	ORL A, @Rr	40 41	1/1	'OR' RAM data, addressed by Rr, with A	(A)←(A) OR ((R0)) (A)←(A) OR ((R1))		
	ORL A, #data	43 data	2/2	'OR' immediate data with A	(A)←(A) OR data		
	XRL A, Rr	D*	1/1	'XOR' Rr with A	(A)←(A) XOR (Rr)	r = 0–7	
	XRL A, @Rr	D0 D1	1/1	'XOR' RAM, addressed by Rr, with A	(A)←(A) XOR ((R0)) (A)←(A) XOR ((R1))		
	XRL A, #data	D3 data	2/2	'XOR' immediate data with A	(A)←(A) XOR data		
	INC A	17	1/1	increment A by 1	(A)←(A) + 1		
	DEC A	07	1/1	decrement A by 1	(A)←(A) – 1		
	CLR A	27	1/1	clear A to zero	(A)←0		
	CPL A	37	1/1	one's complement A	(A)←NOT(A)		
	RL A	E7	1/1	rotate A left	$(A_{n+1})\leftarrow(A_n)$ $(A_0)\leftarrow(A_7)$	n = 0–6	

	mnemonic	opcode (hex.)	bytes/ cycles	description	function	notes
ACCUMULATOR (cont.)	RLC A	F7	1/1	rotate A left through carry	$(A_{n+1})\leftarrow A_n$ n = 0–6 $(A_0)\leftarrow(C)$, $(C)\leftarrow(A_7)$	2
	RR A	77	1/1	rotate A right	$(A_n)\leftarrow(A_{n+1})$ n = 0–6 $(A_7)\leftarrow(A_0)$	
	RRC A	67	1/1	rotate A right through carry	$(A_n)\leftarrow(A_{n+1})$ n = 0–6 $(A_7)\leftarrow(C)$, $(C)\leftarrow(A_0)$	2
	DA A	57	1/1	decimal adjust A		2
	SWAP A	47	1/1	swap nibbles of A	$(A_{4-7})\leftrightarrow(A_{0-3})$	
DATA MOVES	MOV A, Rr	F*	1/1	move register contents to A	(A)←(Rr) r = 0–7	
	MOV A, @Rr	F0 F1	1/1	move RAM data, addressed by Rr, to A	(A)←((R0)) (A)←((R1))	
	MOV A, #data	23 data	2/2	move immediate data to A	(A)←data	
	MOV Rr, A	A*	1/1	move accumulator contents to register	(Rr)←(A) r = 0–7	
	MOV @Rr, A	A0 A1	1/1	move accumulator contents to RAM location addressed by Rr	((R0))←(A) ((R1))←(A)	
	MOV Rr, #data	B* data	2/2	move immediate data to Rr	(Rr)←data	
	MOV @Rr, #data	B0 data B1 data	2/2	move immediate data to RAM location addressed by Rr	((R0))←data ((R1))←data	
	XCH A, Rr	2*	1/1	exchange accumulator contents with Rr	(A)↔(Rr) r = 0–7	
	XCH A, @Rr	20 21	1/1	exchange accumulator contents with RAM data addressed by Rr	(A)↔((R0)) (A)↔((R1))	
	XCHD A, @Rr	30 31	1/1	exchange lower nibbles of A and RAM data addressed by Rr	$(A_{0-3})\leftrightarrow((R0_{0-3}))$ $(A_{0-3})\leftrightarrow((R1_{0-3}))$	
	MOV A, PSW	C7	1/1	move PSW contents to accumulator	(A)←(PSW)	
	MOV PSW, A	D7	1/1	move accumulator bit 3 to PSW_3	$(PSW_3)\leftarrow(A_3)$	3
	MOVP A, @A	A3	1/2	move indirectly addressed data in current page to A	$(PC_{0-7})\leftarrow(A)$, $(A)\leftarrow((PC))$	

FLAGS	CLR C	97	1/1	clear carry bit	(C)←0	2
	CPL C	A7	1/1	complement carry bit	(C)←NOT(C)	2
REGISTER	INC Rr	1*	1/1	increment register by 1	(Rr)←(Rr) + 1 r = 0–7	
	INC @Rr	10 11	1/1	increment RAM data, addressed by Rr, by 1	((R0))←((R0)) + 1 ((R1))←((R1)) + 1	
	DEC Rr	C*	1/1	decrement register by 1	(Rr)←(Rr) – 1 r = 0–7	
	DEC @Rr	C0 C1	1/1	decrement RAM data, addressed by Rr, by 1	((R0))←((R0)) – 1 ((R1))←((R1)) – 1	
BRANCH	JMP addr	• 4 address	2/2	unconditional jump within a 2K bank	(PC_{8-10})←$addr_{8-10}$ (PC_{0-7})←$addr_{0-7}$ (PC_{11-12})←MBFF 0–1	
	JMPP @A	B3	1/2	indirect jump within a page	(PC_{0-7})←((A))	
	DJNZ Rr, addr	E* address	2/2	decrement Rr by 1 and jump if not zero to addr	(Rr)←(Rr) – 1 r = 0–7 if (Rr) not zero (PC_{0-7})←addr	
	DJNZ @Rr, addr	E0	2/2	decrement RAM data, addressed by Rr, by 1 and jump if not zero to addr	((R0))←((R0)) – 1 if ((R0)) not zero (PC_{0-7})←addr	
		E1			((R1))←((R1)) – 1 if ((R1)) not zero (PC_{0-7})←addr	
	JBb addr	▲ 2 address	2/2	jump to addr if Acc. bit b = 1	if b = 1: (PC_{0-7})←addr b = 0–7	
	JC addr	F6 address	2/2	jump to addr if C = 1	if C = 1: (PC_{0-7})←addr	
	JNC addr	E6 address	2/2	jump to addr if C = 0	if C = 0: (PC_{0-7})←addr	
	JZ addr	C6 address	2/2	jump to addr if A = 0	if A = 0: (PC_{0-7})←addr	
	JNZ addr	96 address	2/2	jump to addr if A is NOT zero	if A ≠ 0: (PC_{0-7})←addr	
	JT0 addr	36 address	2/2	jump to addr if T0 = 1	if T0 = 1: (PC_{0-7})←addr	
	JNT0 addr	26 address	2/2	jump to addr if T0 = 0	if T0 = 0: (PC_{0-7})←addr	
	JT1 addr	56 address	2/2	jump to addr if T1 = 1	if T1 = 1: (PC_{0-7})←addr	
	JNT1 addr	46 address	2/2	jump to addr if T1 = 0	if T1 = 0: (PC_{0-7})←addr	
	JTF addr	16 address	2/2	jump to addr if Timer Flag = 1	if TF = 1: (PC_{0-7})←addr	4
	JNTF addr	06 address	2/2	jump to addr if Timer Flag = 0	if TF = 0: (PC_{0-7})←addr	

	mnemonic	opcode (hex.)	bytes/ cycles	description	function	notes
TIMER/EVENT COUNTER	MOV A, T	42	1/1	move timer/event counter contents to accumulator	(A)←(T)	
	MOV T, A	62	1/1	move accumulator contents to timer/event counter	(T)←(A)	
	STRT CNT	45	1/1	start event counter		
	STRT T	55	1/1	start timer		
	STOP TCNT	65	1/1	stop timer/event counter		
	EN TCNTI	25	1/1	enable timer/event counter interrupt		
	DIS TCNTI	35	1/1	disable timer/event counter interrupt		
CONTROL	EN I	05	1/1	enable external interrupt		
	DIS I	15	1/1	disable external interrupt		
	SEL RB0	C5	1/1	select register bank 0	(RBS)←0	5
	SEL RB1	D5	1/1	select register bank 1	(RBS)←1	5
	SEL MB0	E5	1/1	select program memory bank 0	(MBFF0)←0, (MBFF1)←0	
	SEL MB1	F5	1/1	select program memory bank 1	(MBFF0)←1, (MBFF1)←0	
	SEL MB2	A5	1/1	select program memory bank 2	(MBFF0)←0, (MBFF1)←1	
	SEL MB3	B5	1/1	select program memory bank 3	(MBFF0)←1, (MBFF1)←1	
SUBROUTINE	CALL addr	▲ 4 address	2/2	jump to subroutine	((SP))←(PC), ($PSW_{4, 6, 7}$) (SP)←(SP) + 1 (PC_{9-10})←$addr_{8-10}$ (PC_{0-7})←$addr_{0-7}$ (PC_{11-12})←MBFF 0–1	6
	RET	83	1/2	return from subroutine	(SP)←(SP) – 1 (PC)←((SP))	6
	RETR	93	1/2	return from interrupt and restore bits 4, 6, 7 of PSW	(SP)←(SP) – 1 ($PSW_{4, 6, 7}$) + (PC)←((SP))	6

PARALLEL INPUT/OUTPUT	IN A, Pp	08 09 0A	1/2	input port p data to accumulator	(A)←(P0) (A)←(P1) (A)←(P2)	7
	OUTL Pp, A	38 39 3A	1/2	output accumulator data to port p	(P0)←(A) (P1)←(A) (P2)←(A)	
	ANL Pp, #data	98 data 99 data 9A data	2/2	AND port p data with immediate data	(P0)←(P0) AND data (P1)←(P1) AND data (P2)←(P2) AND data	
	ORL Pp, #data	88 data 89 data 8A data	2/2	OR port p data with immediate data	(P0)←(P0) OR data (P1)←(P1) OR data (P2)←(P2) OR data	
	OUTL PO,A	90	1/2	Output accumulator data to port ϕ	(P0)←(A)	
SERIAL INPUT/OUTPUT	MOV A, S_n	0C 0D	1/2	move serial I/O register contents to accumulator	(A)←(S0) (A)←(S1)	8
	MOV S_n, A	3C 3D 3E	1/2	move accumulator contents to serial I/O register	(S0)←(A) (S1)←(A) (S2)←(A)	
	MOV S_n, #data	9C data 9D data 9E data	2/2	move immeidate date to serial I/O register	(S0)←data (S1)←data (S2)←data	
	EN SI	85	1/1	enable serial I/O interrupt		
	DIS SI	95	1/1	disable serial I/O interrupt		
	NOP	00	1/1	no operation		

Notes to Table 3.

1. PSW CY, AC affected
2. PSW CY affected
3. PSW PS affected
4. Execution of JTF and JNTF instructions resets the Timer Flag (TF).
5. PSW RBS affected
6. PSW SP_0, SP_1, SP_2 affected
7. (A) = 1111 P23, P22, P21, P20.
8. (S1) has a different meaning for read and write operation, see serial I/O interface.

* : 8, 9, A, B, C, D, E, F
● : 0, 2, 4, 6, 8, A, C, E
▲ : 1, 3, 5, 7, 9, B, D, F

MAB84X1 family instruction set

first hexadecimal character of opcode

second hexadecimal character of opcode

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP			ADD A,#data	JMP page 0	EN I	JNTF addr	DEC A	IN A,Pp 0 1 2				MOV A,Sn 0 1			
1	INC @Rr 0 1	JBO addr	ADDC A,#data	CALL page 0	DIS I	JTF addr	INC A	INC Rr 0 1 2 3 4 5 6 7								
2	XCH A,@Rr 0 1		MOV A,#data	JMP page 1	EN TCNTI	JNTO addr	CLR A	XCH A,Rr 0 1 2 3 4 5 6 7								
3	XCHD A,@Rr 0 1	JB1 addr		CALL page 1	DIS TCNTI	JTO addr	CPL A	OUTL Pp,A 0 1 2					MOV Sn,A 0 1		2	
4	ORL A,@Rr 0 1	MOV A,T	ORL A,#data	JMP page 2	STRT CNT	JNT1 addr	SWAP A	ORL A,Rr 0 1 2 3 4 5 6 7								
5	ANL A,@Rr 0 1	JB2 addr	ANL A,#data	CALL page 2	STRT T	JT1 addr	DA A	ANL A,Rr 0 1 2 3 4 5 6 7								
6	ADD A,@Rr 0 1	MOV T,A		JMP page 3	STOP TCNT		RRC A	ADD A,Rr 0 1 2 3 4 5 6 7								
7	ADDC A,@Rr 0 1	JB3 addr		CALL page 3			RR A	ADDC A,Rr 0 1 2 3 4 5 6 7								
8			RET	JMP page 4	EN SI			ORL Pp,#data 0 1 2								
9	OUTL P0,A	JB4 addr	RETR	CALL page 4	DIS SI	JNZ addr	CLR C	ANL Pp,#data 0 1 2					MOV Sn,#data 0 1 2			
A	MOV @Rr,A 0 1		MOVP A,@A	JMP page 5	SEL MB2		CPL C	MOV Rr,A 0 1 2 3 4 5 6 7								
B	MOV @Rr,#data 0 1	JB5 addr	JMPP @A	CALL page 5	SEL MB3			MOV Rr,#data 0 1 2 3 4 5 6 7								
C	DEC @Rr 0 1			JMP page 6	SEL RB0	JZ addr	MOV A,PSW	DEC Rr 0 1 2 3 4 5 6 7								
D	XRL A,@Rr 0 1	JB6 addr	XRL A,#data	CALL page 6	SEL RB1		MOV PSW,A	XRL A,Rr 0 1 2 3 4 5 6 7								
E	DJNZ @Rr,addr 0 1			JMP page 7	SEL MB0	JNC addr	RL A	DJNZ Rr, addr 0 1 2 3 4 5 6 7								
F	MOV A,@Rr 0 1	JB7 addr		CALL page 7	SEL MB1	JC addr	RLC A	MOV A,Rr 0 1 2 3 4 5 6 7								

6.0 MAB8401WP $\overline{\text{HALT}}$ function

The $\overline{\text{HALT}}$ facility permits single stepping through the user program. In each $\overline{\text{HALT}}$ state the consecutive program counter contents are placed on the address bus. Hence, the user can follow the program instruction by instruction. Fig. 21 shows the timing cycle of the $\overline{\text{HALT}}$ function. When stopped, the address of the next instruction to be fetched is available on the address bus.

The MAB8401WP operates in a $\overline{\text{HALT}}$ mode as follows:

1. The processor is requested to stop by applying a LOW level on $\overline{\text{HALT}}$. The processor senses this input continuously: it must be active LOW during at least TS4-TS9.

2. If the $\overline{\text{HALT}}$ line is found active, the processor finishes its current instruction and inserts a NOP opcode in the instruction register. The program counter and timer/counter are no longer updated. The address of the next instruction is present on the address-bus.

3. Each following cycle that the $\overline{\text{HALT}}$ line is sampled active, a NOP is inserted in the instruction register (IR).

4. The processor continues if the $\overline{\text{HALT}}$ input is found HIGH during TS4-TS9, i.e. instead of NOP, the next instruction is fetched and inserted in IR.

7.0 GENERATION OF CLOCK SIGNALS

7.1 General PCB layout

Generally all clock circuits should be layed-out with great care. Since they are analogue circuits in a digital environment they are susceptible to noise from inductive and capacitive coupling. Also, high voltage switching circuits and other components generating high electromagnetic fields, should be located away from the clock area.

Ideally, the clock circuit should be surrounded by a grounded 'guard ring' shared by all the components. If the board is two-sided, the reverse side of the board should have a grounded plate (connected to V_{SS}) over the area of the clock circuit.

7.2 Crystal clock generator

Outline of MAB84X1 family clock inputs.

The on board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 6 MHz. A crystal or inductor connected between XTAL 1 (pin 15) and XTAL 2 (pin 16) provides the feedback and phase shift required for oscillation. For accurate frequency reference and maximum processor speed a crystal should be used, but for frequencies between 3 to 5 MHz where accuracy is not important an LC oscillator circuit may be implemented.

Note : Because the MAB84X1 family has dynamic logic, the oscillator frequency must be at least 1 MHz to provide adequate refreshing.

The recommended clock circuit for frequencies of 1 to 6 MHz using a crystal oscillator is shown in fig. 21.

C_t is optional

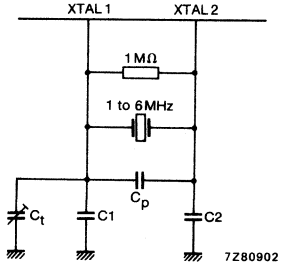


Fig. 21. Circuit diagram.

Approximate component values :

C1 = C2 = 27 pF

C_p = ⩽ 6,75 pF (parasitic capacitance)

XTAL (AT cut) = The fundamental frequency crystal (determines the operating frequency of the device)

R_S = ⩽ 60 Ω (resonator resistance) preferred for highest stability although higher values may be used where stability is not critical

C_t = Trimming capacitor.

Note: If needed, trimming to lower frequencies is accomplished by placing a capacitive trimmer on the high impedance end of the oscillator, in parallel with C1. Trimming in the direction of higher frequencies may be acheived by inserting a 10 pF capacitor in series with the crystal.

The stability of the clock circuit is greatly dependent upon the external components, the microcontroller has little influence.

Causes of instability may arise from :

1) Thermal influence upon quartz crystal and dependent components.

2) Tolerance of trimming components.

3) Power dissipation of crystal; for greatest stability the series resistance (R_S) should be as low as possible as the power dissipation is directly proportional. i.e $R_S \leqslant 60\,\Omega$, higher values of R_S are possible but result in lower stability of the clock frequency.

4) Ageing (long term parameter variation), note: such parameter variation is due to long term changes in the crystal unit and is usually expressed in fractional parts per unit time. Pre-application appraisement may be useful.

When designing crystal circuits where optimum temperature stability is desired, the values of C1 and C2 should be taken from the following block;

CSC 30 pF

KYOCERA

KBR - 3,58 M (3,58 MHz)
KBR - 4,0 M (4,0 MHz)
KBR - 5,0 M (5,0 MHz)
KBR - 6,0 M (6,0 MHz)

7.3 Clock generation using ceramic resonators

When implementing ceramic resonators, the circuit used is the same as that for crystal oscillation (see Fig 21.)

The following ceramic resonators are recommended:

MURATA

CSA 300 MT (3,0 MHz)
CSA 400 MT (4,0 MHz)
CSA 600 MT (6,0 MHz)

7.4 Clock generation using LC oscillators

For lower cost and where accurate timing is not critical an inductive circuit may be constructed.

The recommended circuit using an LC network is shown in Fig. 22.

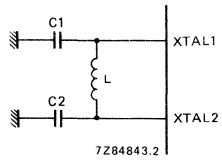


Fig. 22 Circuit diagram for LC oscillator.

For clock generation using LC networks, the values of L and C for required frequency ranges are given in table form below.

clock frequency [MHz]	L [µH]	C1 = C2 [pF]
3,0	100	33
4,0	56	33
4,4	47	33
5,0	33	33
6,0	22	33

7.5 Single crystal oscillator serving multiple devices

It may often be desired to run several devices off a single source of clock signal. For this case two general possibilities exist.

1) Use of a central oscillator circuit that services all the required devices.

2) Using the oscillator circuit of an MAB84XX device, the signal may be amplified and distributed to dependent devices.

7.5.1 Central oscillator

Fig. 23 shows a suitable circuit for a central CMOS oscillator.

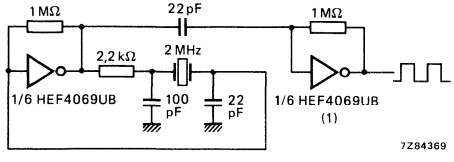


Fig. 23. A CMOS oscillator.

The HEF4069UB (seen in Fig. 23) must be used as an inverter as buffered devices do not readily oscillate.

7.5.2 One MAB84X1 driving another separate MAB84X1

Fig. 24 shows the circuit for clocking two MAB84X1 controllers.

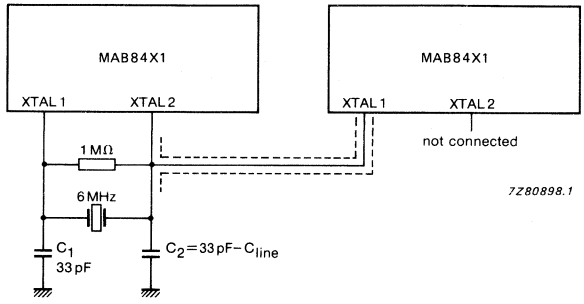


Fig. 24. Clocking two MAB84X1 devices.

Two MAB84X1 devices may be clocked using a single crystal source from either of the devices. The second 84X1 is fed with the clock pulse to its XTAL 1 input sourced by the originating XTAL 2 output. The originating capacitor C2 should be modified to suit the additional capacitance of the driving line.

In order to make the connection of the second microcontroller as uncritical as possible, it should be placed (at maximum) only a few centimetres away from the source microcontroller.

7.5.3 Buffered on-chip oscillation

Figures 25, 26, 27 and 28 illustrate some practical circuits:

1) Using TTL devices

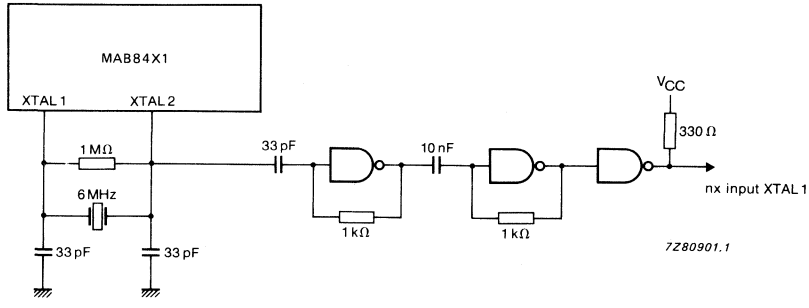


Fig 25. TTL linear amp and TTL buffer (SN74 series).

Maximum load at 6 MHz = 12 MAB84X1 devices
load capacitance ` 500 pF

2) Using CMOS devices

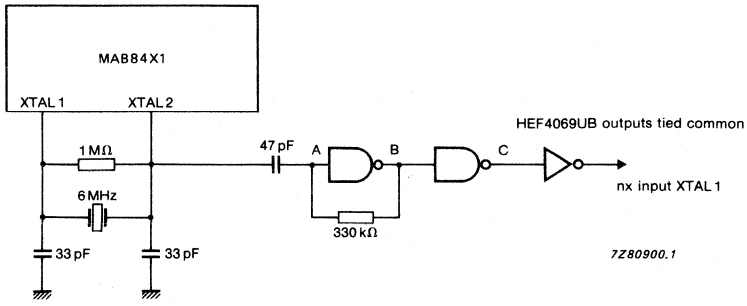


Fig 26. CMOS amp and CMOS buffer (HEF4069UB)

Maximum load at 6 MHz = 3 MAB84X1 devices
load capacitance = 140 pF
wiring capacitance generated at points A, B and C is less than 8 pF.

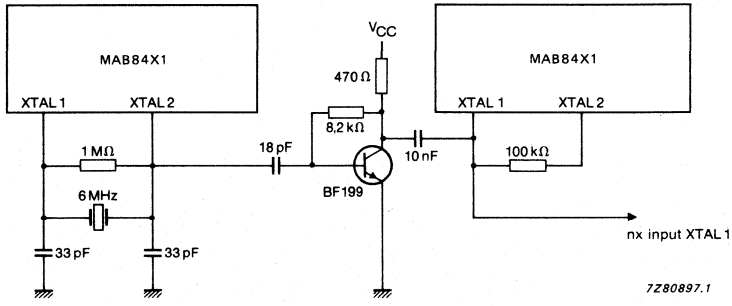


Fig. 27 Bipolar amplifier stage offers good pulse reproduction and D.C. isolation for subsequent XTAL1 inputs. (Transistor BF199)

At 6 MHz : Maximum series load with 100 kΩ resistance = 1 MAB84X1
Maximum parallel load across 100 kΩ resistance = 3 MAB84X1
Total load capacitance ⩽ 160 pF.

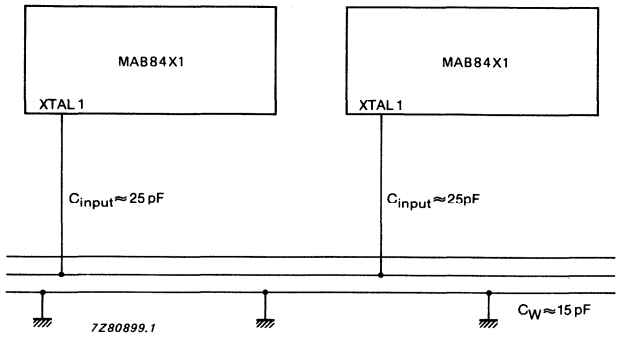


Fig. 28 Capacitive load for each MAB84X1 = 40 pF (approximate value)

C_{input} = input capacitance

C_W = wiring capacitance

8.0 TIMING

8.1 Clock adjustment

Quartz crystal adjustment

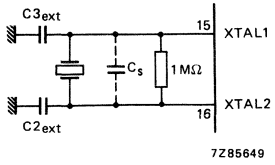


Fig. 29 Typical crystal oscillator.

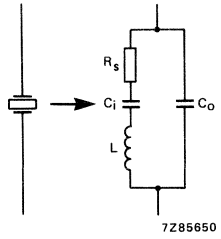


Fig. 30 Quartz crystal equivalent.

Quartz crystal parameter

Cut, resonance frequency: AT, 1 to 6 MHz (fundamental)

Dynamic capacitance (C1): $\leqslant$ 10 pF

Parallel capacitance: see below (C_0+C_S)

Series resistance: $\leqslant$ 60 Ω

Oscillator circuit

		min.	max.
External + wiring capacitance on XTAL1:	$C3_{ext}$	27	70 pF
External + wiring capacitance on XTAL2:	$C2_{ext}$	27	70 pF
Capacitance in parallel with crystal:	C_0+C_S	-	$2C_L$

Where:

LC adjustment

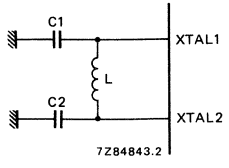


Fig. 31 Typical L C oscillator.

LC oscillator timing

frequency	C1 = C2	L
3,0 MHz	33 pF	100 μH
4,0 MHz	33 pF	56 μH
4,4 MHz	33 pF	47 μH
5,0 MHz	33 pF	33 μH
6,0 MHz	33 pF	22 μH

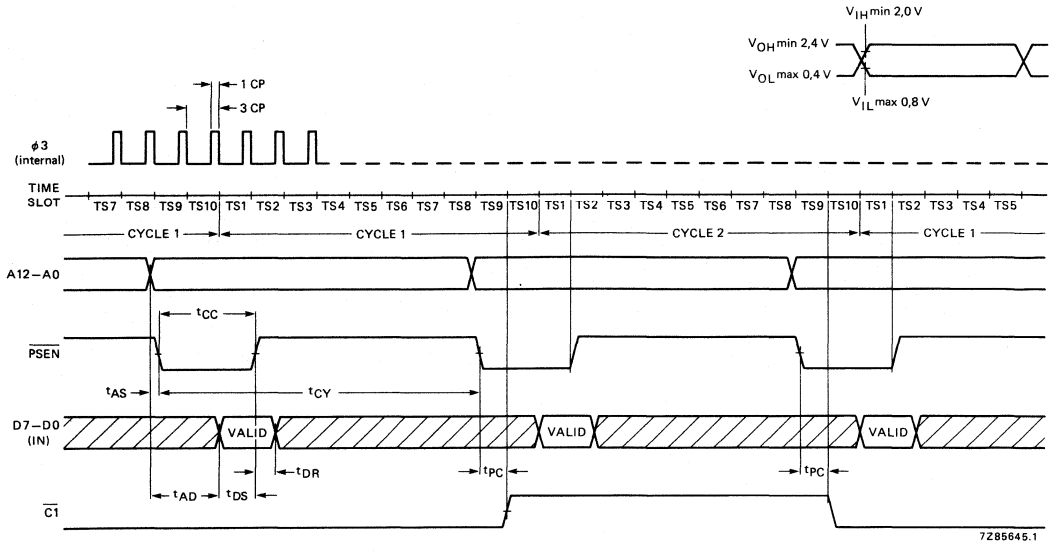


Fig. 32 Memory access timing plus I/O voltage parameters.

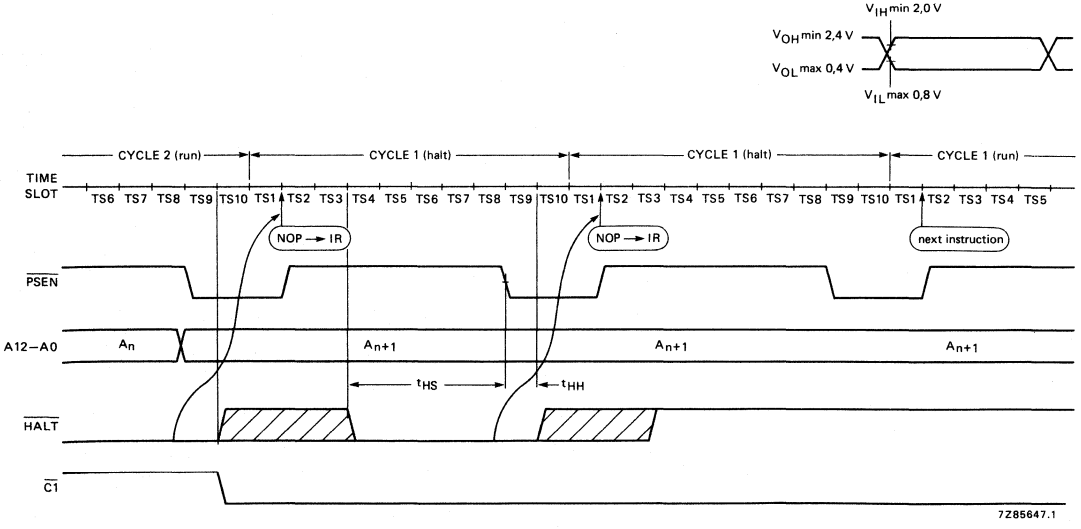


Fig. 33 $\overline{C1}$ and $\overline{HALT}$ timing plus I/O voltage parameters.

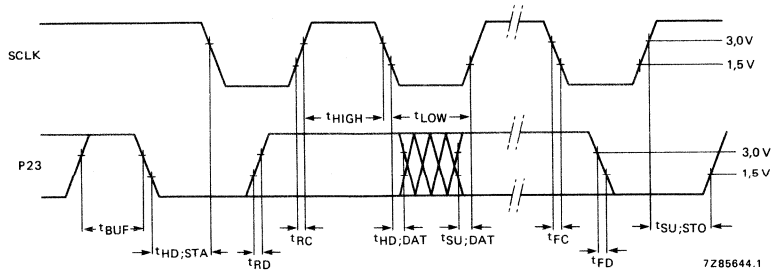


Fig. 34 Input timing.

8.2 Timing requirements for the P23 and SCLK input signals:

t_{BUF}	$\geqslant 14t_{XTAL}$
$t_{HD;STA}$	$\geqslant 14t_{XTAL}$
t_{HIGH}	$\geqslant 17t_{XTAL}$
t_{LOW}	$\geqslant 17t_{XTAL}$
$t_{SU;STO}$	$\geqslant 14t_{XTAL}$
$t_{HD;DAT}$	>0
$t_{HD;DAT}$	$\geqslant 250$ ns
t_{RD}	$\leqslant 1$ 7s
t_{RC}	$\leqslant 1$ 7s
t_{FD}	$\leqslant 1$ 7s
t_{FC}	$\leqslant 1$ 7s

t_{XTAL} = one period of the XTAL input frequency (f_{XTAL}) = 226 ns for f_{XTAL} = 4,43 MHz.

These figures apply to all modes. The difference in oscillator frequency between receiver and transmitter should not be greater than $\pm$ 0.5 MHz.

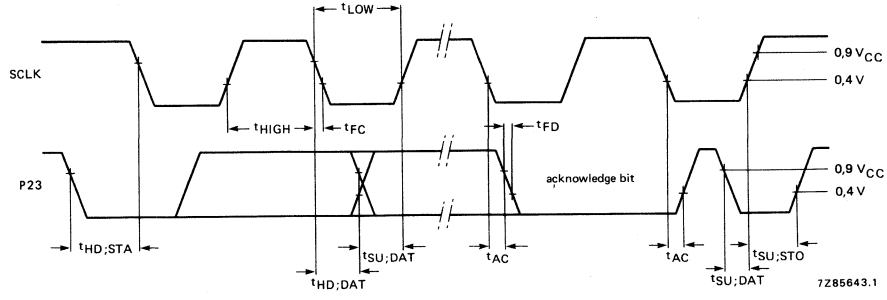


Fig. 35 Output timing.

8.3 Timing requirements for the P23 and SCLK output signals:

$t_{HD;STA}$		$1/2(DF + 9)\ t_{XTAL}$
t_{HIGH}		$1/2(DF + 3)\ t_{XTAL}$
t_{LOW}		$1/2(DF - 3)\ t_{XTAL}$
$t_{SU;STO}$		$1/2(DF - 3)\ t_{XTAL}$
$t_{HD;DAT}$	(only for SLAVE TRANSMITTER)	$\geqslant 6t_{XTAL}$ $\leqslant 12t_{XTAL}$
$t_{SU;DAT}$	(only for MASTER TRANSMITTER)	$6t_{XTAL}$
t_{AC}		$\geqslant 6t_{XTAL}$ $\leqslant 12t_{XTAL}$
t_{FD}, t_{FC}		$\leqslant 100ns$ at C_b = 400 pF *

t_{XTAL} = one period of the XTAL input frequency(f_{XTAL}) = 226 ns
for f_{XTAL} = 4,43 MHz.

DF = divisor (see Table 3 in serial I/O section MAB84X1 data sheet)

* C_b is maximum bus capacitance for each line.

5. The MAB8422/42 microcontroller family

CONTENTS – THE 8422/42 MICROCONTROLLER FAMILY

		page
1.0	GENERAL DESCRIPTION	5-4
2.0	FEATURES	5-6
3.0	PACKAGE OUTLINE	5-6
3.1	Pinning	5-7
4.0	THE MEMORY	5-8
4.1	Program memory (ROM)	5-8
4.2	Data memory (RAM)	5-9
5.0	INPUT/OUTPUT FACILITIES	5-12
5.1	Parallel port mode	5-12
5.2	Serial I/O mode	5-15
5.2.1	Hardware and software functions	5-15
5.2.2	Description of the signals to and from port P10 to P17 in the serial I/O mode	5-17
5.2.3	Stretch and edge detection	5-20
5.3	Test input T1	5-20
5.4	Test input T0	5-20
5.5	High current outputs	5-21
6.0	INTERRUPT	5-22
6.1	Interrupt logic	5-22
6.2	Interrupt program examples	5-25
7.0	CENTRAL PROCESSING UNIT	5-27
8.0	PROGRAM STATUS WORD	5-28
9.0	PROGRAM COUNTER	5-29
10.0	OSCILLATOR AND TIMING	5-30
11.0	TIMER/EVENT COUNTER	5-31
12.0	RESET	5-32
13.0	DEVELOPMENT SUPPORT	5-33
14.0	I^2C-BUS SOFTWARE EXAMPLES	5-34
14.1	Slave transmitter/receiver subroutine	5-34
14.2	Multi-master subroutine	5-45
15.0	INSTRUCTION SET	5-62

1.0 GENERAL DESCRIPTION

In systems where control is the main function of a microcontroller, the system cost can be reduced by using microcontrollers with dedicated hardware functions, memory capacity and a number of I/O lines. Low cost is often the key to incorporating microcontrollers into systems intended for high volume markets. This is the philosophy behind the MAB8422/8442 microcontroller. Being a single-chip microcontroller it offers the advantages of short system development times, fast assembly and high reliability because few external connections are necessary.

The MAB8422/8442 is a 20-pin, single-chip 8-bit microcontroller that has been developed from the 28-pin MAB8421/8441 microcontrollers. The versions are:

- MAB8422 - 2 K ROM/64 RAM bytes
- MAB8442 - 4 K ROM/128 RAM bytes

Each version has 15 I/O port lines comprising one 8-bit parallel port (P0), one 2-bit parallel port (P10 and P11 that are shared with the serial I/O lines SDA and SCL), one 3-bit parallel port (P20 - P22) and two input lines (INT/T0 and T1).

The serial I/O interface is I^2C compatible and therefore the MAB8422/8442 can operate as a slave or a master in single and multi-master systems. Conversion from parallel to serial data when transmitting, and vice versa when receiving, is done mainly in software. There is a minimum of hardware for the serial I/O implemented. This hardware is controlled by the status of the SDA and SCL lines and can be read or written under software control.

The development of MAB8422/8442 software is supported by prototyping boards and development systems available now.

Fig. 1.1 shows the block diagram of the MAB8422/8442. Both microcontrollers are in 20 pin DIL packages. The pin configuration is given in Fig. 3.2 with the designation of the pins in Table 1.

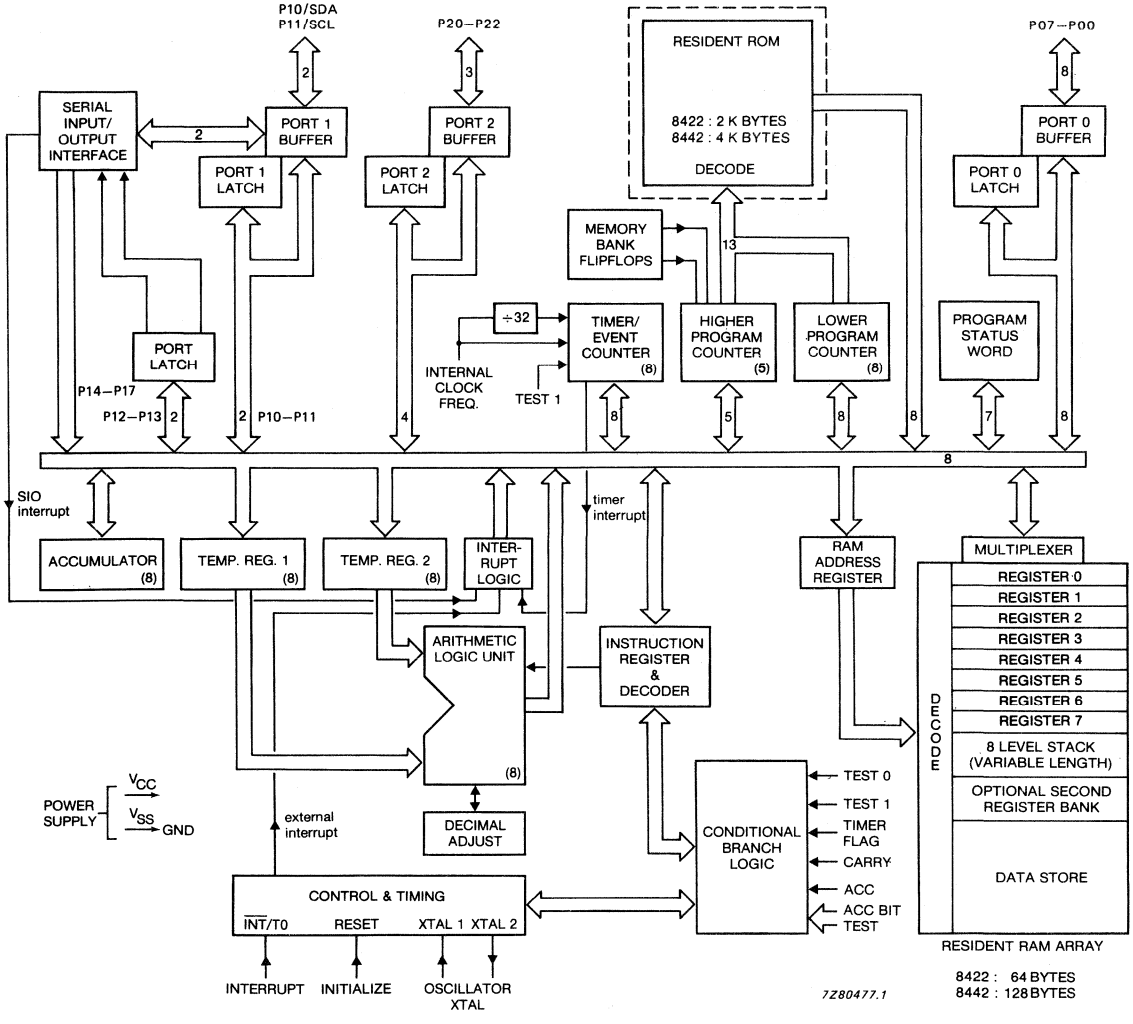


Fig. 1.1 Block diagram of the MAB8422/8442.

2.0 FEATURES

- 8-bit: CPU, ROM, RAM and I/O.
- 20 pin package
- MAB8422 : 2 K ROM/64 RAM bytes
- MAB8442 : 4 K ROM/128 RAM bytes
- 13 quasi-bidirectional I/O port lines
- Two testable inputs T1 and $\overline{INT}$/T0
- High current output on P0
- One external interrupt line combined with the testable input line $\overline{INT}$/T0
- Single-level interrupts: external, timer/event counter and serial I/O
- I^2C-compatible serial I/O that can be used in single or multi-master systems (serial I/O data and clock via P10 and P11 port lines, respectively)
- 8-bit programmable timer/event counter
- Switchable 5-bit prescaler in timer mode
- Internal oscillator, frequency determined by inductor or crystal, ceramic resonator or external source
- Over 80 instructions (based on MAB8048)
- All instructions 1 or 2 cycles, cycle time dependent on oscillator frequency
- Single power supply

3.0 PACKAGE OUTLINE

MAB/MAF84X2, MAF84AX2: 20-lead DIL; plastic (SOT-146).

3.1 Pinning

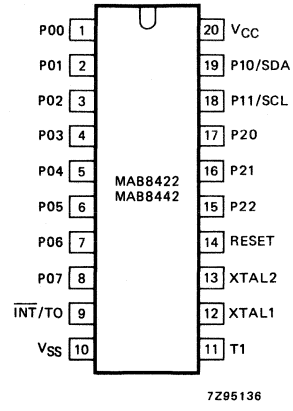


Fig. 3.1 MAB8422/8442 Pinning diagram

Table 1. The pin designation of the MAB8422/8442

Designation	Pin number	Function
P00 - P07	1 - 8	8-bit quasi-bidirectional I/O port (Port 0 - high current output).
$\overline{INT}$/T0	9	External interrupt input (sensitive to a negative going edge) and/or input, testable using the JT0 and JNT0 instructions.
V_{SS}	10	Ground
T1	11	Input pin, testable using the JT1 and JNT1 instructions. It can be designated as event counter input using the STRT CNT instruction. It also allows zero cross-over sensing of slowly moving A.C. inputs.
XTAL1	12	Connection to timing component that determines the frequency of the internal oscillator. It is also the input for an external clock source.
XTAL2	13	Connection to the other side of the timing component.
RESET	14	Input to initialize the processor (active HIGH).
P10/SDA	19	Quasi-bidirectional port in parallel port mode. Serial data I/O in serial I/O mode.
P11/SCL	18	Quasi-bidirectional port in parallel port mode. Serial clock in serial I/O mode.
P20 - P22	17 - 15	Quasi-bidirectional port.
V_{CC}	20	Power supply.

4.0 THE MEMORY

4.1 Program memory (ROM)

The mask-programmed ROM contains 2048 bytes for the MAB8422 and 4096 bytes for the MAB8442. The program memory in each family member is addressed by the program counter. The program counter is 13-bits wide so that up to 8 K-bytes of program memory can be addressed. With the MAB8422/8442 offering a choice of ROM capacity to suit a variety of applications, external ROM expansion is not required. The program memory map is shown in Fig. 4.1.

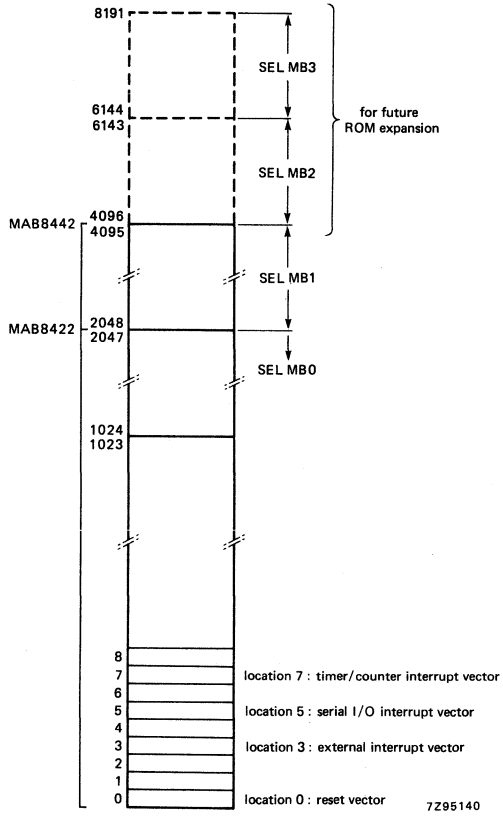


Fig. 4.1 The program memory map.

Four program memory locations are of special importance:

location 0 - contains the first instruction to be executed after the processor is initialized (RESET),

location 3 - contains the first byte of an external interrupt service routine,

location 5 - contains the first byte of a serial I/O interrupt service routine,

location 7 - contains the first byte of a timer/event counter interrupt service routine.

The program memory is arranged in banks of 2 K bytes that are selected by SEL MB instructions, and each bank is divided into 256-byte pages. Only the unconditional jump instructions (JMP and CALL) can cause jumps across page boundaries. Memory bank boundaries can only be crossed by using these unconditional jump instructions after the appropriate memory bank has been selected. A CALL instruction can transfer control to a subroutine on any page; RET and RETR instructions can transfer control from a subroutine back to the main program.

Note; if a memory bank change is in order, the required bank must be selected prior to a CALL or JMP instruction.

RETR must always be used to finish an interrupt service routine because this instruction re-enables the interrupt logic. After returning from an interrupt service routine, at least one instruction of the interrupted program is always executed before any new interrupt service routine can be entered. Since it is not possible to change memory banks during interrupt routines, these routines must reside in memory bank 0 where the interrupt vectors are also located.

4.2 Data memory (RAM)

The data memory consists of 64 bytes on the MAB8422 and 128 bytes on the MAB8442. All locations are indirectly addressable using RAM pointer registers and up to 16 designated locations are directly addressable. The memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Fig. 4.2 shows the data memory map.

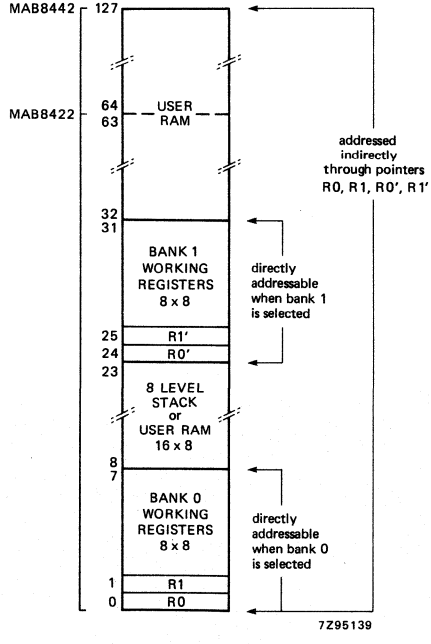


Fig. 4.2 The data memory map.

Locations 0 to 7 are defined as working registers, directly addressable by the direct register instructions. These registers are easily addressed and require the minimum of instruction bytes to manipulate their contents, hence they are commonly used to store frequently accessed intermediate results. This bank of registers can be selected by the SEL RB0 instruction. Executing the select register bank instruction SEL RB1, defines locations 24 to 31 as working registers, instead of locations 0 to 7, and these are then directly addressable. This second bank may be used as an extension of the first one or reserved for use during interrupt service routines, saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first two locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations. RAM locations R0 to R7 and R0' to R7', make efficient program loop counters when used with the decrement register and test instruction DJNZ.

Locations 8 to 23 may be defined as an 8-level program counter stack (2 locations per level), or as general purpose RAM. The program counter stack (Fig. 4.3) enables the processor to keep track of the return addresses and status generated by interrupts or CALL instructions by storing the contents of the program counter prior to servicing the subroutine. A 3-bit stack pointer determines which of the program counter stack's eight register pairs will be loaded with the next return address generated.

The stack pointer, when initialized to 000B by reset, points to RAM locations 8 and 9. On the first subroutine CALL or interrupt, the contents of the program counter and bits 4, 6 and 7 of the program status word (PSW) are transferred to locations 8 and 9. The stack pointer then increments by one and points to locations 10 and 11 ready for another CALL. An address is 13 bits long and therefore, two bytes must be used to store each address.

At the end of a subroutine, which is signalled by a return instruction (RET), the stack pointer decrements by one and the contents of the register pair at the top of the stack are transferred to the program counter. The saved PSW bits are tranferred to the PSW only when the RETR instruction is used.

Levels that are not used for subroutine and interrupt nesting may be used just as any other indirectly addressable RAM locations. Locations from 32 upwards may be used as storage for program variables or data.

If more than 8 levels of interrupt and subroutine nesting are used, the stack pointer will overflow. If an overflow does occur, the deepest address (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. The pointer also underflows from 000 to 111.

The value of the saved contents of the program counter is different for an interrupt CALL compared to a normal CALL subroutine. With an interrupt CALL, the program counter return address is saved; with a subroutine CALL, the saved program counter value is one less than the program counter return address.

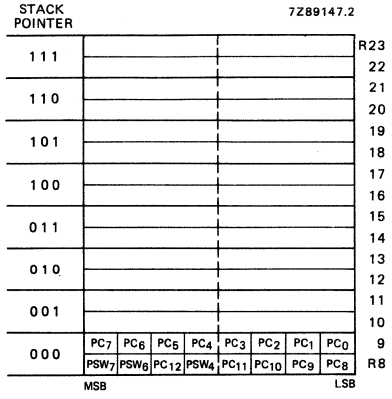


Fig. 4.3 Program counter stack.

5.0 INPUT/OUTPUT FACILITIES

The MAB8422/8442 has 13 I/O lines and 2 testable inputs arranged as:

- An 8 line parallel port P00 - P07, high current outputs with three optional output configurations: push-pull output with pull-up, open drain and open drain with pull up;
- A 2 line port, P10/SDA and P11/SCL, that is software selectable as a parallel port or as an I^2C-bus compatible serial I/O port.

  After RESET, P10/SDA and P11/SCL will be in the parallel port mode. To stay in this mode the internal port latches P12 and P13 must be kept in the logic 1 state. Inputs P12 - P17 are not valid in the parallel port mode.
- A 3 line parallel port P20 - P22 with output configurations as P00 - P07;
- An external interrupt and test input $\overline{INT}$/T0, that when used as a test input can be tested by the conditional jump instructions JT0 and JNT0;
- A test input T1, tested by the conditional jump instructions JT1 and JNT1. T1 can also be used as an input to the timer/event counter. T1 allows zero cross-over sensing of slowly moving A.C. signals.

After a RESET, the microcontroller remains in the parallel port mode until the serial I/O mode is enabled.

5.1 Parallel port mode

Output data written to a port is latched and remains unchanged until rewritten. Input data is not latched and so must not be present until read by an input instruction.

For Ports 0 and 2 all input lines are fully TTL compatible and Port 0 has a 10 mA drive capability. Figure 5.1(a) shows the quasi-bidirectional I/O interface with push-pull output and pull-up transistor. Each line is continuously pulled-up to + 5 V via a relatively high impedance transistor TR3. When a 0 is written to the line, the low impedance of TR1 overcomes the pull-up and provides TTL current sink capability. When a 1 is written, TR2 is momentarily switched on to give fast pull-up. One state of a machine cycle later, the 1 level is still maintained by the high impedance pull-up, so that the line can be used as an input line. This can be done by software control. After RESET, all I/O lines are in the input mode (i.e. TR1 is high impedance). The 1 on these lines can then be easily pulled down to a 0 by external CMOS or TTL circuits. The I/O electrical characteristics of P10/SDA and P11/SCL are I^2C-bus compatible.

Two other interface output characteristics can also be specified:

- open drain output with pull-up (Fig. 5.1(b))
- open drain output without pull-up (Fig. 5.1(c))

Port 0 and port 2 can be specified in any of the three configurations shown. Port 1 is an open drain without pull-up.

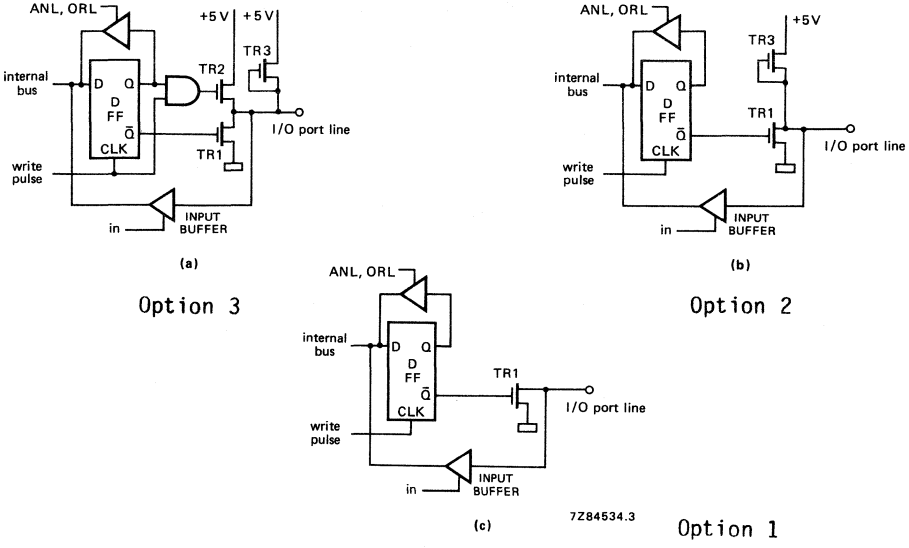


Fig. 5.1 Quasi-bidirectional I/O interface with (a) push-pull output with pull-up, (b) open drain output with pull-up, (c) open drain output without pull-up.

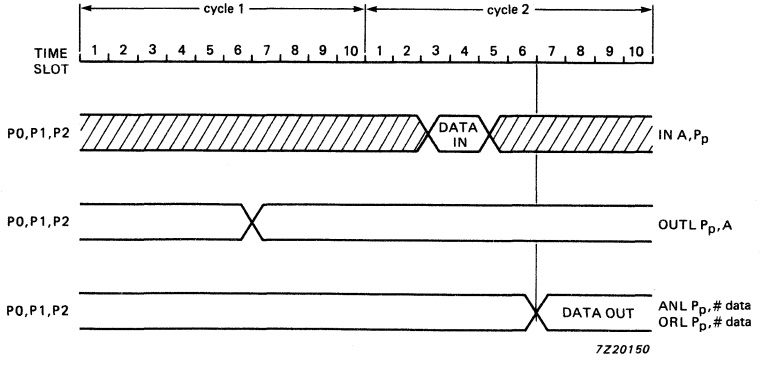


Fig. 5.2 Shows the timing diagram for all ports using IN, OUTL, ANL and ORL instructions. For the OUTL instruction data changes on time slot 7 of cycle 1. For the MOV, ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

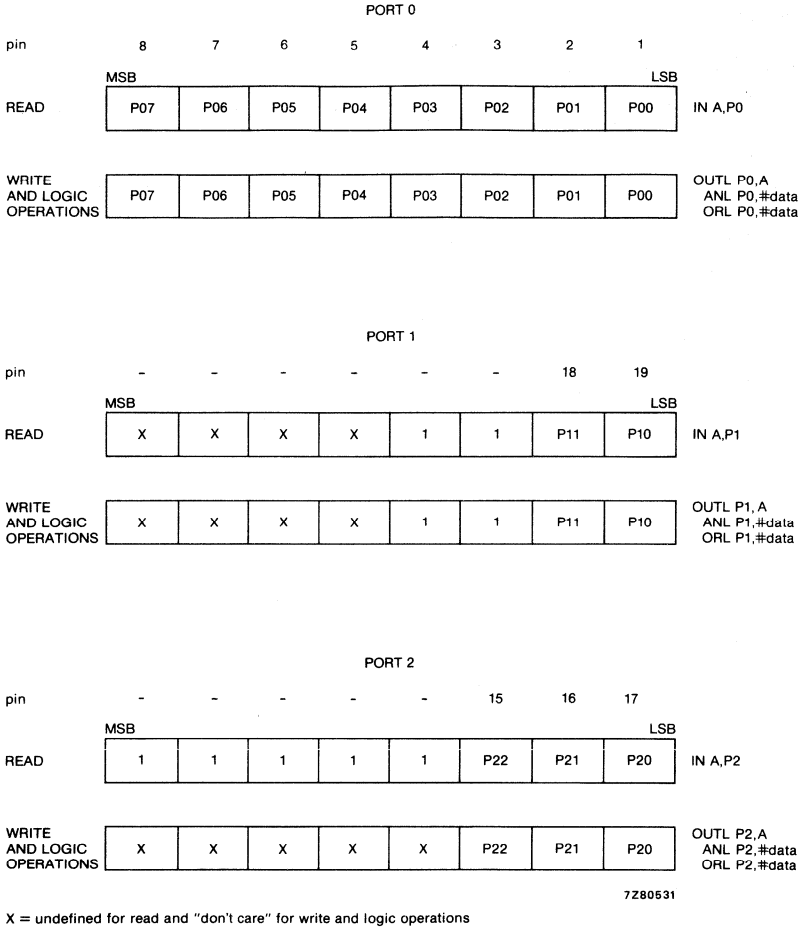


Fig. 5.3 Overview of port signals in the parallel mode.

5.2 Serial I/O mode

5.2.1 Hardware and software functions

The instructions IN A,P1, OUTL P1,A, ORL P1,#data and ANL P1,#data control the serial I/O via port P10 - P17.

Two port lines, P10/SDA and P11/SCL are available as the serial data and serial clock lines, respectively. The input and output levels of both lines are I^2C-bus compatible. P12 - P17 are not available as port pins but are connected internally to the serial I/O circuitry.

All I^2C-bus modes are supported, i.e. master transmitter/receiver and slave transmitter/receiver.

After RESET, P10/SDA and P11/SCL will be in the parallel port mode. The serial I/O mode is enabled by switching port line P12 = NXTBN to logic 0. Then the hardware, in the slave receiver mode, hunts for a valid START condition and if the START condition is found, generates a serial I/O interrupt request. After receiving a START condition, the LOW period of the clock pulse is stretched, suspending the serial transfer to allow the software to run. Switching P13 = STCHN to a logic 0 stretches the next serial clock pulse. Serial transfer is resumed by switching P12 = NXTBN to 1 and then back to 0. The rest of the serial transfer can be ignored by switching P13 = STCHN to 1, e.g. after reception of an address that doesn't match the receivers own slave address. Port lines, P14 - P17 give the status of the I^2C bus and information on the serial bit.

Switching from serial I/O mode to the parallel port mode is achieved by switching P10 - P13 from 1 to 0 followed by writing 1's to P10 - P13.

The serial I/O hardware operates at serial bit level and performs the following functions:

- Filtering the incoming serial data and clock signals;
- Recognizing the START condition;
- Generating a serial interrupt request SIOINT after reception of a START condition;
- Recognizing the STOP condition;
- Recognizing a serial clock pulse on the SCL line;
- Latching a serial bit on the SDA line;
- Stretching the LOW period of the serial clock pulse on the SCL line to suspend the transfer of the next serial data bit;
- Releasing the SCL line to resume the transfer of the next serial data bit.

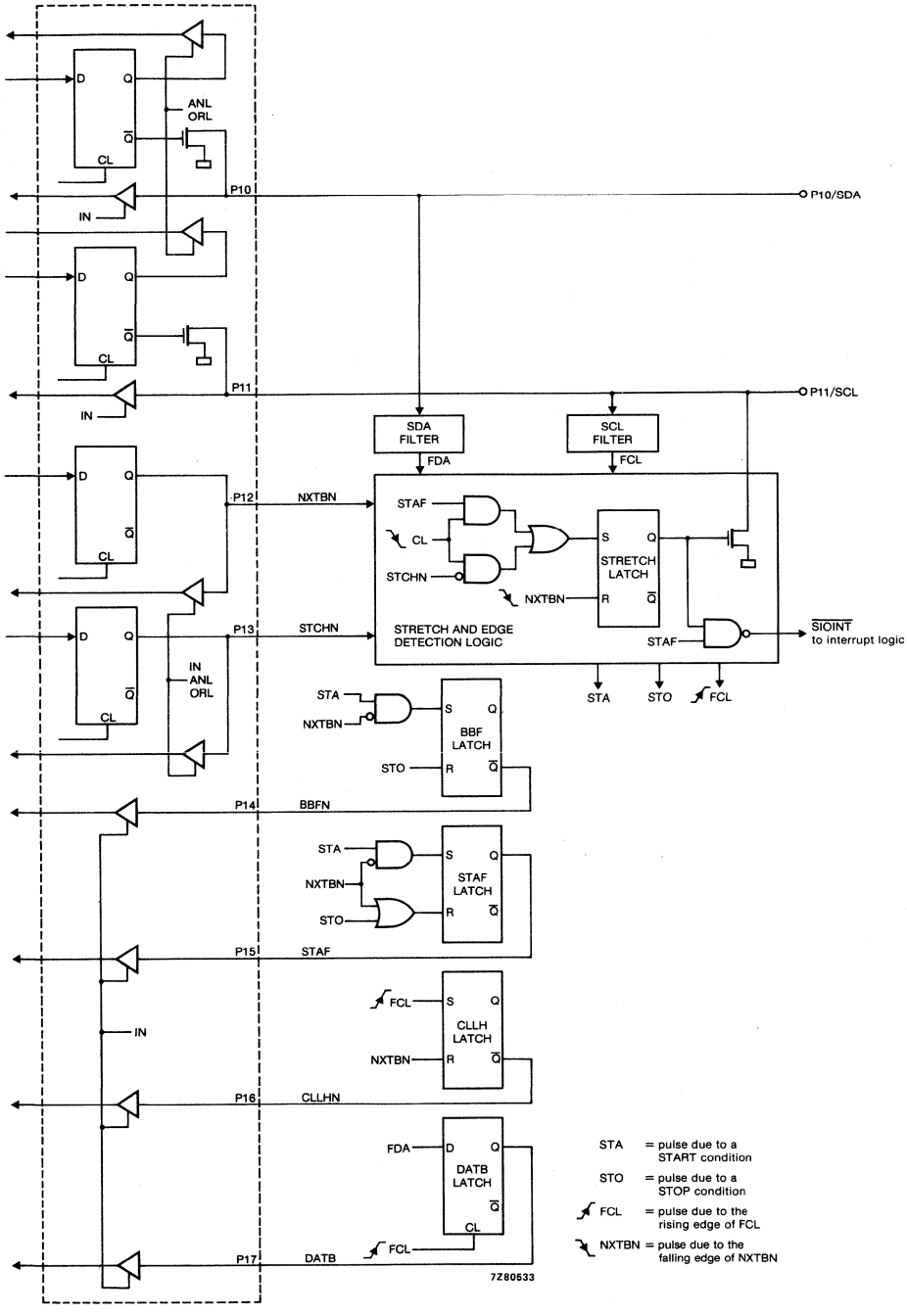


Fig. 5.4 The serial I/O interface.

The following functions should be carried out in software:

- Converting serial to parallel data when receiving;
- Converting parallel to serial data when transmitting;
- Generating START and STOP conditions;
- Generating serial clock pulses;
- Comparing the received slave address with its own address;
- Interpreting the acknowledge information;
- Handling bus arbitration;
- Guarding the I^2C-bus status.

5.2.2 Description of the signals to and from port P10 - P17 in the serial I/O mode

P10: P10/SDA - The serial data line.

P10 is a quasi-bidirectional I/O port with open drain output. It can be read or written with the four port instructions IN A,P1, OUTL P1,A, ORL P1,#data, and ANL P1,#data.

Output data is latched and remains unchanged until it is rewritten. Input data is not latched and is fed to the internal bus and to the SIO logic via a filter *. The filter rejects spikes with a duration of less than 2 clock cycles at the XTAL1 input. The port timing is shown in figure 5.1.

* This filter is only used with SIO signals.

P11: P11/SCL - the serial clock line

P11 has the same I/O properties as P10. In addition it can be pulled down by the SIO logic to stretch the LOW period of the serial clock pulse on the SCL line.

P12: NXTBN - next bit not

P12 is controlled by the instructions
OUTL P1,A, ORL P1,#data, and ANL P1,#data and remains unchanged until it is rewritten. Using the instruction IN A,P1, the state of the port latch is read.

NXTBN = 1
- The STAF latch is reset and held at 0
- The CLLH latch is reset and CLLHN is held at 1

Note: Because no START condition will be detected and no serial interrupt requests can be generated, NXTBN should not be switched to 1 when the SCL line is HIGH.

NXTBN = from 1 to 0
- This transient releases the SCL line if it had been pulled down by the STRETCH latch that is reset by the transient.

NXTBN = 0
- The STAF and CLLH latches are released and the next bit can be transferred on the I^2C-bus.

P13: STCHN - stretch not

P13 is read and written in the same way as P12.

STCHN = 1
- The LOW period of the SCL clock is only stretched if the STAF latch is set.

STCHN = 0
- Every LOW period of the SCL clock is stretched.

P14: BBFN - bus busy flag not

P14 is read-only (via the instruction IN A,P1). OUTL, ANL and ORL instructions to P10 - P17 will not affect P14. The BBF latch is set by a START condition on the I^2C bus, resulting in BBFN = 0 (the bus is busy). The BBF latch is reset by a STOP condition on the I^2C bus, resulting in BBFN = 1 (the bus is free).

P15: STAF - start flag

P15 is read in the same way as P14 and neither will OUTL, ANL and ORL instructions affect P15. The STAF latch is set to 1 by a START condition on the I^2C bus. It is reset to 0 by a STOP condition on the I^2C bus or by NXTBN = 1. The reset overrules the set function.

P16: CLLHN - clock LOW to HIGH not

P16 is read in the same way as P14 and neither will OUTL, ANL and ORL instructions affect P15. The CLLH latch is set by the rising edge of the serial clock pulse on the SCL line, giving CLLHN = 0. It indicates that there is a valid data bit in the DATB latch. The CLLH latch is reset by NXTBN = 1, resulting in CLLHN = 1. The reset overrules the set function.

P17: DATB - data bit

P17 is read in the same way as P14 and neither will OUTL, ANL and ORL instructions affect P15. The DATB latch is clocked at the rising edge of the serial clock pulse on the SCL line. The level on the SDA line is latched, resulting in DATB = 0 or DATB = 1 when the SDA line is LOW or HIGH, respectively.

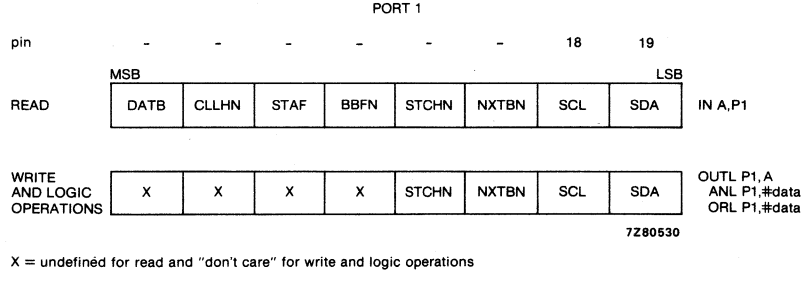


Fig. 5.5 Overview of port signals in the serial mode.

5.2.3 Stretch and edge detection

The STRETCH latch is set at the falling edge of the serial clock pulse on the SCL line if STAF = 1 or if STCHN = 0. The SCL line is pulled down by an open-drain output stage of the STRETCH latch. The STRETCH latch is reset by the falling edge of the NXTBN signal, releasing the SCL line.

Edge detection logic provides control pulses at a START and STOP condition, a rising edge of FCL and a falling edge of NXTBN.

For more detail refer to section 14.0.

5.3 Test input T1

The T1 input line can be used as

- a test input for branch instructions,
- an input for zero voltage cross-over detection,
- an external input to the event counter.

By default, a pull-up transistor is provided on the T1 input. This is useful when the input is from a switch or a standard TTL output. As an option, the pull-up on T1 can be removed to allow the zero cross-over detection circuitry to be used.

When T1 is used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels respectively. The T1 input has a self-biasing circuit that can detect when an a.c. signal crosses zero (within $\pm$ 100 mV when coupled through a 1 μF capacitor). T1 must be used with the open drain port option. The maximum input voltage is 3 V (peak-to-peak) and the maximum frequency is 1 kHz. Zero cross-over detection, when used in conjunction with the timer/event counter interrupt, is useful in thyristor control in power equipment.

The operation of T1 as an input to the event counter is described in section 11.0.

5.4 Test input TO/$\overline{\text{INT}}$

The TO/$\overline{\text{INT}}$ input may be used as:

- an external interrupt
- an input level that may be tested by the conditional jump instructions JTO and JNTO.

When used as an external interrupt input, the interrupt signal must be held LOW for a minimum of 7 clock periods and HIGH for a minimum of 4 clock periods. The interrupt is detected on its negative going edge. The interrupt facility is discussed in more detail in chapter 6.0.

5.5 High current outputs

Ten pins are provided that can sink high currents:

- P10/SDA, pin 19 5 mA at 0,45 V
- P11/SCL, pin 18 5 mA at 0,45 V
- P00-P07, pin 1-8 10 mA at 1 V (High current input)

All ouputs of the same port may be paralleled to obtain high current sinking (up to 80 mA with Port 0 for example).

6.0 INTERRUPT

When the external interrupt is enabled, a HIGH-to-LOW transition on the $\overline{INT}$/TO input initiates an external interrupt subroutine which causes a call to program memory location 3 following completion of the current instruction. The serial I/O interrupt when enabled, invokes a call to location 5. A timer/event counter overflow interrupt, when enabled, invokes a call to location 7.

External interrupts are always latched, even when the external interrupt is disabled. Therefore, keyboard or sensor interrupt requests are not lost when the processor must first perform some necessary functions whilst the external interrupt is disabled. When an interrupt subroutine starts, the program counter contents and bits 4, 6 and 7 of the PSW will be saved in the program counter stack. Accumulator contents have to be saved by user software. Interrupt acknowledgement can be carried out by software via port pins. All interrupt routines must reside in memory bank 0.

The interrupt system is single-level - once an interrupt is detected, further interrupt requests are latched but ignored until the execution of a RETR instruction re-enables the interrupt input logic. After executing RETR, the program continues in the main part. When a second interrupt occurs during the running of the first routine the computer will enter the second routine after having executed one instruction in the main program. If 2 or 3 interrupts occur simultaneously, their priority is: (1) external, (2) serial I/O, (3) timer/ event counter,

Another external interrupt can be created by enabling the timer/event counter interrupt loading FFH into the counter (one less than overflow) and enabling the event counter mode. A LOW-to-HIGH transition on the T1 input will then initiate an interrupt subroutine and cause a call to the timer/counter interrupt vector location 7.

6.1 Interrupt logic

The external interrupt of the MAB84X1 is active on the negative edge; a HIGH to LOW transition on the $\overline{INT}$/TO input will be latched in a 'digital filter/latch' (3-bit shift register in which the negative interrupt is stored; see interrupt logic diagram Fig. 6.1) which when enabled, initiates an external interrupt service routine. The interrupt activity remains latched in the 'digital filter' until it is taken over by the 'External Interrupt Flag', which in turn resets the digital filter. This flag can only be set when external interrupts are enabled.

The 'External Interrupt Flag' when set, causes the current instruction to end, and, if INT has been asserted before time slot 7 of the last cycle of the current instruction, forces a subroutine CALL to program memory location 3. If asserted after this time, the next instruction will be performed first. During this forced CALL the 'External Interrupt Flag' is reset, which enables new interrupts to be latched in the digital filter/latch'. Also the 'Interrupt in Progress Flag' is set, which causes other interrupts to be ignored and latched until the RETR instruction is executed. These (latched) interrupts will invoke an interrupt routine after the program has returned to the main part.

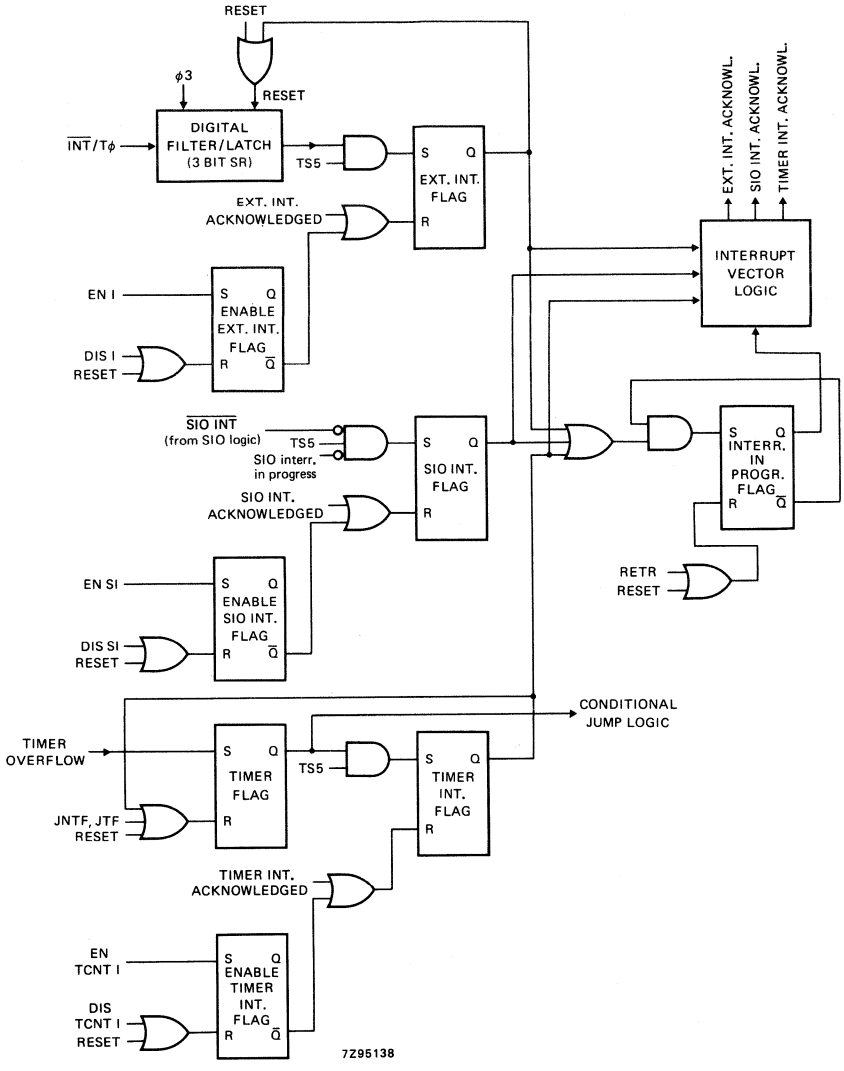


Fig. 6.1 Interrupt logic

Note:

1. $\overline{INT}$/T0 negative edge is always latched in the digital filter/latch.

2. Correct interrupt timing is ensured when $\overline{INT}$/T0 is HIGH for longer than (4 oscillator periods) followed by LOW of longer than (7 oscillator periods).

3. A DIS I instruction within, or immediatly following an interrupt service routine clears a pending external interrupt. A DIS TCNT I within an interrupt service routine clears a pending timer/counter interrupt.

4. When the interrupt in progress flag is set, further interrupts are latched but ignored, until RETR is executed.

5. When the timer/counter interrupt flag has been enabled, the JNTF and JTF instructions will always find the timer flag at 0. This is because the timer flag is only set between TS3 and TS5 of any machine cycle.

To ensure latching in the 'digital filter/latch', the external interrupt should be HIGH for at least 1,33 time slots and LOW for at least 2,33 time slots (1 time slot = 3 clock cycles). The digital filter, therefore acts as a noise supression filter.

The maximum rate at which interrupt edges can be repeated and latched is 4 machine cycles.

Note that $\overline{INT}$/T0 edges are always latched, even when the external interrupts are disabled.

The SIO interrupt circuit is similar to the external interrupt circuit. Here the $\overline{SIOINT}$ (from the SIO logic) can initiate service routines.

Timer interrupt routines can be started by the overflow of the timer which sets the timer flag. This would also set the 'Timer Interrupt Flag', which in turn resets the timer flag.

6.2 Interrupt program examples

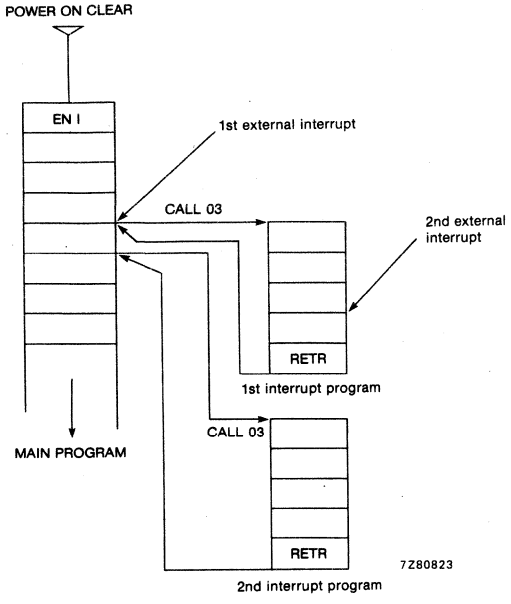


Fig. 6.2 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

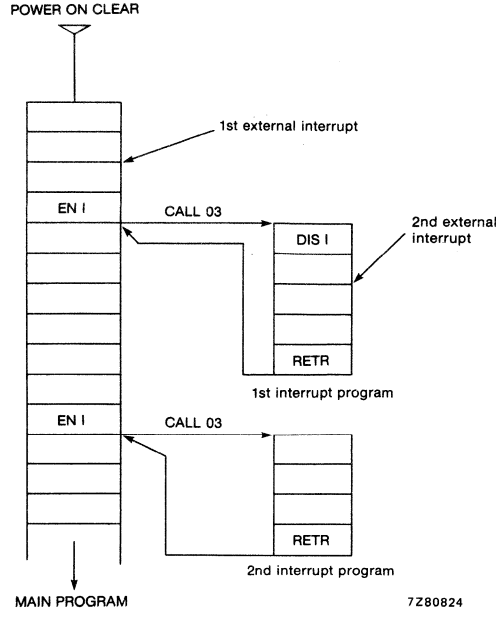


Fig. 6.3 Second external interr -upt is postponed until enabled again.

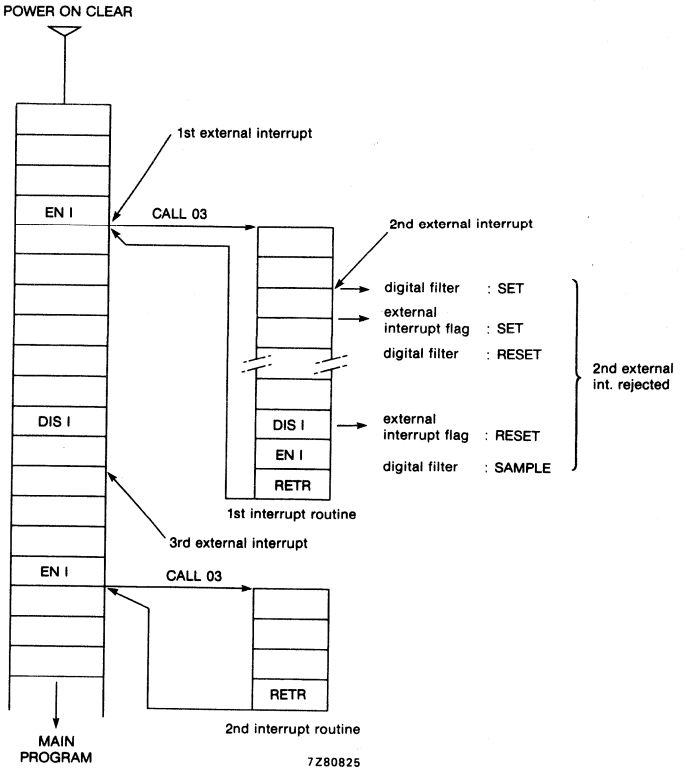


Fig. 6.4 Clearing of previous external interrupt.

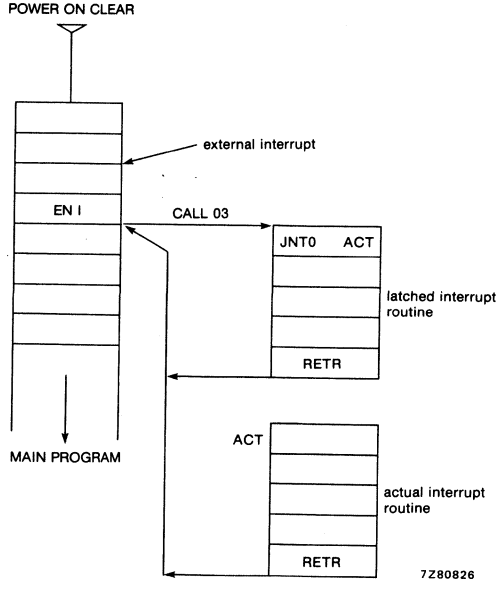


Fig. 6.5 Detection of previous and actual external interrupt.

7.0 CENTRAL PROCESSING UNIT

The MAB8422/42 family has arithmetic, logic and branch capabilities. The DA A, SWAP A, and XCHD instructions simplify BCD arithmetic and bit handling. The MOVP A,@A instruction permits table look up from the current ROM page.

The conditional branch logic within the processor enables several conditions, to be tested by the user's program, internal and external to the processor. Table 2 lists the conditional jump instructions used to change program sequence. The DJNZ instruction decrements a designated register or data memory location and branches if the contents are not zero. This instruction is useful for looping control. The JMPP@A instruction allows multiway branches to destinations the address of which are pointed to by the accumulator.

Table 2 Conditional branches

test	jump condition	jump instruction
accumulator	0 or non-zero	JZ,JNZ
accumulator bit test	1	JB0 to JB7
carry flag	0 or 1	JNC,JC
timer overflow	0 or 1	JNTF,JTF
test input T0	0 or 1	JNT0,JT0
test input T1	0 or 1	JNT1,JT1
register	non-zero	DJNZ

8.0 PROGRAM STATUS WORD

The program status word (PSW) is an 8-bit word (1 byte) in the CPU which stores information about the current status of the microcontroller (Fig. 8.1). The PSW bits are:

bits 0, 1 and 2 - stack pointer bits (SP_0, SP_1, SP_2),
bit 3 - prescaler select (PS); 0 = modulo-32; 1 = modulo-1 (no prescaling),
bit 4 - working register bank select (RBS); 0 = register bank 0; 1 = register bank 1,
bit 5 - not used(1),
bit 6 - auxiliary carry (AC); half carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A,
bit 7 - carry (CY); the carry flag indicates that the previous operation has resulted in an overflow of the accumulator.

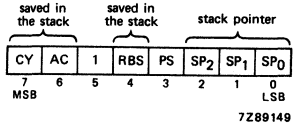


Fig. 8.1 Program status word.

All bits can be read using the MOV A,PSW instruction. Bits 7 and 6 are set are cleared by CPU operation. Bit 4 can be changed by a SEL RB instruction, bit 3 by the MOV PSW,A instruction, and bits 0, 1 and 2 by the CALL, RET or RETR instructions. Bits 7, 6 and 4 are stored in the program counter stack during subroutines and interrupt calls. These bits are restored in the PSW using the RETR (return and restore) instruction which must be used at the end of every interrupt, but which may be used in place of RET at the end of a normal subroutine. The RET instruction has no restore feature and must not be used at the end of an interrupt.

9.0 PROGRAM COUNTER

A 13-bit program counter enables up to 8 K bytes of ROM to be addressed. Figure 9.1 shows the arrangement of the bits. During an interrupt subroutine PC_{11} and PC_{12} are forced to 0. All 13 bits are saved in the stack (see Fig. 4.3) during CALL and interrupt routines.

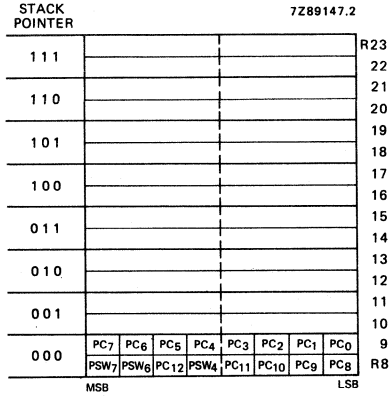


Fig. 9.1 Program counter.

10.0 OSCILLATOR AND TIMING

Clock frequency is determined by using the internal oscillator or by connecting an external clock to XTAL1. Where the internal oscillator is used the frequency is set by a crystal between XTAL1 and XTAL2, or by a ceramic resonator or an inductor, each with two associated capacitors, between XTAL1 and XTAL2. A machine cycle consists of 10 states, each state being 3 oscillator periods. The MAB8422/8442 has dynamic logic, and therefore, for adequate refreshing the oscillator frequency must be at least 1 MHz. (See MAB8422/42 data sheet for a more detailed description).

11.0 TIMER/EVENT COUNTER

An internal 8-bit binary up-counter is provided. This can count external events, modulo-32 machine cycles, or machine cycles directly (Fig. 11.1). Table 3 shows the instructions that control the counter and prescaler and the functions performed.

TABLE 3 Timer/event counter control

function	timer mode modulo-1, module-32*	counter mode
CLEAR	MOV T,A (A) = 0 or RESET	MOV T,A (A) = 0 or RESET
PRESET	MOV T,A	MOV T,A
START	STRT T	STRT CNT
STOP	STOP TCNT or RESET	STOP TCNT or RESET
TEST	JTF/JNTF	JTF/JNTF
READ**	MOV A,T	MOV A,T

* With prescaler select, PS = 0, the timer counts modulo-32 machine cycles, with PS = 1 it counts modulo-1 cycles (prescaler not used); prescaler cleared with STRT T, prescaler not readable.

** READ does not disturb the counting process.

When used as a timer, the input to the counter is either the overflow or input from a 5-bit prescaler. When used as an event counter, LOW-TO-HIGH transitions on T1 (pin 11) are counted. The maximum rate at which the counter may be incremented is once every machine cycle (200 kHz for a 6,77 µs machine cycle). When the counter overflows, the timer flag is set. The flag can be tested and reset using both the JTF (jump if timer flag = 1) and JNTF instruction. Overflow generates an interrupt to the processor when the timer/event counter interrupt is enabled.

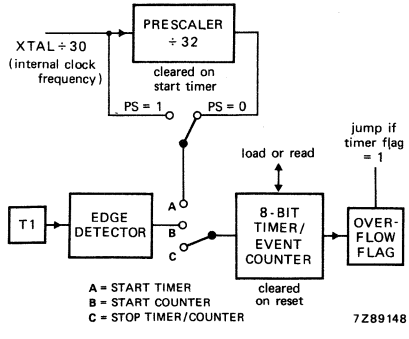


Fig. 11.1 Timer event counter.

12.0 RESET

A positive-going signal on the RESET input:

- sets the program counter to zero
- selects location 0 of memory bank 0
- sets the stack pointer to zero (000); pointing to RAM address 8
- disables the interrupts (external, timer and serial I/O)
- stops the timer/event counter, then sets it to zero
- sets the timer prescaler to modulo-32
- resets the timer flag
- sets all ports to logic 1; the input mode
- sets ports P10/SDA and P11/SCL in the parallel port mode and disables the serial I/O mode

The external power-on reset circuit may consist of a 1 µF capacitor connected between V_{CC} and the RESET pin. A diode may be incorporated between the RESET pin and ground to provide a reset if the supply voltage falls momentarily.

RESET has to be active HIGH for at least 2 machine cycles after the power supply and clock have stabilized

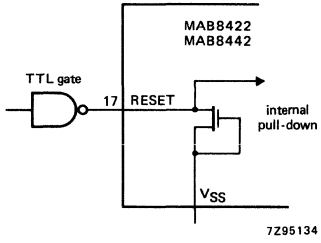


Fig. 12.1 External reset.

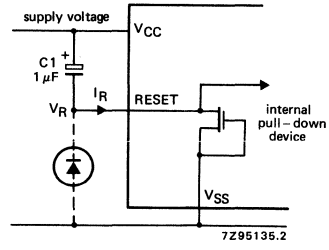


Fig. 12.2 Power-on reset.

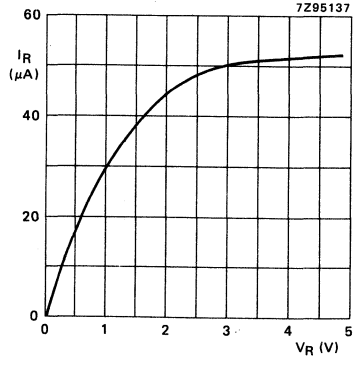


Fig. 12.3 Typical input characteristics.

13.0 DEVELOPMENT SUPPORT

Software for the MAB8422/8442 can be developed with the MAB8422/8442 adapter board (ADB84X2). The adapter board can be used in two ways:

- In combination with the PMDS Microcomputer Adapter Box 8400 (PM8327); in this application the emulation probe is replaced by the ADB84X2.
- For prototyping of MAB8422/8442 based systems; in this application the user program is executed from an EPROM located on the board.

The configuration of the adapter board is selected via jumpers, a block diagram of the board is given in Fig. 13.1.

On the board, a combination of the MAB8401WP bond-out chip and an FPLS circuit 82S105, emulates the MAB8422/8442. The FPLS circuit contains the logic to emulate the serial I/O implemented on the MAB8422/8442. It interfaces with the bond-out chip via P10 - P17. The "user" signals of the bond-out chip are connected directly and via the FPLS circuit to a 20-pin emulation probe that is linked to the target system.

The address and data bus of the MAB8401WP lead to a connector for the PMDS Microcomputer Adapter Box 8400 (PM8327). The address/data bus is also connected to a 28-pin socket that contains the user program in EPROM during prototyping.

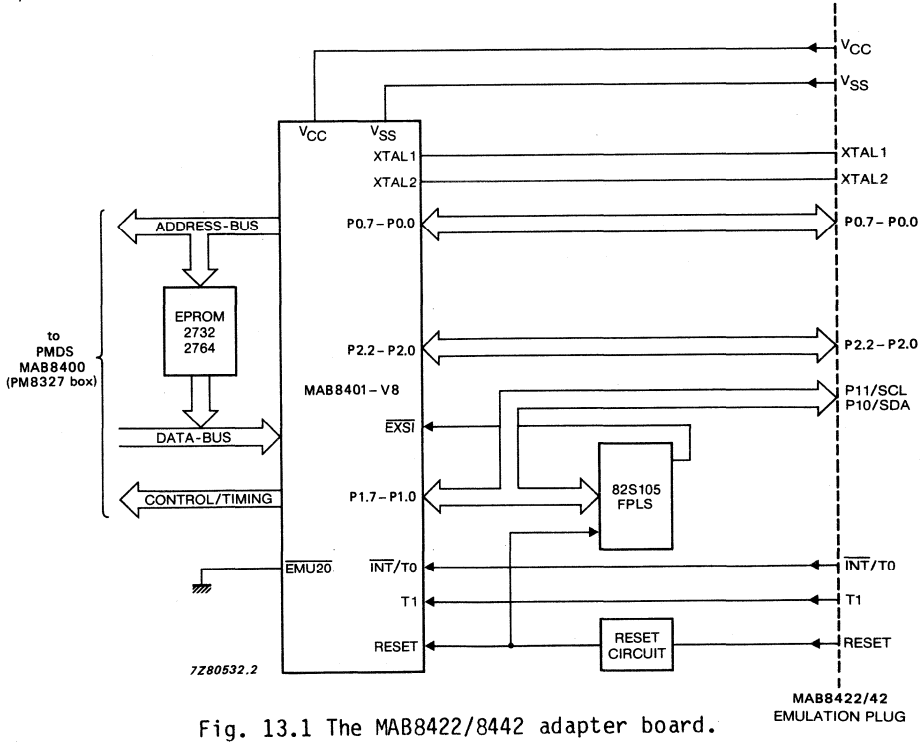


Fig. 13.1 The MAB8422/8442 adapter board.

14.0 I²C-BUS SOFTWARE EXAMPLES

The following subroutines can be used separately, where the processor acts as either a slave or a master, or used together, where the processor is required to be both master and slave. When these two subroutines are used together then line 123 in the slave subroutine and line 126 in the master subroutine should be changed to MASTER EQU USE and SLAVE EQU USED, respectively.

14.1 Slave transmitter/receiver subroutine

Description

This subroutine transmits up to C bytes or receives up to D bytes in one transfer. The limits C and D depend upon the number of bytes (C + D + 1) that can be allocated in the data memory. With a crystal clock frequency of 6 MHz, a maximum transfer rate of 10 kbits/s is possible. However, clock synchronization slows high speed transfers (100 kbits/s) on the I²C-bus to this value. When the MAB8422/42 is not involved in a transfer, it has no effect on I²C-bus speed.

Since this subroutine forms part of the serial I/O interrupt routine, it should reside in the lower 2 K bytes of the program memory. Entering the subroutine via a serial I/O interrupt call, it starts with the slave address and the subsequent R/$\overline{W}$ bit determines the mode of operation (slave transmitter/receiver). The MAB8422/42 acknowledges the master if its own slave address is received, and if it can perform the desired operation. If not, the microcontroller returns a not-acknowledge signal and exits the subroutine by a "return to main program" instruction. The subroutine then enters either the receiver or the transmitter program section, depending on the value of the R/$\overline{W}$ bit.

In the slave receiver mode, data is stored in the allocated registers, and, after each byte is received an acknowledge is sent. On receiving a STOP condition or when all available registers are full, the processor sets the data-valid flag and exits the subroutine by a "return to main program" instruction. Subsequent data bytes are ignored, resulting in a not-acknowledge to the master-transmitter. When a bus error (e.g. a bus obstruction where SDA and/or SCL are held at a fixed level) or spurious START and/or STOP conditions occur, the processor exits the subroutine without setting the data-valid flag. As long as the data-valid flag is set, the slave receiver does not acknowledge its address, indicating that it cannot receive data (it can still operate as a slave transmitter). After the data registers have been read, the main program must clear the data-valid flag.

In the slave transmitter section, data available in the allocated registers is sent but between each byte transmitted, an acknowledge must be received. On receiving a not-acknowledge signal or when all available bytes have been transmitted, the transmission stops, the data-ready flag is cleared and the processor exits the subroutine by a "return to main program" instruction. If a bus error or spurious START and/or STOP condition occurs the I^2C-bus is released and the processor exits the subroutine without clearing the data-ready flag. As long as the data-ready flag is not set, the slave transmitter does not acknowledge its address, indicating that it is not ready to transmit data (it can still operate as a slave receiver).

Note* If a master routine is present, line 123 must be changed into:

```
MASTER   EQU   USED
```

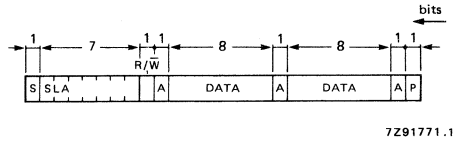
this slave subroutine is also called when in the master routine the arbitration is lost and the devices own slave address is called.

DATA FORMATS

SLAVE TRANSMITTER

R/$\overline{W}$ BIT = '0'

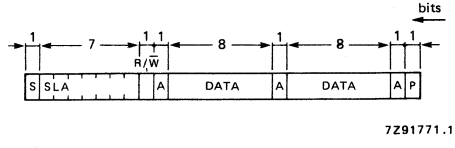
FINAL ACKNOWLEDGE BIT = '1'



7Z91771.1

SLAVE RECEIVER

R/$\overline{W}$ BIT = '1'



7Z91771.1

S = START CONDITION
SLA = 7-BIT SLAVE ADDRESS
R/$\overline{W}$ = READ/NOT WRITE BIT
A = ACKNOWLEDGE BIT
DATA = 8-BIT DATA BYTE
P = STOP CONDITION

Fig. 14.1 Slave transmitter/receiver modes.

Table 4 Serial port equates

signal	definition	pin name	pin number	function
SDA	EQU 01 H	SDA/P10	19	I^2C-bus data I/O
SCL	EQU 02 H	SCL/P11	18	I^2C-bus clock I/O

The internal flags of Port 1 are used to control the serial I/O hardware. These flags cannot be accessed directly by the user via the port pinning.

signal	definition	port bit	R/W	function
NXTBN	EQU 04 H	P12	W	Next bit transfer NOT
STCHN	EQU 08 H	P13	W	Stretch SCL LOW period NOT
BBFN	EQU 10 H	P14	R	Bus busy flag NOT
STAF	EQU 20 H	P15	R	Start condition flag
CLLHN	EQU 40 H	P16	R	SCL LOW to HIGH NOT
DATB	EQU 80 H	P17	R	Data bit received

NOT = active-LOW

```
108         EJECT
109 *       SYMBOL DEFINITION.
110 *       ################
111 *
112 ERRF2   EQU     H'40'           SCL PERIOD EXEEDS TIME LIMIT.
113 SCLC    EQU     250             250 X 12 MACHINE CYCLES TO DEFINE SCL PERIOD
114 *                               TIME LIMIT.(15 MSEC. @ 6 MHZ XTAL)
115 NTBY    EQU     4               MAX. NUMBER OF BYTES TO BE TRANSMITTED.
116 *                               (MUST BE ADAPTED TO THE APPLICATION).
117 NRBY    EQU     4               MAX. NUMBER OF BYTES TO BE RECEIVED.
118 *                               (MUST BE ADAPTED TO THE APPLICATION).
119 OWNAD   EQU     H'50'           OWN SLAVE ADDRESS IN THE 7 MSB. BIT 0 ="0".
120 *                               (MUST BE ADAPTED TO THE APPLICATION).
121 *
122 USED    EQU     1               FOR THE PRESENTS OF A MASTER SUBROUTINE.
123 MASTER  EQU     .NOT.USED       MASTER SUBROUTINE IS NOT PRESENT.
124 *                               IF MASTER SUBROUTINE IS PRESENT, CHANGE INTO:
125 *                               MASTER   EQU    USED
126 *
127 *       DATA MEMORY ALLOCATION.
128 *       ######################
129 *
130 SLVST   EQU     H'30'           SLAVE STATUS REGISTER. CAN BE RELOCATED.
131 *
132 *               SLAVE STATUS REGISTER CONTAINS:
133 *-------------------------------------------------------------------------
134 SRDAVF  EQU     H'80'           SLV/REC DATA-VALID FLAG.
135 STDRF   EQU     H'40'           SLV/TRX DATA-READY FLAG.
136 *-------------------------------------------------------------------------
137 *
138 STDTR1  EQU     SLVST+1           SLV/TRX DATA BYTE REGISTER 1
139 STDTR2  EQU     STDTR1+1                  ,,               2
140 STDTR3  EQU     STDTR2+1                  ,,               3
141 STDTR4  EQU     STDTR3+1                  ,,               4
142 *ETC.
143 SRDTR1  EQU     STDTR4+1          SLV/REC DATA BYTE REGISTER 1 (LAST STDTR+1)
144 SRDTR2  EQU     SRDTR1+1                  ,,               2
145 SRDTR3  EQU     SRDTR2+1                  ,,               3
146 SRDTR4  EQU     SRDTR3+1                  ,,               4
147 *ETC.
148 *THE NUMBER OF DATA BYTE REGISTERS MUST BE ADAPTED TO THE APPLICATION
149 *-------------------------------------------------------------------
```

```
150         EJECT
151 *
152 *       INITIALIZATION.
153 *       ###############
154 *
155 *       THE FOLLOWING INITIALIZATION MUST BE CARRIED OUT BY THE MAIN
156 *       PROGRAM TO ENABLE THE SLAVE OPERATION (IN THE GIVEN ORDER) :
157 *
158 *    1) CLEAR SRDAVF AND STDRF BY WRITING H'00' TO SLVST REGISTER.
159 *       THIS INDICATES NO RECEIVED DATA AVAILABLE AND NO DATA READY TO
160 *       TRANSMIT.
161 *
162 *    2) ENABLE SERIAL INTERRUPT.
163 *
164 *    3) WRITE .NOT.NXTBN (H'FB') TO PORT 1 TO ENABLE THE SERIAL HARDWARE.
165 *
166 *       AFTER THIS INITIALIZATION THE MICROCONTROLLER CAN BE ADDRESSED AS
167 *       SLAVE RECEIVER AND RECEIVE SUBSEQUENT DATA BYTES.
168 *-------------------------------------------------------------------------
169 *
170 *       THIS SLAVE SUBROUTINE WILL AFFECT THE MICROCONTROLLER AS FOLLOWS:
171 *
172 *     - R1, R2, R3, R4, R5 AND R6 IN REGISTER BANK 1 ARE MODIFIED.
173 *
174 *     - SUBROUTINE NESTING = 1 (EXCLUDED THE SERIAL INTERRUPT CALL).
175 *
176 *     - NUMBER OF DATA MEMORY BYTES USED = 1+C+D STARTING AT LOCATION
177 *       H'30' (C = NUMBER OF DATA BYTES TRANSMITTED, D = THE NUMBER OF DATA
178 *       BYTES RECEIVED).
179 *-------------------------------------------------------------------------
```

```
                        180         EJECT
                        181 *
                        182 *       ###########################################################
                        183 *       #       START SLAVE TRANSMITTER/RECEIVER SUBROUTINE       #
                        184 *       ###########################################################
                        185 *
                        186 *ENTRY:  STDTR1 UP TO STDTR(C) CONTAIN UP TO C BYTES TO BE TRANSMITTED.
                        187 *         STDRF   SLV/TRX DATA-READY FLAG; HAS TO BE SET TO "1" AFTER STDTR
                        188 *                 DATA MEMORY LOCATIONS HAVE BEEN LOADED.
                        189 *         SRDAVF  SLV/REC DATA-VALID FLAG; MUST BE CLEARED TO "0" IF SRDTR
                        190 *                 DATA MEMORY LOCATIONS ARE FREE TO RECEIVE DATA.
                        191 *
                        192 *
                        193 *EXIT:   SRDTR1 UP TO STDTR(D) CONTAIN UP TO D RECEIVED BYTES.
                        194 *         SRDAVF  IS SET TO "1" IF DATA IS CORRECTLY RECEIVED.
                        195 *         STDRF   IS CLEARED TO "0" IF DATA CORRECTLY TRANSMITTED.
                        196 *
                        197 *
                        198 SLAVE1 ASECT  ROM
                        199         PAGE   256
000000                  200         ORG    H'05'          SIO INTERRUPT VECTOR
                        201         IF     MASTER=USED
                        202         ENTRY  SIV            ENTRY FROM MULTI-MASTER SUBROUTINE.
                        203         ENTRY  SAR            ENTRY FROM MULTI-MASTER SUBROUTINE.
                        204         ENDIF
                        205 *
000005 0410        1    206 SIV     JMP    SLV            JUMP TO BEGIN OF THE SLAVE SUBROUTINE.
                        207 *
000007                  208         ORG H'010'
                        209 *-------------------------------------------------------------------------
                        210 *               RECEPTION OF SLAVE ADDRESS + READ/WRITE BIT.
                        211 *-------------------------------------------------------------------------
000010 D5          2    212 SLV     SEL    RB1            SELECT REGISTER BANK 1
000011 FE          3    213         MOV    A,R6           SAVE ACCU.
                        214 *
000012 1474        4    215 SLV1    CALL   SRDB           CALL SLV/REC DATA BYTE SUBROUTINE.
000014 B212        5    216         JB5    SLV1           JUMP IF START CONDITION DETECTED.
000016 9643        6    217         JNZ    SLVEX          JUMP IF STOP OR SCL ERROR.
000018 FC          7    218 SAR     MOV    A,R4           FETCH RECEIVED SLAVE ADDRESS.
000019 D5          8    219         SEL    RB1            POSSIBLE ENTRY AT SAR FROM MASTER SUBROUTINE.
00001A D350        9    220         XRL    A,#OWNAD       COMPARE WITH OWN SLAVE ADDRESS.
00001C C649       10    221         JZ     SRDT           JUMP IF ADDRESSED AS SLAVE RECEIVER.
00001E 07         11    222         DEC    A              COMPLEMENT READ/WRITE BIT.
00001F 9643       12    223         JNZ    SLVEX          JUMP IF NOT ADDRESSED AS SLAVE TRANSMITTER.
```

```
                                224         EJECT
                                225 *       SLAVE TRANSMITTER ROUTINE.
                                226 *       ==========================
                                227 *-------------------------------------------------------------------------
                                228 *           - GENERATION OF AN ACKNOWLEDGE ("0") IF THE DATA-READY FLAG
                                229 *             "STDRF" HAS BEEN SET, OTHERWISE THE ROUTINE IS EXITTED.
                                230 *           - TRANSMISSION OF DATA BYTES STORED IN THE DATA MEMORY
                                231 *             LOCATIONS STDTR1, STDTR2 ETC.
                                232 *           - RECEPTION OF THE ACKNOWLEDGE BIT AFTER EACH TRANSMITTED
                                233 *             BYTE. WHEN A NOT-ACKNOWLEDGE IS RECEIVED, THE
                                234 *             TRANSMISSION STOPS AND "STDRF" IS CLEARED.
                                235 *           - IF THE MAX. NUMBER OF BYTES "NTBY" HAS BEEN
                                236 *             TRANSMITTED, THE TRANSMISSION STOPS AND "STDRF" IS CLEARED.
                                237 *           - IF A BUS ERROR IS DETECTED, THE SLAVE SUBROUTINE TERMINATES
                                238 *             WITHOUT CLEARING "STDRF".
                                239 *-------------------------------------------------------------------------
                                240 *
000021 B930          13         241 STDT    MOV     R1,#SLVST       LOAD SLAVE STATUS LOCATION IN R1
000023 F1            14         242         MOV     A,@R1           FETCH SLAVE STATUS.
000024 37            15         243         CPL     A
000025 D243          16         244         JB6     SLVEX           JUMP IF STDRF IS NOT SET.
000027 14B8          17         245         CALL    SRAC            OUTPUT ACKNOWLEDGE.
000029 9643          18         246         JNZ     SLVEX           JUMP IF SCL ERROR.
00002B 19            19         247         INC     R1              LOCATION OF FIRST SLV/TRX DATA REG. TO R1.
00002C BD04          20         248         MOV     R5,#NTBY        MAXIMUM NUMBER OF BYTES IN R5.
                                249 *
00002E F1            21         250 STDT1   MOV     A,@R1           FETCH DATA BYTE TO BE TRANSMITTED.
00002F 19            22         251         INC     R1              NEXT SLV/TRX DATA REG. LOCATION.
000030 1496          23         252         CALL    STDB            OUTPUT DATA BYTE.
000032 9643          24         253         JNZ     SLVEX           JUMP IF START,STOP OR SCL ERROR.
000034 14BD          25         254         CALL    STAC            INPUT ACKNOWLEDGE BIT.
000036 9643          26         255         JNZ     SLVEX           JUMP IF START, STOP OR SCL ERROR.
000038 FC            27         256         MOV     A,R4            FETCH ACKNOWLEDGE BIT
000039 963D          28         257         JNZ     STEX            JUMP IF NOT-ACKNOWLEDGE.
00003B ED2E          29         258         DJNZ    R5,STDT1        JUMP IF NOT MAX. NUMBER OF BYTES TRANSMITTED.
                                259 *
00003D B930          30         260 STEX    MOV     R1,#SLVST       LOAD SLAVE STATUS LOCATION IN R1.
00003F F1            31         261         MOV     A,@R1           FETCH SLAVE STATUS.
000040 53BF          32         262         ANL     A,#.NOT.STDRF   CLEAR SLV/TRX DATA READY FLAG.
000042 A1            33         263         MOV     @R1,A           SAVE SLAVE STATUS.
                                264 *
                                265 *
                                266 *       SLAVE MODE EXIT ROUTINE.
                                267 *       ========================
000043 89FF          34         268 SLVEX   ORL     P1,#SDA+SCL+NXTBN+STCHN+H'F0'   SET SDA TO HIGH AND RELEASE
                                269 *                                           SCL(STILL STRETCHED); DISABLE STRETCHING.
000045 99FB          35         270         ANL     P1,#.NOT.NXTBN  SET SCL OUTPUT TO HIGH.
                                271 *
000047 FE            36         272 SIOEX   MOV     A,R6            RESTORE ACCU.
000048 93            37         273         RETR
```

```
                        274         EJECT
                        275 *
                        276 *       SLAVE RECEIVER ROUTINE.
                        277 *       =======================
                        278 *-------------------------------------------------------------------------
                        279 *             - GENERATION OF AN ACKNOWLEDGE ("0") IF THE DATA-VALID FLAG
                        280 *               "SRDAVF" HAS BEEN CLEARED, OTHERWISE THE ROUTINE IS EXIT.
                        281 *             - RECEPTION OF DATA BYTES THAT ARE THEN STORED IN THE DATA
                        282 *               MEMORY LOCATIONS SRDTR1, SRDTR2, ETC.
                        283 *             - TRANSMISSION OF AN ACKNOWLEDGE ("0") AFTER EACH RECEIVED
                        284 *               BYTE.
                        285 *             - ON RECEIPT OF A STOP CONDITION, OR IF ALL DATA MEMORY
                        286 *               LOCATIONS ARE OCCUPIED, THE "SRDAVF" IS SET AND THE
                        287 *               ROUTINE IS LEFT.
                        288 *             - IF A BUS ERROR IS DETECTED THE SLAVE SUBROUTINE TERMINATES
                        289 *               WITHOUT SETTING "SRDAVF".
                        290 *-------------------------------------------------------------------------
                        291 *
000049 B930      38     292 SRDT    MOV     R1,#SLVST       LOAD SLAVE STATUS LOCATION IN R1.
00004B F1        39     293         MOV     A,@R1           FETCH SLAVE STATUS.
00004C F243      40     294         JB7     SLVEX           JUMP IF SRDAVF IS SET.
00004E B935      41     295         MOV     R1,#SRDTR1      LOCATION OF FIRST SLV/REC.
000050 BD04      42     296         MOV     R5,#NRBY        MAX.NUMBER OF BYTES IN R5.
                        297 *
000052 14B8      43     298 SRDT1   CALL    SRAC            OUTPUT ACKNOWLEDGE.
000054 9643      44     299         JNZ     SLVEX           JUMP IF SCL ERROR.
000056 1474      45     300         CALL    SRDB            INPUT DATA BYTE.
000058 966B      46     301         JNZ     SRDT3           JUMP IF START,STOP COND. OR SCL ERROR.
00005A FC        47     302         MOV     A,R4            FETCH RECEIVED DATA BYTE.
00005B A1        48     303         MOV     @R1,A           SAVE RECEIVED DATA BYTE.
00005C 19        49     304         INC     R1              NEXT SLV/REC DATA REG. LOCATION.
00005D ED52      50     305         DJNZ    R5,SRDT1        JUMP IF NOT MAX NUMBER OF BYTES.
00005F 14B8      51     306         CALL    SRAC            OUTPUT ACKNOWLEDGE.
000061 9643      52     307         JNZ     SLVEX           JUMP IF STOP OR SCL ERROR.
000063 B930      53     308 SRDT2   MOV     R1,#SLVST       LOAD SLAVE STATUS LOCATION IN R1.
000065 F1        54     309         MOV     A,@R1           FETCH SLAVE STATUS.
000066 4380      55     310         ORL     A,#SRDAVF       SET "SRDAVF" FLAG.
000068 A1        56     311         MOV     @R1,A           SAVE SLAVE STATUS.
000069 0443      57     312         JMP     SLVEX
                        313 *
00006B 5243      58     314 SRDT3   JB2     SLVEX           JUMP IF SCL ERROR.
                        315 *                               OTHERWISE A STOP CONDITION HAS BEEN RECEIVED,
                        316 *                               POSSIBLY FOLLOWED BY A NEW START CONDITION.
00006D FB        59     317         MOV     A,R3            FETCH BIT COUNTER.
00006E D308      60     318         XRL     A,#8            TEST IF BIT COUNTER=8
000070 C663      61     319         JZ      SRDT2           JUMP IF STOP CONDITION WAS AT THE END OF THE
                        320 *                               BYTE.
000072 0443      62     321         JMP     SLVEX           ERROR, STOP CONDITION WAS WITHIN THE BYTE.
                        322 *
```

```
                        323         EJECT
                        324 *       ##########################################################
                        325 *       #       SINGLE BYTES AND ACKNOWLEDGE SUBROUTINES.        #
                        326 *       ##########################################################
                        327 *
                        328 *       SLAVE RECEIVER SINGLE DATA BYTE SUBROUTINE.
                        329 *       ==========================================
                        330 *-------------------------------------------------------------------------------
                        331 * SUBROUTINE INPUTS 8 BITS OF DATA IN SLAVE RECEIVER MODE
                        332 * EXIT: A = 00; DATA IS RECEIVED CORRECTLY
                        333 *               R4 CONTAINS DATA BYTE AND
                        334 *               BIT COUNTER R3 = 0
                        335 *        A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        336 *                  A START CONDITION IS DETECTED.
                        337 *        A = BBNF; STOP CONDITION DETECTED
                        338 *        A = ERRF2; SCL EXCEEDS TIME LIMIT
                        339 *-------------------------------------------------------------------------------
000074 BB08       63    340 SRDB    MOV     R3,#8           SET BIT COUNTER.
000076 8907       64    341 SRDB1   ORL     P1,#SDA+SCL+NXTBN  SET P10/SDA HIGH, RELEASE P11/SCL.
000078 99F3       65    342         ANL     P1,#.NOT.(NXTBN+STCHN)  SET P11/SCL HIGH; ENABLE STRETCHING.
00007A BAFA       66    343         MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
                        344 *
00007C 09         67    345 SRDB2   IN      A,P1            INPUT STATUS.
00007D F7         68    346         RLC     A               SHIFT RECEIVED BIT IN CARRY.
00007E 53E4       69    347         ANL     A,#B'11100100'  TEST CLLHN*STAF*BBNF*SCL=0
000080 C68B       70    348         JZ      SRDB4           JUMP IF DATA BIT HAS BEEN RECEIVED.
                        349 *
000082 B292       71    350 SRDB3   JB5     SRDB5           JUMP IF STOP CONDITION HAS BEEN RECEIVED.
000084 D292       72    351         JB6     SRDB5           JUMP IF START CONDITION HAS BEEN RECEIVED.
000086 EA7C       73    352         DJNZ    R2,SRDB2        JUMP IF SCL COUNTER NOT ZERO.
000088 2340       74    353         MOV     A,#ERRF2        SET ERROR FLAG; SCL PERIOD TIME-OUT.
00008A 83         75    354         RET
                        355 *
00008B 2C         76    356 SRDB4   XCH     A,R4            DATA BYTE TO ACCU.
00008C F7         77    357         RLC     A               SHIFT RECEIVED BIT IN DATA BYTE.
00008D 2C         78    358         XCH     A,R4            SAVE DATA BYTE IN R4.
00008E EB76       79    359         DJNZ    R3,SRDB1        DECR. AND JUMP IF BIT COUNTER NOT ZERO.
000090 27         80    360         CLR     A               SET FLAGS TO ZERO.
000091 83         81    361         RET
                        362 *
000092 77         82    363 SRDB5   RR A
000093 5330       83    364         ANL     A,#STAF+BBFN    MASK FLAGS.
000095 83         84    365         RET
                        366 *
```

```
                        367         EJECT
                        368 *
                        369 *       SLAVE TRANSMITTER SINGLE DATA BYTE SUBROUTINE.
                        370 *       ==============================================
                        371 *-------------------------------------------------------------------------
                        372 * SUBROUTINE OUTPUTS 8 BITS OF DATA IN SLAVE TRANSMITTER MODE.
                        373 * ENTRY: A CONTAINS DATA BYTE TO BE TRANSMITTED.
                        374 * EXIT: A = 00; DATA BYTE IS OUTPUT CORRECTLY
                        375 *                 R4 CONTAINS THE TRANSMITTED DATA BYTE
                        376 *        A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        377 *                  A START CONDITION IS DETECTED.
                        378 *        A = BBNF; STOP CONDITION DETECTED
                        379 *        A = ERRF2; SCL EXCEEDS TIME LIMIT.
                        380 *-------------------------------------------------------------------------
000096 BB08       85    381 STDB    MOV     R3,#8           SET BIT COUNTER.
000098 AC         86    382         MOV     R4,A            DATA BYTE TO R4.
000099 FC         87    383 STDB0   MOV     A,R4            FETCH DATA BYTE.
00009A E7         88    384         RL      A               DATA BIT TO BIT 0 OF ACCU.
00009B AC         89    385         MOV     R4,A            SHIFTED DATA BYTE TO R4.
00009C 43FE       90    386 STDB1   ORL     A,#SCL+NXTBN+STCHN+H'F0'  PREPARE P1 SIGNALS.
00009E 39         91    387         OUTL    P1,A            WRITE DATA BIT TO P10/SDA, RELEASE P11/SCL.
00009F 99F3       92    388         ANL     P1,#.NOT.(NXTBN+STCHN)  SET P11/SCL HIGH.
0000A1 BAFA       93    389         MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
                        390 *
0000A3 09         94    391 STDB2   IN      A,P1            FETCH STATUS.
0000A4 5372       95    392         ANL     A,#CLLHN+STAF+BBFN+SCL  MASK FLAGS.
0000A6 C6B1       96    393         JZ      STDB4           JUMP IF DATA BIT CORRECTLY TRANSMITTED.
                        394 *
0000A8 92B5       97    395 STDB3   JB4     STDB5           JUMP IF STOP CONDITION HAS BEEN RECEIVED.
0000AA B2B5       98    396         JB5     STDB5           JUMP IF START CONDITION HAS BEEN RECEIVED.
0000AC EAA3       99    397         DJNZ    R2,STDB2        JUMP IF SCL TIME OUT COUNTER NOT ZERO
0000AE 2340      100    398         MOV     A,#ERRF2        SET ERROR FLAG; SCL PERIOD TIME OUT.
0000B0 83        101    399         RET
                        400 *
0000B1 EB99      102    401 STDB4   DJNZ    R3,STDB0        JUMP IF BIT COUNTER NOT ZERO.
0000B3 27        103    402         CLR     A               CLEAR FLAGS
0000B4 83        104    403         RET
                        404 *
0000B5 5330      105    405 STDB5   ANL     A,#STAF+BBFN    MASK FLAGS.
0000B7 83        106    406         RET
```

```
                        407         EJECT
                        408 *       SLAVE RECEIVER ACKNOWLEDGE SUBROUTINE.
                        409 *       ======================================
                        410 *-------------------------------------------------------------------------
                        411 *SUBROUTINE OUTPUTS AN ACKNOWLEDGE ("0") IN SLAVE RECEIVER MODE
                        412 * EXIT: A = 00; ACKNOWLEDGE IS OUTPUT CORRECTLY
                        413 *       A = ERRF2; SCL EXCEEDS TIME LIMIT
                        414 *-------------------------------------------------------------------------
0000B8 BB01       107   415 SRAC    MOV     R3,#1           SET BIT COUNTER.
0000BA 27         108   416         CLR     A               TO SEND A "0" AS ACKNOWLEDGE.
0000BB 049C       109   417         JMP     STDB1           JUMP TO SLV/TRX MODE.
                        418 *
                        419 *
                        420 *
                        421 *       SLAVE RECEIVER NOT-ACKNOWLEDGE SUBROUTINE OR
                        422 *       SLAVE TRANSMITTER INPUT ACKNOWLEDGE SUBOUTINE.
                        423 *       ==============================================
                        424 *-------------------------------------------------------------------------
                        425 * SUBROUTINE OUTPUTS A NOT-ACKNOWLEDGE ("1") IN SLAVE RECEIVER MODE.
                        426 * SUBROUTINE INPUTS ACKNOWLEDGE IN SLAVE TRANSMITTER MODE.
                        427 * EXIT: A = 00; ACKNOWLEDGE BIT IS OUTPUT OR RECEIVED CORRECTLY.
                        428 *               R4 = 0; ACKNOWLEDGE ("0") RECEIVED.
                        429 *               R4 = 1; NOT-ACKNOWLEDGE ("1") RECEIVED.
                        430 *       A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        431 *                 A START CONDITION IS DETECTED.
                        432 *       A = BBNF; STOP CONDITION DETECTED
                        433 *       A = ERRF2; SCL EXCEEDS TIME LIMIT
                        434 *-------------------------------------------------------------------------
0000BD BB01       110   435 STAC    MOV     R3,#1           SET BIT COUNTER.
0000BF BC00       111   436         MOV     R4,#0           CLEAR DATA REGISTER.
0000C1 0476       112   437         JMP     SRDB1           JUMP TO SLV/REC MODE.
                        438 *
0000C3                  439         END

  NO ERRORS DETECTED
```

14.2 Multi-master subroutine

Description

This subroutine transmits up to E bytes or receives up to F bytes in one transfer, with or without pointer byte and repeated START condition. The limits E and F depend upon the number of bytes (4 + E + F) that can be allocated in the data memory. The I^2C interface of the MAB8422/42 filters out spikes of less than 2 crystal clock periods, improving data transfer reliability. With a crystal clock frequency of 6 MHz, transfer rates between 8 and 9,5 kbits/s are possible. However, clock synchronization slows high speed transfers (100 kbits/s) on the I^2C-bus to this value. When the MAB8422/42 is not involved in a transfer, it has no effect on I^2C-bus speed. Arbitration on the serial data line and synchronization on the clock line follow the I^2C specification permitting multi-master operation.

If another master transmits a START condition, this subroutine reacts via a serial I/O interrupt routine by stretching the SCL line during the START condition. It releases the bus after the START condition, and the rest of the data transfer is unaffected. When arbitration is lost, another subroutine is used to release the bus. These two subroutines are found at the end of the program. Also, if the slave routine is present, these interrupt routines are deleted by changing line 126 to:

```
SLAVE   EQU     USED
```

Error conditions

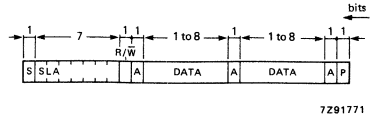
Any additional clock pulses are dealt with in the synchonization of the serial clocks, and require no software action. However, transfer is terminated and appropriate error flags set when spurious START and/or STOP conditions occur. Furthermore, noise that causes an error in the level of a bit will set a flag indicating arbitration lost. Due to the wired-AND structure of the bus, noise that distorts a 0 to form a 1 is unlikely and is not handled by this subroutine. A software timer monitors the SCL LOW period and if a device stretches the period beyond the 7,5 ms time-out value, transfer is terminated and an error flag is set. When the bus is occupied for more than 1 second (6 MHz crystal), up to 10 forced STOP conditions are generated to free the bus and transfer is attempted again. If this attempt is also unsuccessful, an error flag is set.

DATA FORMATS

(a)

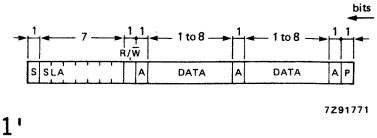
MASTER TRANSMITTER NO POINTER

R/W BIT = '0'

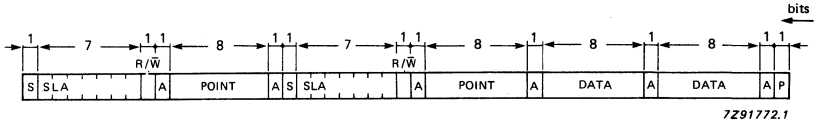


MASTER RECEIVER NO POINTER

R/W BIT = '1', FINAL ACKNOWLEDGE BIT = '1'



(b)



MASTER TRANSMITTER WITH POINTER AND
REPEATED START CONDITION

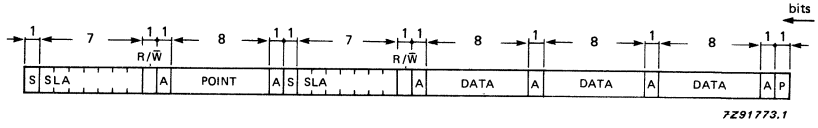


Fig. 14.2 Multi-master transmitter/receiver modes.

MASTER RECEIVER WITH POINTER AND
REPEATED START CONDITION
FINAL ACKNOWLEDGE BIT = '0'

S = START CONDITION
SLA = 7-BIT SLAVE ADDRESS
R/W = READ/NOT WRITE BIT
A = ACKNOWLEDGE/ NOT ACKNOWLEDGE BIT
POINT = 8-BIT POINTER BYTE
DATA = 8-BIT DATA BYTE
P = STOP CONDITION

The number of data bytes transmitted is specified in register 'DBCNTR'

Table 5 serial port equates

signal definition			pin name	pin number	function
SDA	EQU	01 H	SDA/P10	19	I^2C-bus data I/O
SCL	EQU	02 H	SCL/P11	18	I^2C-bus clock I/O

The internal flags of Port 1 are used to control the serial I/O hardware. These flags cannot be accessed directly by the user via the port pinning.

signal definition			port bit	R/W	Function
NXTBN	EQU	04 H	P12	W	Next bit transfer NOT
STCHN	EQU	08 H	P13	W	Stretch SCL LOW period NOT
BBFN	EQU	10 H	P14	R	Bus busy flag NOT
STAF	EQU	20 H	P15	R	Start condition flag
CLLHN	EQU	45 H	P16	R	SCL LOW to HIGH NOT
DATB	EQU	80 H	P17	R	Data bit received

NOT = active-LOW

```
107         EJECT
108 *       SYMBOL DEFINITION.
109 *       ##################
110 *
111 ERRF1   EQU     H'80'           ARBITRATION LOST.
112 ERRF2   EQU     H'40'           BUS OBSTRUCTED; SCL AND/OR SDA STAY LOW.
113 ERRF3   EQU     H'20'           SPURIOUS START CONDITION.
114 ERRF4   EQU     H'10'           SPURIOUS STOP CONDITION.
115 NAACK   EQU     H'02'           NO ADDRESS ACKNOWLEDGE FROM SLAVE.
116 NDACK   EQU     H'01'           NO DATA ACKNOWLEDGE FROM SLAVE RECEIVER.
117 *
118 RWB     EQU     H'01'           READ/WRITE BIT AFTER SLAVE ADDRESS.
119 SCLC    EQU     250             250 X 6 MACHINE CYCLES TO DEFINE SCL PERIOD
120 *                               TIME LIMIT. (7.5 MSEC @ 6 MHZ XTAL)
121 *                               CAN BE ADAPTED TO THE APPLICATION.
122 BTOC    EQU     100             THIS NUMBER TIMES 10 MSEC DEFINES THE BUS.
123 *                               TIME-OUT LIMIT IN CASE OF A BUS OBSTRUCTION.
124 *                               CAN BE ADAPTED TO THE APPLICATION.
125 USED    EQU     1               FOR THE PRESENCE OF A SLAVE SUBROUTINE.
126 SLAVE   EQU     .NOT.USED       SLAVE SUBROUTINE IS NOT PRESENT.
127 *                               IF SLAVE SUBROUTINE IS PRESENT, CHANGE INTO:
128 *                               SLAVE  EQU    USED
129 *
130 *       DATA MEMORY ALLOCATION.
131 *       ######################
132 *
133 SIOMDR  EQU     H'20'           SIO MODE REGISTER. CAN BE RELOCATED.
134 *               SIO MODE REGISTER CONTAINS:
135 *-----------------------------------------------------------------------
136 SLVMF   EQU     H'00'           SLAVE MODE; I.E. NOT ACTIVE AS MASTER.
137 MTMF    EQU     H'08'           MASTER/TRANSMITTER MODE WITHOUT POINTER-BYTE
138 MRMF    EQU     H'04'           MASTER/RECEIVER MODE WITHOUT POINTER-BYTE.
139 MTPMF   EQU     H'02'           MASTER/TRANSMITTER MODE WITH POINTER-BYTE AND
140 *                               REPEATED START CONDITION.
141 MRPMF   EQU     H'01'           MASTER/RECEIVER MODE WITH POINTER-BYTE AND
142 *                               REPEATED START CONDITION.
143 *-----------------------------------------------------------------------
144 SLVADR  EQU     SIOMDR+1        SLAVE ADDRESS REGISTER
145 DBCNTR  EQU     SLVADR+1        DATA BYTE COUNTER REGISTER
146 POINTR  EQU     DBCNTR+1        POINTER VALUE REGISTER
147 *
148 MTDTR1  EQU     POINTR+1        MST/TRX DATA BYTE REGISTER 1
149 MTDTR2  EQU     MTDTR1+1                ,,                 2
150 MTDTR3  EQU     MTDTR2+1                ,,                 3
151 MTDTR4  EQU     MTDTR3+1                ,,                 4
152 * ETC.
153 MRDTR1  EQU     MTDTR4+1        MST/REC DATA BYTE REGISTER 1 (LAST MTDTR+1)
154 MRDTR2  EQU     MRDTR1+1                ,,                 2
155 MRDTR3  EQU     MRDTR2+1                ,,                 3
156 MRDTR4  EQU     MRDTR3+1                ,,                 4
157 * ETC.
158 *THE NUMBER OF DATA BYTE REGISTERS MUST BE ADAPTED TO THE APPLICATION.
159 *-----------------------------------------------------------------------
```

```
160         EJECT
161 *
162 *       INITIALIZATION.
163 *       ###############
164 *
165 *       THE FOLLOWING INITIALIZATION MUST BE CARRIED OUT BY THE MAIN
166 *       PROGRAM TO ENABLE THE MULTI-MASTER OPERATION (IN THE ORDER GIVEN) :
167 *
168 *    1) SELECT REGISTER BANK 0 (THE SLAVE SUBROUTINE USES REGISTER BANK 1).
169 *
170 *    2) ENABLE SERIAL INTERRUPT.
171 *
172 *    3) WRITE .NOT.NXTBN (H'FB') TO PORT 1 TO ENABLE THE SERIAL I/O
173 *       HARDWARE.
174 *-------------------------------------------------------------------------
175 *       THIS  MULTI-MASTER SUBROUTINE WILL AFFECT THE MICROCONTROLLER
176 *       AS FOLLOWS:
177 *
178 *     - R1, R2, R3, R4 AND R5 IN REGISTER BANK 0 ARE MODIFIED.
179 *
180 *     - THE ACCUMULATOR AND CARRY BIT ARE MODIFIED.
181 *
182 *     - SUBROUTINE NESTING = 1 (INCLUDING THE SERIAL INTERRUPT CALL).
183 *
184 *     - NUMBER OF DATA MEMORY BYTES USED = 4+E+F STARTING AT LOCATION
185 *       H'20' (E = NUMBER OF DATA BYTES TRANSMITTED, F = THE NUMBER OF DATA
186 *       BYTES RECEIVED).
187 *-------------------------------------------------------------------------
```

```
188         EJECT
189 *       ##########################################################
190 *       #          START MULTI-MASTER I2C SUBROUTINE             #
191 *       ##########################################################
192 *
193 *ENTRY:  LOAD THE NEXT LOCATIONS IN THE DATA MEMORY:
194 *
195 *         SIOMDR    WITH THE MODE (E.G. #MTMF = MST/TRX WITHOUT POINTER).
196 *         SLVADR    IN THE 7 MSB WITH THE ADDRESS OF THE SLAVE TO BE
197 *                   SELECTED. IN BIT 0 THE READ/WRITE BIT MUST BE LOADED.
198 *         DBCNTR    WITH THE NUMBER OF DATA BYTES TO BE TRANSMITTED OR
199 *                   RECEIVED.
200 *         POINTR    WITH THE POINTER BYTE.
201 *         MTDTR(1)
202 *         UP TO
203 *         MTDTR(E)  WITH UP TO E BYTES TO BE TRANSMITTED.
204 *
205 *         THE MULTI-MASTER SUBROUTINE IS INVOKED BY: CALL MMST
206 *                                                    ---------
207 *
208 *EXIT:   THE LOCATIONS IN DATA MEMORY MRDTR(1) UP TO MRDTR(F) CONTAIN:
209 *                   UP TO F RECEIVED DATA BYTES.
210 *
211 *         THE ACCUMULATOR (A) CONTAINS THE FOLLOWING FLAGS:
212 *         A=H'00'   DATA TRANSMITTED OR RECEIVED CORRECTLY.
213 *         A=H'80'   ARBITRATION LOST AND BYTE COMPLETED WITH CLOCK PULSES.
214 *                   THE BUS IS LEFT IN A RELEASED STATE.
215 *         A=H'40'   BUS OBSTRUCTED; SCL AND/OR SDA STAY LOW; NO TRANSFER
216 *                   POSSIBLE.
217 *         A=H'20'   SPURIOUS START CONDITION; TRANSFER ABORTED AND CONCLUDED
218 *                   WITH A STOP CONDITION.
219 *         A=H'10'   SPURIOUS STOP CONDITION; TRANSFER ABORTED; BUS RELEASED.
220 *         A=H'02'   NO ADDRESS ACKNOWLEDGE FROM SLAVE; TRANSFER CONCLUDED
221 *                   WITH A STOP CONDITION.
222 *         A=H'01'   NO DATA ACKNOWLEDGE FROM SLAVE RECEIVER; TRANSFER
223 *                   CONCLUDED WITH A STOP CONDITION.
224 *
225 *         A COMBINATION OF FLAGS IS POSSIBLE.
226 *
227 *
228 *
229 MUMST1  ASECT   ROM
230         PAGE    256
000000  231         ORG     H'100'
232         ENTRY   MMST
233         IF SLAVE=USED
234         EXTRN   SIV
235         EXTRN   SAR
236 OWNAD   EQU     H'50'           OWN SLAVE ADDRESS IN THE 7 MSB. BIT 0 ="0".
237 *                               (MUST BE ADAPTED TO THE APPLICATION).
238         ENDIF
```

```
                        239         EJECT
                        240 *       MASTER TRANSMITTER/RECEIVER WITH OR WITHOUT POINTER-BYTE.
                        241 *       ========================================================
                        242 *-------------------------------------------------------------------------
                        243 *               TEST BUS STATUS. FORCE STOP CONDITIONS IF OBSTRUCTED.
                        244 *-------------------------------------------------------------------------
000100 95          1    245 MMST    DIS     SI              DISABLE SERIAL INTERRUPT.
000101 BB0A        2    246         MOV     R3,#10          LOAD NUMBER FORCED STOP COND TO FREE THE BUS.
000103 BA64        3    247 MS0     MOV     R2,#BTOC        LOAD BUS TIME-OUT COUNTER.
000105 B985        4    248 MS1     MOV     R1,#133         INITIALIZE 10 MS TEST LOOP.
                        249 *
000107 23FB        5    250 MS2     MOV     A,#.NOT.NXTBN
000109 39          6    251         OUTL    P1,A            TO ENSURE THE CORRECT STATE OF P1.
00010A 09          7    252 MS3     IN      A,P1            FETCH STATUS
00010B B220        8    253         JB5     MS5             JUMP IF A START COND. FROM ANOTHER MASTER
                        254 *                               IS RECEIVED.
00010D 43EC        9    255         ORL     A,#.NOT.(SDA+SCL+BBFN)  TEST IF BUS IS FREE.
00010F 37          10   256         CPL     A
000110 C626        11   257         JZ      MST1            JUMP IF THE BUS IS FREE,SCL AND SDA ARE HIGH.
000112 E907        12   258 MS4     DJNZ    R1,MS2          DECR. AND CHECK TEST LOOP COUNTER.
000114 EA05        13   259         DJNZ    R2,MS1          DECR. AND CHECK TIME OUT COUNTER.
000116 99FD        14   260         ANL     P1,#.NOT.SCL    SET SCL OUTPUT TO LOW.
000118 5474        15   261         CALL    MSSTP           FORCE A STOP CONDITION TO FREE THE BUS.
00011A EB03        16   262         DJNZ    R3,MS0          RETRY TO GET THE BUS.
00011C 2340        17   263         MOV     A,#ERRF2        LOAD ERROR FLAG: BUS OBSTRUCTED.
00011E 24BB        18   264         JMP     MSTER           JUMP TO MASTER ERROR ROUTINE.
                        265 *
000120 3212        19   266 MS5     JB1     MS4             WAIT UNTIL SCL GOES LOW.
000122 1405        20   267         CALL    SIV             CALL SLAVE INTERRUPT VECTOR.
000124 240A        21   268         JMP     MS3             RETRY TO GET THE BUS.
                        269 *-------------------------------------------------------------------------
                        270 *           TRANSMISSION OF START CONDITION+SLAVE ADDRESS+READ/WRITE BIT
                        271 *           RECEPTION OF ACKNOWLEDGE BIT.
                        272 *-------------------------------------------------------------------------
                        273 *
000126 99FE        22   274 MST1    ANL     P1,#.NOT.SDA    OUTPUT START CONDITION.
000128 99F5        23   275         ANL     P1,#.NOT.(SCL+STCHN)  SCL TO LOW AND STRETCHING ENABLED.
00012A B920        24   276         MOV     R1,#SIOMDR      LOCATION OF SIO MODE REG. TO R1.
00012C F1          25   277         MOV     A,@R1           FETCH SIO MODE
00012D AD          26   278         MOV     R5,A            SAVE SIO MODE IN R5.
00012E 5301        27   279         ANL     A,#MRPMF        PREPARE R/WN BIT;ONLY FOR MRPM IT IS INVERTED
000130 19          28   280         INC     R1              LOCATION OF SLAVE ADDRESS REGISTER TO R1.
000131 D1          29   281         XRL     A,@R1           PUT SLAVE ADDRESS AND R/WN BIT IN ACCU.
000132 5400        30   282         CALL    MTDB            TRANSMIT SLAVE ADDRESS + R/WN BIT.
000134 E7          31   283         RL      A               SET FLAGS IN RIGHT POSITION.
000135 67          32   284         RRC     A                 "    "   "   "       "
000136 F2BB        33   285         JB7     MAL             JUMP IF ARBITRATION LOST; OWN ADDR. RECEIVED?
000138 96BB        34   286         JNZ     MSTER           JUMP IF BUS ERROR.
00013A 546E        35   287         CALL    MTAC            RECEIVE ACNOWLEDGE BIT.
00013C 2C          36   288         XCH     A,R4            SET FLAGS IN RIGHT POSITION.
00013D E7          37   289         RL      A                "    "   "   "       "
00013E 4C          38   290         ORL     A,R4            COLLECT THE FLAGS IN ACCU.
00013F 96BB        39   291         JNZ     MSTER           JUMP IF BUS ERROR OR IF NOT-ACKNOWLEDGE.
000141 FD          40   292         MOV     A,R5            FETCH SIO MODE.
000142 729F        41   293         JB3     MST5            JUMP IF MST/TRX WITHOUT POINTER-BYTE.
000144 5284        42   294         JB2     MST4            JUMP IF MST/REC WITHOUT POINTER-BYTE.
```

```
                        295         EJECT
                        296 *       MASTER TRANSMITTER/RECEIVER WITH A POINTER-BYTE.
                        297 *       ===============================================
                        298 *-------------------------------------------------------------------------
                        299 *               TRANSMISSION OF THE POINTER-BYTE.
                        300 *               RECEPTION OF THE ACKNOWLEDGE BIT.
                        301 *-------------------------------------------------------------------------
                        302 *
000146 B923      43     303 MST2    MOV     R1,#POINTR      LOCATION OF THE POINTER-BYTE TO R1.
000148 F1        44     304         MOV     A,@R1           FETCH POINTER BYTE.
000149 5400      45     305         CALL    MTDB            TRANSMIT POINTER BYTE.
00014B E7        46     306         RL      A               SET FLAGS IN ACCU IN RIGHT POSITION.
00014C 67        47     307         RRC     A                "   "   "  "  "   "      "
00014D 96BB      48     308         JNZ     MSTER           JUMP IF BUS ERROR OR IF ARBITRATION LOST.
00014F 546E      49     309         CALL    MTAC            INPUT ACKNOWLEDGE BIT.
000151 4C        50     310         ORL     A,R4            COLLECT FLAGS.
000152 96BB      51     311         JNZ     MSTER           JUMP IF BUS ERROR OR NOT-ACKNOWLEDGE.
                        312 *
                        313 *       MASTER TRANSMITTER/RECEIVER WITH POINTER BYTE.
                        314 *       =============================================
                        315 *-------------------------------------------------------------------------
                        316 *               TRANSMISSION OF A REPEATED START CONDITION
                        317 *                            + SLAVE ADDRESS + READ/WRITE BIT.
                        318 *               RECEPTION OF THE ACKNOWLEDGE BIT.
                        319 *-------------------------------------------------------------------------
                        320 *
000154 8906      52     321 MST3    ORL     P1,#SCL+NXTBN   RELEASE P11/SCL (STILL STRETCHED).
000156 99FB      53     322         ANL     P1,#.NOT.NXTBN  SET SCL TO HIGH.
000158 BAFA      54     323         MOV     R2,#SCLC        LOAD SCL TIME OUT COUNTER.
                        324 *
00015A 09        55     325 MP1     IN      A,P1            FETCH STATUS.
00015B 37        56     326         CPL     A
00015C D264      57     327         JB6     MP2             JUMP IF SCL WENT TO HIGH.
00015E EA5A      58     328         DJNZ    R2,MP1          DECREMENT AND TEST TIME-OUT.
000160 2340      59     329         MOV     A,#ERRF2        LOAD ERROR FLAG.
000162 24BB      60     330         JMP     MSTER           JUMP BECAUSE OF SCL TIME-OUT.
                        331 *
000164 99FE      61     332 MP2     ANL     P1,#.NOT.SDA    OUTPUT REPEATED START CONDITION.
000166 99FD      62     333         ANL     P1,#.NOT.SCL    SET SCL OUTPUT TO LOW.
000168 B921      63     334         MOV     R1,#SLVADR      LOCATION OF SLAVE ADDRESS IN R1.
00016A F1        64     335         MOV     A,@R1           PUT SLAVE ADDRESS + R/WN BIT IN ACCU.
00016B 5400      65     336         CALL    MTDB            TRANSMIT SLAVE ADDRESS + R/WN BIT.
00016D E7        66     337         RL      A               SET FLAGS AT THE RIGHT POSITION.
00016E 67        67     338         RRC     A                "   "   "  "  "   "      "
00016F 96BB      68     339         JNZ     MSTER           JUMP IF BUS ERROR OR ARBITRATION LOST.
000171 546E      69     340         CALL    MTAC            RECEPTION OF THE ACKNOWLEDGE BIT.
000173 2C        70     341         XCH     A,R4
000174 E7        71     342         RL      A               SET ACK FLAG IN THE CORRECT POSITION.
000175 4C        72     343         ORL     A,R4            COLLECT THE FLAGS IN ACCU.
000176 96BB      73     344         JNZ     MSTER           JUMP IF BUS ERROR OR NOT-ACKNOWLEDGE.
000178 FD        74     345         MOV     A,R5            FETCH SIO MODE.
000179 1284      75     346         JB0     MST4            JUMP IF MASTER RECEIVER.
```

```
                        347         EJECT
                        348 *-------------------------------------------------------------------------
                        349 *               TRANSMISSION OF THE POINTER BYTE.
                        350 *-------------------------------------------------------------------------
                        351 *
00017B B922      76     352 MTP     MOV     R1,#DBCNTR      LOCATION OF DATA BYTE COUNTER TO R1.
00017D F1        77     353         MOV     A,@R1           NUMBER OF DATA BYTES IN ACCU.
00017E 17        78     354         INC     A               INCREMENT NUMBER FOR POINTER BYTE.
00017F AD        79     355         MOV     R5,A            TOTAL NUMBER IN R5.
000180 B923      80     356         MOV     R1,#POINTR      LOCATION OF POINTER BYTE TO R1.
000182 24A5      81     357         JMP     MT1             JUMP TO TRANSMISSION OF DATA BYTES.
                        358 *-------------------------------------------------------------------------
                        359 *
                        360 *       MASTER RECEIVER WITH OR WITHOUT POINTER-BYTE.
                        361 *       ============================================
                        362 *-------------------------------------------------------------------------
                        363 *               RECEPTION OF A NUMBER OF DATA BYTES. (NUMBER IN DBCNTR).
                        364 *               TRANSMISSION OF ACKNOWLEDGE BITS.
                        365 *               TRANSMISSION OF NOT-ACKNOWLEDGE AFTER LAST DATA BYTE.
                        366 *-------------------------------------------------------------------------
                        367 *
000184 B922      82     368 MST4    MOV     R1,#DBCNTR      LOCATION OF DBCNTR TO R1.
000186 F1        83     369         MOV     A,@R1           FETCH DATA-BYTE COUNTER.
000187 AD        84     370         MOV     R5,A            SAVE DATA-BYTE COUNTER.
000188 B928      85     371         MOV     R1,#MRDTR1      LOCATION OF FIRST DATA REGISTER IN R1.
00018A 543D      86     372 MR1     CALL    MRDB            INPUT DATA BYTE.
00018C 96BB      87     373         JNZ     MSTER           JUMP IF BUS ERROR.
00018E FC        88     374         MOV     A,R4            FETCH RECEIVED DATA BYTE.
00018F A1        89     375         MOV     @R1,A           SAVE RECEIVED DATA BYTE.
000190 19        90     376         INC     R1              NEXT DATA REGISTER LOCATION.
000191 ED99      91     377         DJNZ    R5,MR2          DECR. AND TEST DATA-BYTE CNTR.
000193 5468      92     378         CALL    MRNAC           OUTPUT NOT-ACK.
000195 96BB      93     379         JNZ     MSTER           JUMP IF BUS ERROR.
000197 24B4      94     380         JMP     MSTSTP          OUTPUT STOP CONDITION.
                        381 *
000199 5462      95     382 MR2     CALL    MRAC            OUTPUT ACK. BIT.
00019B 96BB      96     383         JNZ     MSTER           JUMP IF BUS ERROR.
00019D 248A      97     384         JMP     MR1             JUMP TO RECEIVE NEXT DATA BYTE.
```

```
                        385         EJECT
                        386 *       MASTER TRANSMITTER WITH OR WITOUT POINTER-BYTE.
                        387 *       ==============================================
                        388 *-------------------------------------------------------------------------
                        389 *               TRANSMISSION OF A NUMBER OF DATA BYTES.(NUMBER IN DBCNTR).
                        390 *               RECEPTION OF ACKNOWLEDGE BITS.
                        391 *-------------------------------------------------------------------------
                        392 *
00019F B922       98    393 MST5    MOV     R1,#DBCNTR      LOCATION OF DATA-BYTE COUNTER TO R1.
0001A1 F1         99    394         MOV     A,@R1           FETCH DATA-BYTE COUNTER.
0001A2 AD        100    395         MOV     R5,A            DATA-BYTE COUNTER TO R5.
0001A3 B924      101    396         MOV     R1,#MTDTR1      LOCATION OF FIRST MST/TRX DATA REG. TO R1.
                        397 *
0001A5 F1        102    398 MT1     MOV     A,@R1           FETCH DATA BYTE TO BE TRANSMITTED.
0001A6 5400      103    399         CALL    MTDB            TRANSMIT DATA BYTE.
0001A8 E7        104    400         RL      A               SET FLAGS IN RIGHT POSITION.
0001A9 67        105    401         RRC     A                "    "    "    "      "
0001AA 96BB      106    402         JNZ     MSTER           JUMP IF BUS ERROR OR IF DATA DISTURBED.
0001AC 546E      107    403         CALL    MTAC            INPUT ACKNOWLEDGE BIT.
0001AE 4C        108    404         ORL     A,R4            COLLECT FLAGS.
0001AF 96BB      109    405         JNZ     MSTER           JUMP IF BUS ERROR OR NOT-ACKNOWLEDGE.
0001B1 19        110    406         INC     R1              NEXT MST/TRX DATA REG. LOCATION.
0001B2 EDA5      111    407         DJNZ    R5,MT1          DECR. AND TEST DATA BYTE COUNTER.
                        408 *
                        409 *       STOP CONDITION.
                        410 *       ===============
                        411 *-------------------------------------------------------------------------
                        412 *               TRANSMISSION OF THE STOP CONDITION.
                        413 *-------------------------------------------------------------------------
0001B4 5474      112    414 MSTSTP  CALL    MSSTP           OUTPUT STOP CONDITION.
0001B6 D2BB      113    415         JB6     MSTER           JUMP IF SCL AND/OR SDA STAY LOW; BUS
                        416 *                               OBSTRUCTED.
0001B8 27        114    417 MT2     CLR     A               CLEAR FLAGS.
0001B9 85        115    418         EN      SI              ENABLE SERIAL INTERRUPT.
0001BA 83        116    419         RET                     RETURN TO MAIN PROGRAM.
                        420 *
                        421 *       ARBITRATION LOST DURING TRANSMISSION OF THE SLAVE ADDRESS.
                        422 *       ==========================================================
                        423 *-------------------------------------------------------------------------
                        424 *               IF THE SLAVE SUBROUTINE IS PRESENT AND THE OWN SLAVE ADDRESS
                        425 *               IS RECEIVED, THEN CALL THE SLAVE SUBROUTINE, ELSE GENERATE
                        426 *               THE NINTH CLOCK PULSE AND RELEASE THE BUS.
                        427 *-------------------------------------------------------------------------
                        428         IF SLAVE=USED
                        429 MAL     MOV     R1,A            SAVE FLAGS.
                        430         MOV     A,R4            RECEIVED SLAVE ADDRESS TO ACCU.
                        431         ANL     A,#.NOT.RWB     MASK READ/WRITE BIT.
                        432         XRL     A,#OWNAD        COMPARE WITH OWN SLAVE ADDRESS.
                        433         JZ      MAL1            JUMP IF EQUAL.
                        434         CALL    MTAC            GENERATE NINTH CLOCK PULSE.
                        435         ORL     P1,#H'FF'       RESET SIO HARDWARE, RELEASE SDA.
                        436         ANL     P1,#.NOT.NXTBN  RELEASE SCL.
                        437         JMP     MAL2            PREPARE TO RETURN TO MAIN PROGRAM.
                        438 *
                        439 *
                        440 *
```

```
                        441 *
                        442 MAL1    CALL   SAR             CALL SLAVE SUBROUTINE.
                        443 MAL2    MOV    A,R1            COLLECT FLAGS.
                        444        EN     SI              ENABLE SERIAL INTERRUPT.
                        445        RET                    RETURN TO MAIN PROGRAM.
                        446        ENDIF
                        447 *-------------------------------------------------------------------------
                        448 *
                        449        IF SLAVE=.NOT.USED
                        450 MAL     EQU    $               MASTER ERROR ROUTINE IS ENTERED.
                        451        ENDIF
                        452 *-------------------------------------------------------------------------
                        453 *
                        454 *      MASTER ERROR ROUTINE.
                        455 *      ====================
                        456 *-------------------------------------------------------------------------
                        457 *            IF ARBITRATION IS LOST, THE ACKNOWLEDGE BIT IS
                        458 *            GENERATED.
                        459 *            IN CASE OF A NOT-ACKNOWLEDGE OR A SPURIOUS START
                        460 *            CONDITION, A STOP CONDITION IS GENERATED.
                        461 *            AT A SPURIOUS STOP CONDITION OR A BUS OBSTRUCTION, THE
                        462 *            HARDWARE IS RESET AND SDA AND SCL ARE RELEASED.
                        463 *-------------------------------------------------------------------------
                        464 *
0001BB A9         117   465 MSTER  MOV    R1,A            SAVE FLAGS.
0001BC F2C9       118   466        JB7    MSTER2          JUMP IF ARBITRATION LOST.
0001BE 5323       119   467 MSTER0 ANL    A,#ERRF3+NAACK+NDACK  MASK FLAGS.
0001C0 96CF       120   468        JNZ    MSTER3          JUMP IF STOP COND. MUST BE GENERATED.
0001C2 89FF       121   469 MSTER1 ORL    P1,#H'FF'       RESET SIO HARDWARE, RELEASE SDA.
0001C4 99FB       122   470        ANL    P1,#.NOT.NXTBN  RELEASE SCL.
0001C6 49         123   471        ORL    A,R1            COLLECT FLAGS.
0001C7 85         124   472        EN     SI              ENABLE SERIAL INTERRUPT.
0001C8 83         125   473        RET                    RETURN TO MAIN PROGRAM.
                        474 *
0001C9 546E       126   475 MSTER2 CALL   MTAC            GENERATE ACKNOWLEDGE BIT.
0001CB 49         127   476        ORL    A,R1            COLLECT FLAGS.
0001CC A9         128   477        MOV    R1,A            SAVE FLAGS.
0001CD 24BE       129   478        JMP    MSTER0          JUMP FOR STOP COND. OR RELEASE OF THE BUS.
                        479 *
0001CF 5474       130   480 MSTER3 CALL   MSSTP           OUTPUT STOP CONDITION.
0001D1 D2C2       131   481        JB6    MSTER1          JUMP IF SCL OR/AND SDA STAY LOW. BUS
                        482 *                             OBSTRUCTED.
0001D3 F9         132   483 MSTER4 MOV    A,R1            RESTORE FLAGS.
0001D4 85         133   484        EN     SI              ENABLE SERIAL INTERRUPT.
0001D5 83         134   485        RET                    RETURN TO MAIN PROGRAM.
```

```
                        486         EJECT
0001D6                  487         ORG     H'200'
                        488 *       ############################################################
                        489 *       #     SINGLE BYTE, ACKNOWLEDGE AND STOP SUBROUTINES        #
                        490 *       ############################################################
                        491 *
                        492 *       MASTER TRANSMITTER SINGLE DATA BYTE SUBROUTINE.
                        493 *       ==============================================
                        494 *-------------------------------------------------------------------------
                        495 *THIS SUBROUTINE OUTPUTS 8 BITS OF DATA IN MASTER TRANSMITTER MODE.
                        496 * ENTRY: ACCU CONTAINS DATA BYTE TO BE TRANSMITTED.
                        497 * EXIT: A = 0 AND C = 0; DATA IS TRANSMITTED CORRECTLY
                        498 *                        R4 CONTAINS THE DATA BYTE WHICH IS TRANSMITTED.
                        499 *       A = 0 AND C = 1; ARBITRATION LOST. BYTE HAS BEEN COMPLETED
                        500 *                        WITH CLOCK PULSES ON SCL AND SDA RELEASED.
                        501 *       A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        502 *                 A START CONDITION IS DETECTED.
                        503 *       A = BBFN; STOP CONDITION DETECTED
                        504 *       A = ERRF2; SCL EXCEEDS TIME LIMIT
                        505 *-------------------------------------------------------------------------
000200 BB08      135    506 MTDB    MOV     R3,#8           SET BIT COUNTER
000202 AC        136    507         MOV     R4,A            SAVE DATA BYTE.
000203 BAFA      137    508 MTDB0   MOV     R2,#SCLC        LOAD SCLC TIME-OUT COUNTER.
000205 FC        138    509         MOV     A,R4            FETCH DATA BYTE.
                        510 *
000206 F7        139    511 MTDB1   RLC     A               SHIFT DATA BIT INTO CARRY.
000207 AC        140    512         MOV     R4,A            RESTORE SHIFTED DATA BYTE.
000208 E626      141    513         JNC     MTDB8           JUMP IF DATA BIT = 0.
                        514 *-------------------------------------------------------------------------
                        515 *       TRANSMISSION OF AN "ONE" = HIGH LEVEL ON SDA OUTPUT.
                        516 *-------------------------------------------------------------------------
00020A 8907      142    517 MTDB2   ORL     P1,#SDA+SCL+NXTBN   SET SDA OUTPUT HIGH,
                        518 *                                   RELEASE P11/SCL (BUT STILL STRETCHED).
00020C 99FB      143    519         ANL     P1,#.NOT.NXTBN      SET SCL OUTPUT HIGH.
                        520 *
00020E 09        144    521 MTDB3   IN      A,P1            FETCH STATUS.
00020F D221      145    522         JB6     MTDB7           JUMP IF SCL IS STILL LOW.
000211 99FD      146    523         ANL     P1,#.NOT.SCL    SET SCL OUTPUT TO LOW.
000213 09        147    524         IN      A,P1            FETCH STATUS.
000214 F218      148    525         JB7     MTDB4           JUMP IF "ONE" TRANSMITTED CORRECTLY.
000216 444C      149    526         JMP     MTABL           ARBITRATION LOST. JUMP TO BYTE COMPLETE ROUT.
000218 B21E      150    527 MTDB4   JB5     MTDB6           JUMP IF STA CONDITION RECEIVED.
00021A EB03      151    528         DJNZ    R3,MTDB0        DECR. AND TEST BIT COUNTER.
00021C 4433      152    529         JMP     MTDB10          LAST BIT HAS BEEN TRANSMITTED.
                        530 *
                        531 *
00021E 5330      153    532 MTDB6   ANL     A,#STAF+BBFN    MASK FLAGS.
000220 83        154    533         RET
                        534 *
000221 EA0E      155    535 MTDB7   DJNZ    R2,MTDB3        DECR. AND TEST TIME-OUT COUNTER.
000223 2340      156    536 SCLER   MOV     A,#ERRF2        SET SCL TIME-OUT ERROR FLAG.
000225 83        157    537         RET
```

```
                        538         EJECT
                        539 *
                        540 *-------------------------------------------------------------------------------
                        541 *               TRANSMISSION OF A "ZERO"= LOW LEVEL ON THE SDA OUTPUT.
                        542 *               SPURIOUS START AND STOP CONDITIONS ARE NOT POSSIBLE.
                        543 *               NO BIT DISTURBANCE POSSIBLE.
                        544 *-------------------------------------------------------------------------------
000226 8906      158    545 MTDB8   ORL     P1,#SCL+NXTBN  RELEASE P11/SCL (BUT STILL STRETCHED).
000228 99FE      159    546         ANL     P1,#.NOT.SDA   SET SDA OUTPUT TO LOW.
00022A 99FB      160    547         ANL     P1,#.NOT.NXTBN SET SCL OUTPUT TO HIGH.
                        548 *
00022C 09        161    549 MTDB9   IN      A,P1           FETCH STATUS.
00022D D239      162    550         JB6     MTDB11         JUMP IF SCL IS STILL LOW.
00022F 99FD      163    551         ANL     P1,#.NOT.SCL   SET SCL OUTPUT TO LOW.
000231 EB03      164    552         DJNZ    R3,MTDB0       DEC.AND TEST BIT COUNTER.
                        553 *
000233 FC        165    554 MTDB10  MOV     A,R4           SHIFT LAST TRANSMITTED BIT INTO DATA BYTE.
000234 F7        166    555         RLC     A                "    "       "       "    "    "    "
000235 AC        167    556         MOV     R4,A             "    "       "       "    "    "    "
000236 27        168    557         CLR     A              CLEAR FLAGS
000237 97        169    558         CLR     C                "    "
000238 83        170    559         RET
                        560 *
000239 EA2C      171    561 MTDB11  DJNZ    R2,MTDB9       DECR. AND TEST SCL TIME OUT COUNTER.
00023B 4423      172    562         JMP     SCLER          JUMP IF SCL STAYS LOW.
```

```
                        563         EJECT
                        564 *
                        565 *       MASTER RECEIVER SINGLE DATA BYTE SUBROUTINE.
                        566 *       ============================================
                        567 *-------------------------------------------------------------------------
                        568 *THIS SUBROUTINE INPUTS 8 BITS OF DATA IN MASTER RECEIVER MODE.
                        569 * EXIT: A = 0; DATA IS RECEIVED CORRECTLY
                        570 *              R4 CONTAINS THE DATA BYTE WHICH IS RECEIVED
                        571 *       A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        572 *                 A START CONDITION IS DETECTED.
                        573 *       A = BBFN; STOP CONDITION DETECTED
                        574 *       A = ERRF2; SCL EXCEEDS TIME LIMIT
                        575 *-------------------------------------------------------------------------
                        576 *
00023D BB08      173    577 MRDB    MOV     R3,#8           SET BIT COUNTER
00023F 8901      174    578 MRDB0   ORL     P1,#SDA         SET SDA OUTPUT TO HIGH.
000241 BAFA      175    579 MRDB1   MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
000243 8906      176    580         ORL     P1,#SCL+NXTBN   RELEASE P11/SCL (BUT STILL STRETCHED).
000245 99FB      177    581         ANL     P1,#.NOT.NXTBN SET SCL OUTPUT TO HIGH.
                        582 *
000247 09        178    583 MRDB2   IN      A,P1            FETCH STATUS.
000248 D25A      179    584         JB6     MRDB3           JUMP IF SCL STILL LOW.
00024A 99FD      180    585         ANL     P1,#.NOT.SCL    SET SCL OUTPUT TO LOW.
                        586 *-------------------------------------------------------------------------
                        587 *           FROM HERE THE BYTE COMPLETE ROUTINE (ARBITRATION LOST)
                        588 *           SHARES THE MST/REC DATA BYTE SUBROUTINE.
                        589 *-------------------------------------------------------------------------
00024C F7        181    590 MTABL   RLC     A               SHIFT RECEIVED DATA BIT IN CARRY BIT.
00024D FC        182    591         MOV     A,R4            DATA BYTE TO ACCU.
00024E F7        183    592         RLC     A               SHIFT RECEIVED BIT INTO DATA BYTE.
00024F AC        184    593         MOV     R4,A            SAVE DATA BYTE.
000250 09        185    594         IN      A,P1            FETCH STATUS.
000251 5330      186    595         ANL     A,#STAF+BBFN    MASK FLAGS.
000253 965E      187    596         JNZ     MRDB4           JUMP IF START OR STOP CONDITION DETECTED.
000255 EB41      188    597         DJNZ    R3,MRDB1        DECR. AND TEST BIT COUNTER.
000257 97        189    598         CLR     C
000258 A7        190    599         CPL     C               SET CARRY TO INDICATE ARBITRATION LOST.
000259 83        191    600         RET
                        601 *
00025A EA47      192    602 MRDB3   DJNZ    R2,MRDB2        DECR. SCL TIME-OUT COUNTER AND TEST.
00025C 4423      193    603         JMP     SCLER           JUMP IF SCL STAYS LOW.
                        604 *
00025E 2C        194    605 MRDB4   XCH     A,R4            EXCHANGE FLAGS AND DATA BYTE.
00025F 67        195    606         RRC     A               DELETE LAST RECEIVED BIT FROM DATA BYTE.
000260 2C        196    607         XCH     A,R4            SAVE DATA BYTE IN R4.
000261 83        197    608         RET
```

```
                        609         EJECT
                        610 *       MASTER RECEIVER ACKNOWLEDGE SUBROUTINE.
                        611 *       ======================================
                        612 *-------------------------------------------------------------------------
                        613 *THIS SUBROUTINE OUTPUTS AN ACKNOWLEDGE (LOW) IN MASTER REC. MODE.
                        614 * EXIT: A = 00; ACKNOWLEDGE IS TRANSMITTED CORRECTLY.
                        615 *        A = ERRF2; SCL EXCEEDS TIME LIMIT
                        616 *-------------------------------------------------------------------------
000262 BB01      198    617 MRAC    MOV     R3,#1           SET BIT COUNTER.
000264 BAFA      199    618         MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
000266 4426      200    619         JMP     MTDB8           JUMP TO MST/TRX MODE.
                        620 *
                        621 *
                        622 *       MASTER RECEIVER NOT-ACKNOWLEDGE SUBROUTINE.
                        623 *       ==========================================
                        624 *-------------------------------------------------------------------------
                        625 *THIS SUBROUTINE OUTPUTS A NOT-ACKNOWLEDGE (HIGH) IN MASTER REC. MODE
                        626 * EXIT: A = 00; NOT-ACKNOWLEDGE IS TRANSMITTED CORRECTLY.
                        627 *        A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        628 *                  A START CONDITION IS DETECTED.
                        629 *        A = BBFN; STOP CONDITION DETECTED
                        630 *        A = ERRF2; SCL EXCEEDS TIME LIMIT
                        631 *-------------------------------------------------------------------------
000268 BB01      201    632 MRNAC   MOV     R3,#1           SET BIT COUNTER.
00026A BAFA      202    633         MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
00026C 440A      203    634         JMP     MTDB2           JUMP TO MST/TRX MODE.
                        635 *
                        636 *
                        637 *       MASTER TRANSMITTER INPUT ACKNOWLEDGE SUBROUTINE.
                        638 *       ================================================
                        639 *-------------------------------------------------------------------------
                        640 * EXIT: A = 00; ACKNOWLEDGE BIT IS RECEIVED CORRECTLY
                        641 *                 R4 = 0; ACKNOWLEDGE (LOW) RECEIVED
                        642 *                 R4 = 1; NOT-ACKNOWLEDGE (HIGH) RECEIVED
                        643 *        A = STAF; START CONDITION OR STOP CONDITION FOLLOWED BY
                        644 *                  A START CONDITION IS DETECTED.
                        645 *        A = BBFN; STOP CONDITION DETECTED
                        646 *        A = ERRF2; SCL EXCEEDS TIME LIMIT
                        647 *-------------------------------------------------------------------------
00026E BB01      204    648 MTAC    MOV     R3,#1           SET BIT COUNTER.
000270 BC00      205    649         MOV     R4,#0           CLEAR DATA REGISTER.
000272 443F      206    650         JMP     MRDB0           JUMP TO MST/REC MODE.
```

```
                        651         EJECT
                        652 *
                        653 *       MASTER STOP CONDITION SUBROUTINE.
                        654 *       ================================
                        655 *-------------------------------------------------------------------------
                        656 * SUBROUTINE OUTPUTS STOP CONDITION IN MASTER MODE.
                        657 * EXIT: A = BBFN OR STAF; STOP CONDITION IS TRANSMITTED CORRECTLY.
                        658 *       A = ERRF2; SCL AND/OR SDA STAY LOW AND EXCEED THE TIME LIMIT.
                        659 *-------------------------------------------------------------------------
000274 99FE      207    660 MSSTP   ANL     P1,#.NOT.SDA    SET SDA OUTPUT TO LOW
000276 890E      208    661         ORL     P1,#SCL+NXTBN+STCHN   RELEASE P11/SCL; DISABLE STRETCHING.
000278 99FB      209    662         ANL     P1,#.NOT.NXTBN SET SCL OUTPUT TO HIGH.
00027A BAFA      210    663         MOV     R2,#SCLC        LOAD SCL TIME-OUT COUNTER.
                        664 *
00027C 09        211    665 MSSTP1  IN      A,P1            FETCH STATUS.
00027D D286      212    666         JB6     MSSTP2          JUMP IF SCL STILL LOW.
00027F 8901      213    667         ORL     P1,#SDA         OUTPUT STOP CONDITION.
000281 09        214    668         IN      A,P1            FETCH STATUS.
000282 5330      215    669         ANL     A,#BBFN+STAF    MASK FLAGS.
000284 968C      216    670         JNZ     MSSTP3          JUMP IF A STOP COND. IS TRANSMITTED. A NEW
                        671 *                               START COND. MAY BE RECEIVED.
                        672 *
000286 EA7C      217    673 MSSTP2  DJNZ    R2,MSSTP1       DECR. AND TEST SCL TIME-OUT COUNTER.
000288 8901      218    674         ORL     P1,#SDA         RELEASE SDA.
00028A 2340      219    675         MOV     A,#ERRF2        SET ERROR FLAG; SCL AND/OR SDA STAY LOW.
00028C 83        220    676 MSSTP3  RET
                        677 *-------------------------------------------------------------------------
```

```
                        678         EJECT
                        679 *
                        680         IF SLAVE=.NOT.USED
                        681 *
                        682 ********************************************************
                        683 *       THE NEXT SUBROUTINE SIMULATES THE SLAVE        *
                        684 *       FUNCTION. IT IS DELETED IF A SLAVE SUBROUTINE  *
                        685 *       IS PRESENT.                                    *
                        686 ********************************************************
                        687 *                                                      *
                        688 *       THE SUBROUTINE STARTS AT THE SERIAL INTERRUPT  *
                        689 *       VECTOR AND RELEASES THE SCL LINE IF IT IS      *
                        690 *       STRETCHED AFTER RECEPTION OF A START CONDITION *
                        691 *       FROM ANOTHER MASTER. THIS SUBROUTINE IS        *
                        692 *       ENTERED EITHER BY THE SERIAL INTERRUPT OR FROM *
                        693 *       THE MULTI-MASTER SUBROUTINE.                   *
                        694 *       THE LAST CASE HAPPENS IF AFTER THE SERIAL      *
                        695 *       INTERRUPT IS DISABLED AND BEFORE A START       *
                        696 *       CONDITION IS TRANSMITTED, ANOTHER MASTER       *
                        697 *       OCCUPIES THE BUS.                              *
                        698 *                                                      *
                        699 *------------------------------------------------------*
00028D                  700         ORG     5               SERIAL INTERRUPT VECTOR. *
000005 0410      221    701 SIV     JMP     RSCL                                    *
000007                  702         ORG     H'10'                                   *
000010 89FF      222    703 RSCL    ORL     P1,#H'FF'       RESET SIO HARDWARE.     *
000012 99FB      223    704         ANL     P1,#.NOT.NXTBN FINISH STRETCHING SCL    *
000014 93        224    705         RETR                                            *
                        706 *------------------------------------------------------*
                        707 ********************************************************
                        708 *
                        709         ENDIF
000015                  710         END

  NO ERRORS DETECTED
```

15.0 INSTRUCTION SET

The MAB8422/42 instruction set consists of over eighty, one and two byte instructions. It is identical to the MAB84X1 instruction set, except that the instructions MOV A,Sn, MOV Sn,A and MOV Sn,#data, servicing the serial port, are not used. Program code efficiency is high because all RAM locations on a 256 byte page require only a single byte address.

Table 6 gives the instruction set of the MAB8422/42; Table 7 shows the instruction map. The following symbols and abbreviations are used.

Symbol	description
A	accumulator
addr	program memory address
Bb	bit designation (b = 0-7)
RBS	register bank select
C	carry (bit CY)
CNT	event counter
D	mnemonic for 4-bit digit (nibble)
data	8-bit number or expression
I	interrupt
MB	memory bank
MBFF	memory bank flip-flop
P	mnemonic for 'in-page' operation
PC	program counter
Pp	port designation (p = 0,1,2)
PSW	program status word
RB	register bank
Rr	register designation (r = 0-7)
S_n	serial I/O register
SP	stack pointer
T	timer
TF	timer flag
T1	test 1 input
T0	test 0 input
#	immediate data prefix
@	indirect address prefix
(X)	contents of X
((X))	contents of location addressed by X
←	is replaced by
↔	is exchanged with

Table 6 Instruction set

	mnemonic	opcode (hex.)	bytes/ cycles	description	function		notes
ACCUMULATOR	ADD A, Rr	6*	1/1	Add register contents to A	(A)←(A) + (Rr)	r = 0–7	1
	ADD A, @Rr	60 61	1/1	Add RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) (A)←(A) + ((R1))		1
	ADD A, #data	03 data	2/2	Add immediate data to A	(A)←(A) + data		1
	ADDC A, Rr	7*	1/1	Add carry and register contents to A	(A)←(A) + (Rr) + (C)	r = 0–7	1
	ADDC A, @Rr	70 71	1/1	Add carry and RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) + (C) (A)←(A) + ((R1)) + (C)		1
	ADDC A, #data	13 data	2/2	Add carry and immediate data to A	(A)←(A) + data + (C)		1
	ANL A, Rr	5*	1/1	'AND' Rr with A	(A)←(A) AND (Rr)	r = 0–7	
	ANL A, @Rr	50 51	1/1	'AND' RAM data, addressed by Rr, with A	(A)←(A) AND ((R0)) (A)←(A) AND ((R1))		
	ANL A, #data	53 data	2/2	'AND' immediate data with A	(A)←(A) AND data		
	ORL A, Rr	4*	1/1	'OR' Rr with A	(A)←(A) OR (Rr)	r = 0–7	
	ORL A, @Rr	40 41	1/1	'OR' RAM data, addressed by Rr, with A	(A)←(A) OR ((R0)) (A)←(A) OR ((R1))		
	ORL A, #data	43 data	2/2	'OR' immediate data with A	(A)←(A) OR data		
	XRL A, Rr	D*	1/1	'XOR' Rr with A	(A)←(A) XOR (Rr)	r = 0–7	
	XRL A, @Rr	D0 D1	1/1	'XOR' RAM, addressed by Rr, with A	(A)←(A) XOR ((R0)) (A)←(A) XOR ((R1))		
	XRL A, #data	D3 data	2/2	'XOR' immediate data with A	(A)←(A) XOR data		
	INC A	17	1/1	increment A by 1	(A)←(A) + 1		
	DEC A	07	1/1	decrement A by 1	(A)←(A) – 1		
	CLR A	27	1/1	clear A to zero	(A)←0		
	CPL A	37	1/1	one's complement A	(A)←NOT(A)		
	RL A	E7	1/1	rotate A left	$(A_{n+1})\leftarrow(A_n)$ $(A_0)\leftarrow(A_7)$	n = 0–6	

Table 6 (continued)

	mnemonic	opcode (hex.)	bytes/ cycles	description	function		notes
ACCUMULATOR (cont.)	RLC A	F7	1/1	rotate A left through carry	$(A_{n+1})\leftarrow A_n$ $(A_0)\leftarrow(C)$, $(C)\leftarrow(A_7)$	n = 0–6	2
	RR A	77	1/1	rotate A right	$(A_n)\leftarrow(A_{n+1})$ $(A_7)\leftarrow(A_0)$	n = 0–6	
	RRC A	67	1/1	rotate A right through carry	$(A_n)\leftarrow(A_{n+1})$ $(A_7)\leftarrow(C)$, $(C)\leftarrow(A_0)$	n = 0–6	2
	DA A	57	1/1	decimal adjust A			2
	SWAP A	47	1/1	swap nibbles of A	$(A_{4-7})\leftrightarrow(A_{0-3})$		
DATA MOVES	MOV A, Rr	F*	1/1	move register contents to A	(A)←(Rr)	r = 0–7	
	MOV A, @Rr	F0 F1	1/1	move RAM data, addressed by Rr, to A	(A)←((R0)) (A)←((R1))		
	MOV A, #data	23 data	2/2	move immediate data to A	(A)←data		
	MOV Rr, A	A*	1/1	move accumulator contents to register	(Rr)←(A)	r = 0–7	
	MOV @Rr, A	A0 A1	1/1	move accumulator contents to RAM location addressed by Rr	((R0))←(A) ((R1))←(A)		
	MOV Rr, #data	B* data	2/2	move immediate data to Rr	(Rr)←data		
	MOV @Rr, #data	B0 data B1 data	2/2	move immediate data to RAM location addressed by Rr	((R0))←data ((R1))←data		
	XCH A, Rr	2*	1/1	exchange accumulator contents with Rr	(A)↔(Rr)	r = 0–7	
	XCH A, @Rr	20 21	1/1	exchange accumulator contents with RAM data addressed by Rr	(A)↔((R0)) (A)↔((R1))		
	XCHD A, @Rr	30 31	1/1	exchange lower nibbles of A and RAM data addressed by Rr	$(A_{0-3})\leftrightarrow((R0_{0-3}))$ $(A_{0-3})\leftrightarrow((R1_{0-3}))$		
	MOV A, PSW	C7	1/1	move PSW contents to accumulator	(A)←(PSW)		
	MOV PSW, A	D7	1/1	move accumulator bit 3 to PSW_3	$(PSW_3)\leftarrow(A_3)$		3
	MOVP A, @A	A3	1/2	move indirectly addressed data in current page to A	$(PC_{0-7})\leftarrow$(A), (A)←((PC))		

FLAGS	CLR C	97	1/1	clear carry bit	(C)←0		2
	CPL C	A7	1/1	complement carry bit	(C)←NOT(C)		2
REGISTER	INC Rr	1*	1/1	increment register by 1	(Rr)←(Rr) + 1	r = 0–7	
	INC @Rr	10 11	1/1	increment RAM data, addressed by Rr, by 1	((R0))←((R0)) + 1 ((R1))←((R1)) + 1		
	DEC Rr	C*	1/1	decrement register by 1	(Rr)←(Rr) – 1	r = 0–7	
	DEC @Rr	C0 C1	1/1	decrement RAM data, addressed by Rr, by 1	((R0))←((R0)) – 1 ((R1))←((R1)) – 1		
BRANCH	JMP addr	●4 address	2/2	unconditional jump within a 2Kbank	(PC_{8-10})←$addr_{8-10}$ (PC_{0-7})←$addr_{0-7}$ (PC_{11-12})←MBFF 0–1		
	JMPP @A	B3	1/2	indirect jump within a page	(PC_{0-7})←((A))		
	DJNZ Rr, addr	E* address	2/2	decrement Rr by 1 and jump if not zero to addr	(Rr)←(Rr) – 1 if (Rr) not zero (PC_{0-7})←addr	r = 0–7	
	DJNZ @Rr, addr	E0	2/2	decrement RAM data, addressed by Rr, by 1 and jump if not zero to addr	((R0))←((R0)) – 1 if ((R0)) not zero (PC_{0-7})←addr		
		E1			((R1))←((R1)) – 1 if ((R1)) not zero (PC_{0-7})←addr		
	JBb addr	▲2 address	2/2	jump to addr if Acc. bit b = 1	if b = 1: (PC_{0-7})←addr	b = 0–7	
	JC addr	F6 address	2/2	jump to addr if C = 1	if C = 1: (PC_{0-7})←addr		
	JNC addr	E6 address	2/2	jump to addr if C = 0	if C = 0: (PC_{0-7})←addr		
	JZ addr	C6 address	2/2	jump to addr if A = 0	if A = 0: (PC_{0-7})←addr		
	JNZ addr	96 address	2/2	jump to addr if A is NOT zero	if A≠0: (PC_{0-7})←addr		
	JT0 addr	36 address	2/2	jump to addr if T0 = 1	if T0 = 1: (PC_{0-7})←addr		
	JNT0 addr	26 address	2/2	jump to addr if T0 = 0	if T0 = 0: (PC_{0-7})←addr		
	JT1 addr	56 address	2/2	jump to addr if T1 = 1	if T1 = 1: (PC_{0-7})←addr		
	JNT1 addr	46 address	2/2	jump to addr if T1 = 0	if T1 = 0: (PC_{0-7})←addr		
	JTF addr	16 address	2/2	jump to addr if Timer Flag = 1	if TF = 1: (PC_{0-7})←addr		4
	JNTF addr	06 address	2/2	jump to addr if Timer Flag = 0	if TF = 0: (PC_{0-7})←addr		

Table 6 (continued)

	mnemonic	opcode (hex.)	bytes/ cycles	description	function	notes
TIMER/EVENT COUNTER	MOV A, T	42	1/1	move timer/event counter contents to accumulator	(A)←(T)	
	MOV T, A	62	1/1	move accumulator contents to timer/event counter	(T)←(A)	
	STRT CNT	45	1/1	start event counter		
	STRT T	55	1/1	start timer		
	STOP TCNT	65	1/1	stop timer/event counter		
	EN TCNTI	25	1/1	enable timer/event counter interrupt		
	DIS TCNTI	35	1/1	disable timer/event counter interrupt		
CONTROL	EN I	05	1/1	enable external interrupt		
	DIS I	15	1/1	disable external interrupt		
	SEL RB0	C5	1/1	select register bank 0	(RBS)←0	5
	SEL RB1	D5	1/1	select register bank 1	(RBS)←1	5
	SEL MB0	E5	1/1	select program memory bank 0	(MBFF0)←0, (MBFF1)←0	
	SEL MB1	F5	1/1	select program memory bank 1	(MBFF0)←1, (MBFF1)←0	
	SEL MB2	A5	1/1	select program memory bank 2	(MBFF0)←0, (MBFF1)←1	
	SEL MB3	B5	1/1	select program memory bank 3	(MBFF0)←1, (MBFF1)←1	
SUBROUTINE	CALL addr	▲4 address	2/2	jump to subroutine	((SP))←(PC), ($PSW_{4, 6, 7}$) (SP)←(SP) + 1 (PC_{8-10})←$addr_{8-10}$ (PC_{0-7})←$addr_{0-7}$ (PC_{11-12})←MBFF 0–1	6
	RET	83	1/2	return from subroutine	(SP)←(SP) – 1 (PC)←((SP))	6
	RETR	93	1/2	return from interrupt and restore bits 4, 6, 7 of PSW	(SP)←(SP) – 1 ($PSW_{4, 6, 7}$) + (PC)←((SP))	6

PARALLEL INPUT/OUTPUT	IN A, Pp	08 09 0A	1/2	input port p data to accumulator	(A)←(P0) (A)←(P1) (A)←(P2)	7
	OUTL Pp, A	38 or 90 39 3A	1/2	output accumulator data to port p	(P0)←(A) (P1)←(A) (P2)←(A)	
	ANL Pp, #data	98 data 99 data 9A data	2/2	AND port p data with immediate data	(P0)←(P0) AND data (P1)←(P1) AND data (P2)←(P2) AND data	
	ORL Pp, #data	88 data 89 data 8A data	2/2	OR port p data with immediate data	(P0)←(P0) OR data (P1)←(P1) OR data (P2)←(P2) OR data	
SERIAL INPUT/OUTPUT	EN SI	85	1/1	enable serial I/O interrupt		
	DIS SI	95	1/1	disable serial I/O interrupt		
	NOP	00	1/1	no operation		

Notes to Table 6.

1. PSW CY, AC affected
2. PSW CY affected
3. PSW PS affected
4. Execution of JTF and JNTF instructions resets the Timer Flag (TF).
5. PSW RBS affected
6. PSW SP_0, SP_1, SP_2 affected
7. (A) = 11111, P22, P21, P20.

* : 8, 9, A, B, C, D, E, F
● : 0, 2, 4, 6, 8, A, C, E
▲ : 1, 3, 5, 7, 9, B, D, F

Table 7 Instruction map

MAB8422/8442 INSTRUCTION MAP

first hexadecimal character of opcode

second hexadecimal character of opcode

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP			ADD A,#data	JMP page 0	EN I	JNTF addr	DEC A	IN A,Pp							
1	INC @Rr		JBO addr	ADDC A,#data	CALL page 0	DIS I	JTF addr	INC A	INC Rr							
2	XCH A,@Rr			MOV A,#data	JMP page 1	EN TCNTI	JNTO addr	CLR A	XCH A,Rr							
3	XCHD A,@Rr		JB1 addr		CALL page 1	DIS TCNTI	JTD addr	CPL A	OUTL Pp,A							
4	ORL A,@Rr		MOV A,T	ORL A,#data	JMP page 2	STRT CNT	JNT1 addr	SWAP A	ORL A,Rr							
5	ANL A,@Rr		JB2 addr	ANL A,#data	CALL page 2	STRT T	JT1 addr	DA A	ANL A,Rr							
6	ADD A,@Rr		MOV T,A		JMP page 3	STOP TCNT		RRC A	ADD A,Rr							
7	ADDC A,@Rr		JB3 addr		CALL page 3			RR A	ADDC A,Rr							
8				RET	JMP page 4	EN SI			ORL Pp,#data							
9	OUTL P0,A		JB4 addr	RETR	CALL page 4	DIS SI	JNZ addr	CLR C	ANL Pp,#data							
A	MOV @Rr,A			MOVP A,@A	JMP page 5	SEL MB2		CPL C	MOV Rr,A							
B	MOV @Rr,#data		JB5 addr	JMPP @A	CALL page 5	SEL MB3			MOV Rr,#data							
C	DEC @Rr				JMP page 6	SEL RB0	JZ addr	MOV A,PSW	DEC Rr							
D	XRL A,@Rr		JB6 addr	XRL A,#data	CALL page 6	SEL RB1		MOV PSW,A	XRL A,Rr							
E	DJNZ @Rr,addr				JMP page 7	SEL MB0	JNC addr	RL A	DJNZ Rr, addr							
F	MOV A,@Rr		JB7 addr		CALL page 7	SEL MB1	JC addr	RLC A	MOV A,Rr							

6. The PCF84CXX microcontroller family

CONTENTS -- THE PCF84CXX MICROCONTROLLER FAMILY

		page
1.0	DESCRIPTION	6-4
2.0	FEATURES	6-4
3.0	PACKAGE OUTLINES	6-4
3.1	Pin designation for PCF84C21/41/81	6-7
3.2	Pin designation for PCF84C00B ROM-less piggy-back version	6-8
3.3	Pin designation for PCF84C00T ROM-less version (VSO-56 package)	6-9
4.0	FUNCTIONAL DESCRIPTION	6-11
4.1	Program memory (ROM)	6-11
4.2	Data memory (RAM)	6-11
4.3	Derivative registers	6-14
4.4	Input/Output	6-15
4.4.1	Parallel ports	6-15
4.4.2	Serial I/O	6-17
4.5	Interrupts	6-18
4.5.1	External interrupts	6-18
4.5.2	SIO interrupts	6-18
4.5.3	Timer/counter interrupts	6-18
4.5.4	Simultaneous interrupts	6-18
4.5.5	Interrupt logic	6-21
4.5.6	Interrupt program examples	6-24
4.5.7	Timer interrupt logic	6-24
4.5.8	Exiting the IDLE mode by an interrupt	6-24
4.6	Test inputs T1, $\overline{\text{INT}}$/T0	6-24
4.6.1	Test input T1	6-24
4.7.2	Test input $\overline{\text{INT}}$/T0	6-25
4.7	Oscillator and clock	6-25
4.8	Timer/event counter	6-26
4.9	Program status word	6-28
4.10	Program counter	6-28
4.11	Central processing unit	6-29
4.12	Reset	6-30
4.12.1	Power-on-reset	6-30
4.13	Connecting the PCF84CXX in a typical system	6-34
4.14	Instruction set	6-35
5.0	IDLE AND STOP MODES	6-42
5.1	IDLE mode	6-42
5.2	STOP mode	6-43
6.0	TIMING	6-44

1.0 DESCRIPTION

The PCF84CXX family of microcontrollers is manufactured in CMOS technology. The family consists of the following devices:

- PCF84C00 - 256 RAM bytes, external program memory
- PCF84C21 - 2 K ROM/64 RAM bytes
- PCF84C41 - 4 K ROM/128 RAM bytes
- PCF84C81 - 8 K ROM/256 RAM bytes

} LED drive capability via Port 1

Each version has 20 quasi-bidirectional I/O port lines, one serial I/O line, one single-level vectored interrupt, an 8-bit timer event counter and on-board clock oscillator and clock circuits.

This microcontroller family is an efficient controller as well as an arithmetic processor. These microcontrollers have bit handling abilities and facilities for both binary and BCD arithmetic.

2.0 FEATURES

- 8-bit CPU, ROM, RAM, I/O in a single 28-lead DIL or SO package
- 2 K, 4 K or 8 K ROM bytes plus a ROM-less version
- 64, 128 or 256 RAM bytes
- 20 quasi-bidirectional I/O port lines
- Two test inputs: one of which is also the external interrupt input
- Single-level vectored interrupts: external, timer/event counter, serial I/O
- I^2C hardware interface which can be used in single or multi-master systems (serial I/O data via an existing port line and clock via a dedicated line)
- 8-bit programmable timer/event counter
- Clock frequency 100 kHz to 10 MHz
- Over 80 instructions (similar to MAB8048) all of 1 or 2 cycles
- Single supply voltage from 2,5 V to 5,5 V
- IDLE and STOP mode
- Power-on-reset circuit
- Operating temperature range: -40 to + 85 ^{o}C
- High current output on P1 (I_{OL} = 10 mA at V_{DL} = 1.2 V), not PCF84C00

3.0 PACKAGE OUTLINES

PCF84C21/41/81P: 28-lead DIL; plastic (SOT-117D).
PCF84C21/41/81T: 28-lead mini-pack; plastic (SO-28; SOT-136A).
PCF84C00B : 28-lead 'Piggy-back' package (with up to 28-pin EPROM on top)
PCF84C00T : 56-lead mini-pack; plastic (VSO-56; SOT-190)

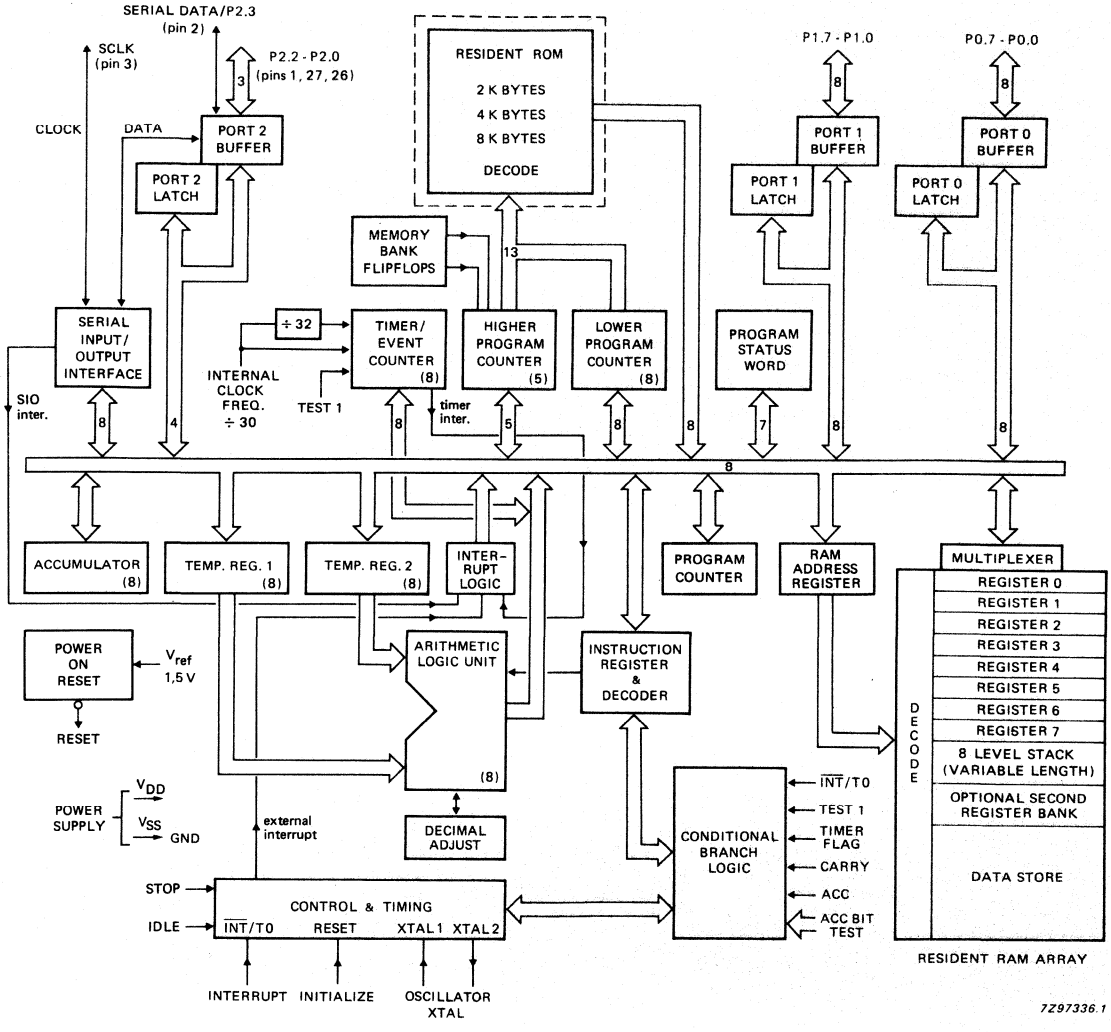
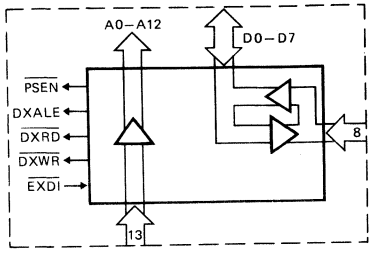
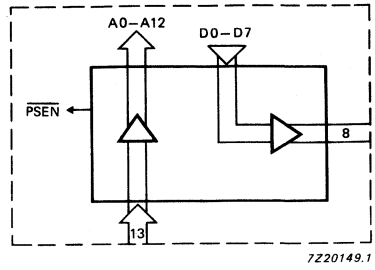


Fig. 1(a) Block diagram of the PCF84CXX



(a)

Fig. 1(b) Replacement of dotted part in Fig. 1(a) for the PCF84C00T ROM-less version.



(b)

Fig. 1(c) Replacement of dotted part in Fig. 1(a) for the PCF84CXX ROMless 'piggy-back' version.

3.1 Pin designation for PCF84C21/41/81

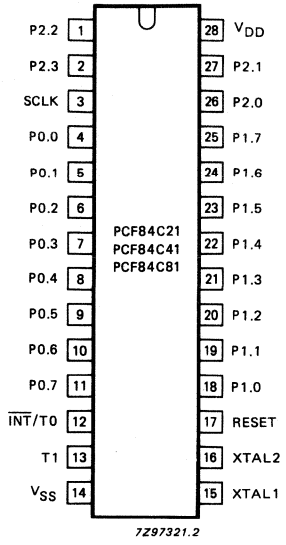


Fig. 2 Pinning diagram PCF84C21/41/81.

PIN DESIGNATION

Pin	symbol	type	function
3	SCLK	I/O	Clock: bidirectional clock for serial I/O.
4-11	P00-P07	I/O	Port 0: 8-bit quasi-bidirectional I/O port.
12	$\overline{INT}$/T0	I	Interrupt/Test 0: external interrupt input (sensitive to negative-going edge)/test input pin; when used as a test input directly tested by conditional branch instructions JT0 and JNT0.
13	T1	I	Test 1: test input pin, directly tested by conditional branch instructions JT1 and JNT1. T1 also functions as an input to the 8-bit timer/event countertimer/event counter, using the STRT CNT instruction.
14	V_{SS}	I	Ground: circuit earth potential.
15	XTAL 1	I	Oscillator input: crystal which determines the internal oscillator frequency or the external clock generator.
16	XTAL 2	I/O	Connection to other side of timing component
17	RESET	I/O	Reset input: used to initialize the processor (active HIGH), or output of power-on-reset circuit.
18-25	P10-P17	I/O	Port 1: 8-bit quasi-bidirectional I/O port.
26, 27, 1,2	P20-P23	I/O	Port 2: 4-bit quasi-bidirectional I/O port. P23 is the serial data input/output in serial I/O mode.
28	V_{DD}	I	Power supply: 2,5 V to 5,5 V.

3.2 Pin designation of PCF84C00B piggy-back version

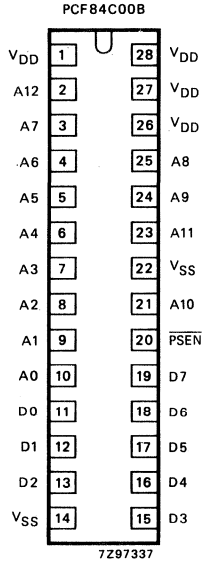


Fig. 3 Pinning diagram: PCF84C00B 'Piggy-back' version top pinning; to access a 2732 or 2764 EPROM.

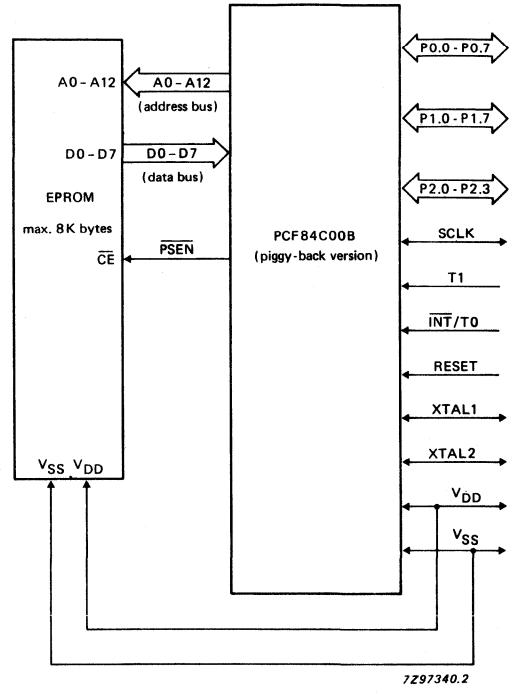


Fig. 3(a) Connection of EPROM to 'Piggy-back' package PCF84C00B.

The PCF84C00B is chiefly used in prototyping and low volume production. The device is mounted in a 'piggy-back' package, i.e. a 4 K or 8 K byte EPROM may be mounted in the 24/28 pin socket on top of the package.

Notes

1. RAM capacity of PCF84C00B is 256 bytes.
2. Access time for ROMS/EPROMS to be below 7 x $1 / f_{XTAL}$.
3. Bottom pinning is identical to that of Fig. 2.

3.3 Pin designation of PCF8400T

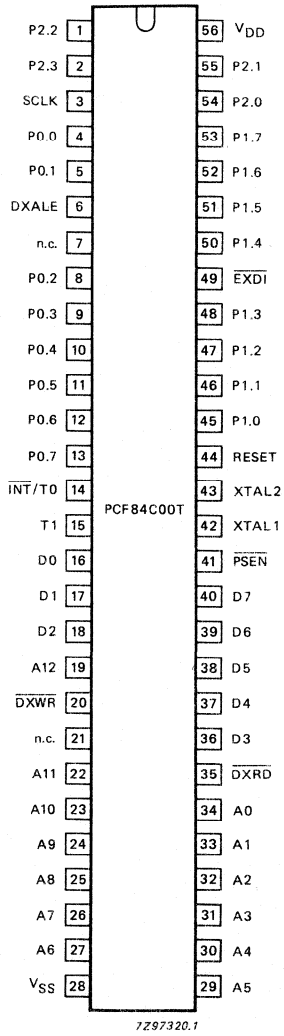


Fig. 4 Pinning diagram; ROMless version PCF84C00T.

The PCF84C00T is chiefly used for prototyping future derivatives of the PCF84CXX family or low-volume production. The device is packaged in a 56-lead VSO outline. Additional signals are available (see pinning information below) to control external program memory and derivative functions.

PIN DESIGNATION

Pin	symbol	type	function
34-29 27-22,19	A00-A12	O	Address outputs
16-18 36-40	D0-D7	I/O	Data lines *
41	$\overline{PSEN}$	O	Program store enable (active low): Used for enabling external program memory. Active during TS9 and TS10 of each machine cycle and TS1 of each following cycle. $\overline{PSEN}$ is HIGH during STOP mode.
6	DXALE	O	Address latch enable: Using the falling edge, the Dx address can be latched in an appropriate external latch. This signal occurs only during execution of the MOV Dx,A, MOV A,Dx, ANL Dx,A and ORL Dx,A instructions, with x = 0 to FF H.
35	$\overline{DXRD}$	O	Read strobe (active LOW): When this signal is active, external registers emulating Dx registers can be enabled to the data bus of the PCF84C00. This signal occurs only during execution of MOV A,Dx, ANL Dx,A and ORL Dx,A instructions, with x = 0 to FF H.
20	$\overline{DXWR}$	O	Write strobe (active low): On the rising edge, data on D0-D7 can be latched in appropriate external registers emulating Dx. This signal occurs only during execution of MOV Dx,A, ANL Dx,A and ORL Dx,A instructions, with x = 0 to FF H.
49	$\overline{EXDI}$	I	External derivative interrupt (active low): $\overline{EXDI}$ is 'OR-ed' with the internal serial interrupt and can be used to initiate an interrupt from external hardware emulating derivative functions. An internal pull-up is provided. A derivative interrupt is internally polled during time slot TS6**, and is only accepted if an EN SI instruction has been executed and the device is not already in an interrupt routine. Derivative interrupts are not latched in the PCF84C00.

* The specified STOP mode supply current can only be reached if external pull-ups are connected to all data lines.

** The interrupt signal must be continually active until the vector address 05H is present on the address bus.

4.0 FUNCTIONAL DESCRIPTION

4.1 Program memory (ROM)

The program memory consists of 1024, 2048, 4096 or 8192 bytes (8-bit words), which are addressed by the program counter. The memory is mask-programmed at production and because the PCF84CXX family offers a range of ROM capacities to suit the application, ROM expansion is not required. Figure 5 shows the program memory map.

Four program memory locations are of special importance:

location 0 - contains the first instruction to be executed after the processor is initialized (RESET)
location 3 - contains the vector of an external interrupt service subroutine
location 5 - contains the vector of a serial I/O interrupt service subroutine
location 7 - contains the vector of a timer/event counter interrupt service subroutine

Program memory is arranged in banks of 2 K bytes, organized in pages of 256 bytes and these are selected by using the SEL MB instructions. Only the unconditional branch instructions (JMP and CALL) can cause jumps over page boundaries. Memory bank boundaries may only be crossed by using the same unconditional branch instructions after the appropriate memory bank has been selected. A CALL instruction transfers control to a subroutine on any page. RET and RETR instructions transfer control from a subroutine back to the main program.

Note: if a memory bank change is necessary, the required bank must be selected prior to a CALL or JMP instruction.

4.2 Data memory (RAM)

Data memory consists of 64, 128 or 256 bytes and all locations are indirectly addressable using RAM pointer registers; for this, up to 16 designated locations are directly addressable. Data memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Figure 6 shows the data memory map.

Location 0 to 7 are designated as working registers, directly addressable by the direct register instructions. Because these registers are easily addressed and require the minimum of instruction bytes to manipulate their contents, they are used to store frequently accessed intermediate results. This bank of registers can be selected by the SEL RB0 instruction.

Executing the select register bank instruction SEL RB1, designates locations 24 to 31 as the working registers in place of locations 0 to 7. These are then also directly addressable. This second bank of working registers may be used as an extension of the first or reserved for use during interrupt service subroutines, saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first two locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations.

All RAM locations make efficient program loop counters when used with the decrement register and test instruction DJNZ.

Locations 8 to 23 may be designed as an 8-level program counter stack (2 locations per level), or as general purpose RAM. The program counter stack (Fig. 7) enables the processor to keep track of the return addresses and status generated by interrupts or CALL instructions by storing the contents of the program counter prior to servicing the subroutine. A 3-bit stack pointer determines which of the program counter stack's eight register pairs will be loaded with the next return address generated.

The stack pointer, when initialized to 000 by RESET, points to RAM locations 8 and 9. On the first subroutine CALL or interrupt, the contents of the program counter and bits 4, 6 and 7 of the program status word (PSW) are transferred to locations 8 and 9. The stack pointer increments by one and points to locations 10 and 11 ready for another call. Because an address may be up to 13 bits long, two bytes must be used to store each address.

At the end of a subroutine, which is signalled by a return instruction (RET), the stack pointer decrements by one and the contents of the register pair on top of the stack are transferred to the program counter. The saved PSW bits are transferred to the PSW only by the RETR instruction.

If not all 8 levels of subroutine and interrupt nesting are used, the unused portion of the stack may be used as indirectly addressable RAM. Locations 32 to max 255 may be used for storage of program variables or data.

Nesting of subroutines within subroutines may continue until overflow of the stack. If an overflow does occur, the deepest address (location 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The value of the saved contents of the program counter is different for an interrupt CALL compared to a normal CALL to subroutine. With an interrupt CALL, the program counter return address is saved; with a subroutine CALL, the saved program counter value is one less than the program counter return address.

Figures 5 and 6 show the program memory and data memory maps. Fig. 7 illustrates the structure of the program counter stack.

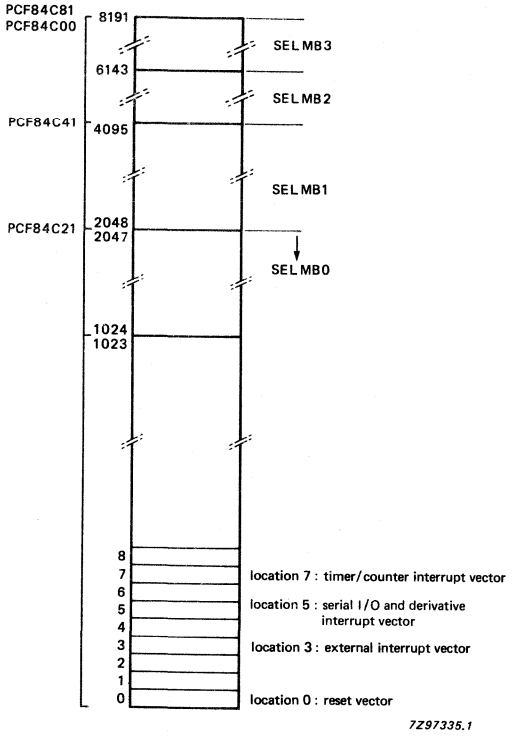


Fig. 5 Program memory map.

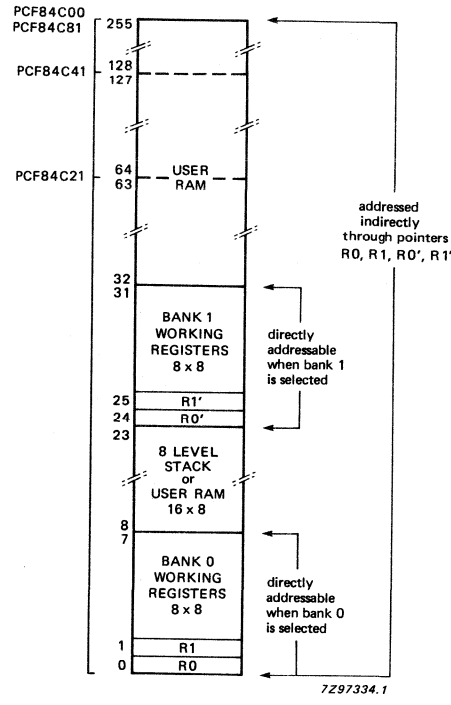


Fig. 6 Data memory map.

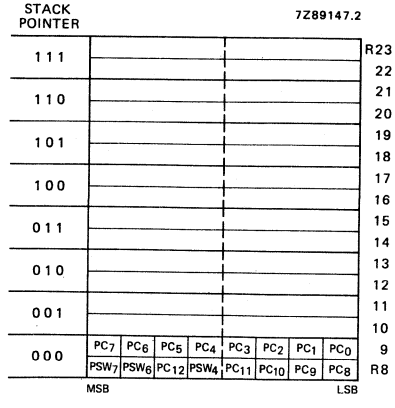


Fig. 7 Program counter stack.

4.3 Derivative registers

Using the instructions MOV A,Dx, MOV Dx,A, ANL Dx,A and ORL Dx,A, the PCF84C00T can perform immediate input/output on up to 256 external 8-bit derivative registers. A typical interface is shown in Fig. 8 below. The PCF84C00T has interface signals DXALE, $\overline{\text{DXWR}}$, $\overline{\text{DXRD}}$ and $\overline{\text{EXDI}}$ to control the derivative registers for emulation and prototyping purposes. Future PCF84CXX versions will control derivative ports and external derivative registers. The contents of a derivative register may be moved to and from the accumulator. Also a logic OR and a logic AND with the accumulator may be performed with the derivative register as the destination.

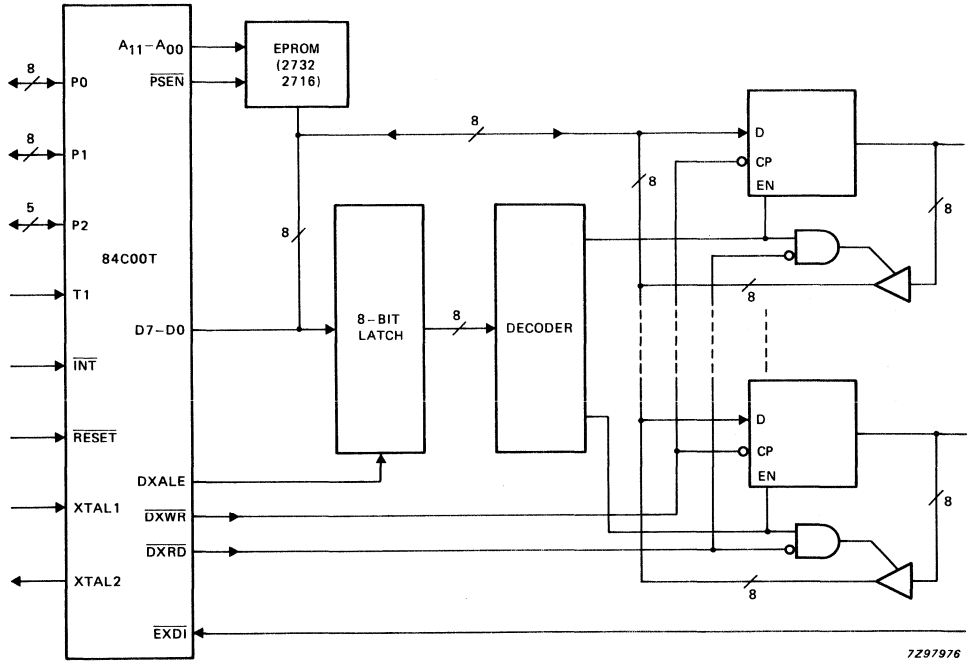


Fig. 8 Block digram of the external Dx interface. The Dx interface can only be used with the PCF84C00T.

4.4 Input/Output -see also MAB84X1 microcontroller family

4.4.1 Parallel

The I/O ports of the PCF84CXX are optimized for low current consumption.

Output data written to P0, P1 or P2 is latched and remains unchanged until rewritten. Input lines are fully CMOS compatible, output lines can drive one standard TTL load.

Figure 9(a) shows the quasi-bidirectional I/O interface with push-pull output and switched pull-up current source. Each line is pulled up to V_{DD} via a constant current source (TR4), this pull-up is enabled whenever one of the two output latches contain a logic 1. This current is sufficient as source for a TTL HIGH level, yet can be pulled LOW by an external CMOS device, this allows the same pin to be used for both input and output.

To provide fast switching times during a 0-to-1 transition, transistor TR2 is switched on (during the length of the internal write pulse, 1 oscillator period) whenever a 1 is written to the port line for the first time (MQ = 1, SQ = 0). Subsequent writing of a 1 to the line will not turn on TR2, this ensures that excessive current through external components is prevented.

When a 0 is written to the line, TR3 is switched off removing pull-up current, transistor TR4 is now switched on and provides current sink capability. When using the line as an input, a 1 must be initially written to the line otherwise the pull-down transistor TR1 will remain low impedance.

After RESET all I/O lines are in input mode. The logic 1 on these lines may be easily pulled down by CMOS or TTL components.

The PCF84CXX family offers the possibility to individually select 19 of the 20 parallel port pins (not P23), using the following mask options:

Option 1-STANDARD PORT; quasi-bidirectional I/O with switched pull-up current source of 100 uA (typ.) and P-channel booster transistor TR2. TR2 is only active during 1 clock cycle (Fig. 9 (a)).

Option 2-OPEN DRAIN; quasi-bidirectional I/O with only an N-channel open drain output. Application as an output requires connection of an external pull-up resistor (Fig. 9 (b)).

Option 3-PUSH-PULL OUTPUT; drive capability of the output will be 1,6 mA (min.) at V_{DD} = 5 V $\pm$10% in both polarities. To avoid a large current flowing through the output transistors during the input mode, these push-pull pins must only be used as outputs (Fig. 9 (c)).

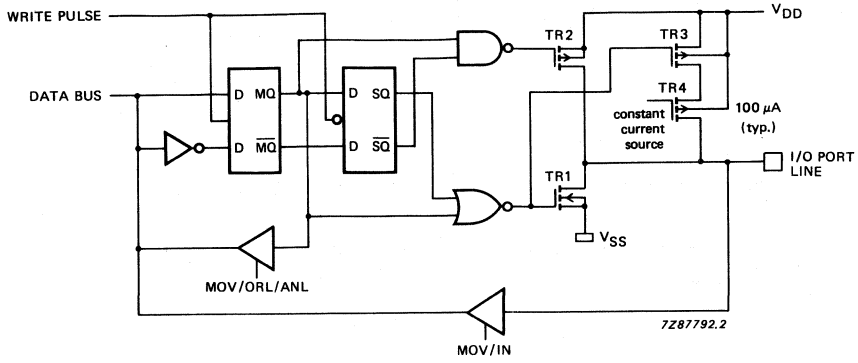


Fig. 9 (a) Standard output with switched pull-up current source.

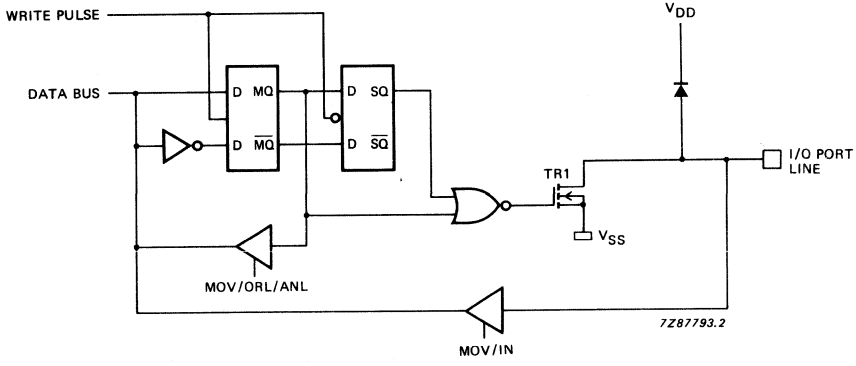


Fig. 9 (b) Open drain output.

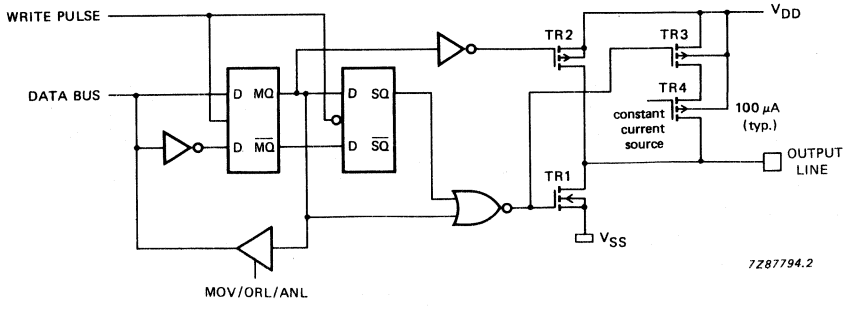


Fig. 9 (c) Push-pull output.

Timing reference on PCF84CXX ports is identical to that of the MAB84X1, see figure 10.

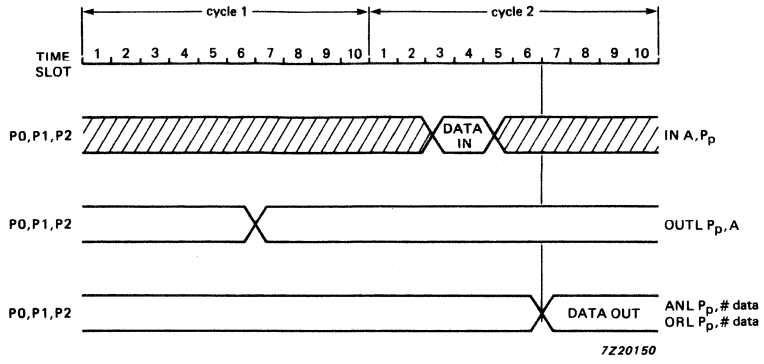


Fig. 10 Shows the timing diagram for all ports using IN, OUTL, ANL and ORL instructions. For the OUTL instruction data changes on time slot 7 of cycle 1. For the ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

4.4.2 Serial I/O

Identical to MAB84XX except:

- The timing specification of P23 and SCLK of the PCF84CXX is slightly different to accommodate the I²C-bus timing at an oscillator frequency of 10 MHz. This information is included in Tables 8 and 9.
- The input levels of P23 and SCLK are 0,3 V_{DD} (LOW) and 0,7 V_{DD} (HIGH).
- In NORMAL (running) and IDLE mode, the serial I/O logic remains active; its internal system clock will be switched off when there is no activity on the serial bus.
- After execution of the STOP instruction, the oscillator of the PCF84CXX is switched off. This means that the serial I/O logic will remain in the state it was at the occurence of the STOP instruction. To avoid "bus block" problems and to assure correct start-up of the bus after exit from the STOP mode, the user should disable the serial logic (ESO = 0) prior to the execution of the STOP instruction.

4.5 Interrupts

The interrupt system is single-level. Once an interrupt is detected, further interrupt requests are latched but ignored until the execution of a RETR instruction re-enables the interrupt logic. After executing RETR, the program continues in the main part. If a second interrupt occurs during the running of the first routine, the device will enter the second routine after having executed one instruction in the main program. The flow chart in figure 11 illustrates the interrupt priority polling sequence.

Upon entering an interrupt routine, the contents of the program counter and bits 4, 6 and 7 of the PSW are saved in the program counter stack. The contents of the accumulator must be saved by user software. Interrupt acknowledgement may be carried out with software via I/O pins. Interrupt routines must be terminated by a RETR instruction, if however, during an interrupt routine a CALL subroutine instruction is executed, this subroutine must terminate with a RET and not RETR. Ending a CALL routine with RETR would terminate the interrupt routine prematurely and could result in an incorrect return address.

4.5.1 External interrupts

When the external interrupt is enabled a high-to-low transition on the $\overline{\text{INT}}$/T0 input initiates an external interrupt subroutine which invokes a CALL to interrupt location 3. Depending upon instruction length, the external interrupt requires 2,6 to 3,6 instruction cycles until the program counter points to the interrupt vector 003 H. External interrupts are latched in the 'External interrupt flag' (EIF) even when they are not enabled. The DIS I instruction clears previously latched interrupts, the digital filter latch and the external interrupt flag.

4.5.2 SIO Interrupt

A serial I/O interrupt if enabled, invokes a CALL to location 5. SIO interrupts are latched within the SIO hardware, regardless of the SIO 'interrupt enable flag'. Latched SIO interrupts are cleared by setting the PIN bit in register S1 or by a dummy read/write of S0. In short, there is no automatic reset of the SIO interrupt flag after a CALL of the SIO interrupt routine.

4.5.3 Timer/counter interrupts

An overflow of the timer/event counter sets the 'timer interrupt flag' (TIF) and forces a CALL to location 7, if the timer interrupt is enabled. Timer/counter interrupts are only latched when the timer interrupt flag is enabled. The timer flag, which is set at each overflow, is not automatically reset after a call of the timer interrupt routine, it can only be cleared by a JTF/JNTF instruction or a hardware RESET.

4.5.4 Simultaneous interrupts

If 2 or 3 interrupts occur simultaneously, their priority is:

1) external
2) serial I/O
3) timer/counter

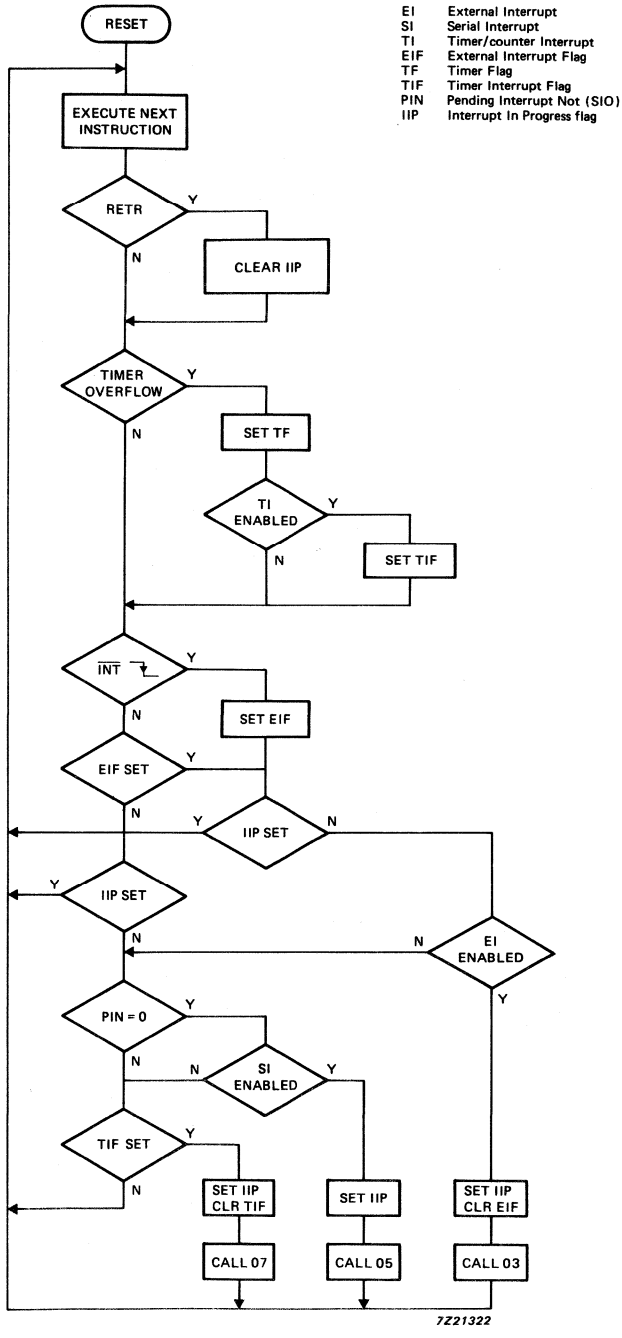


Fig. 11 The interrupt priority polling sequence.

Each interrupt routine is run until completion, regardless of any further latched interrupts. Any further interrupt will be latched with the exception of a timer/counter interrupt which must first enable its timer interrupt flag. After RETR at least one instruction of the main program is executed before a second call to an interrupt routine is made. Interrupt routines can only be aborted by a hardware RESET.

Another external interrupt can be created by enabling the timer/event counter interrupt and loading FFH into the counter (one less than overflow). This enables the event counter mode and a LOW-to-HIGH transition on the T1 input will then initiate an interrupt subroutine and cause a call to the timer/counter interrupt vector location 7.

There is a slight difference between the MAB84X1 and the PCF84CXX interrupt logic. A schematic of the PCF84CXX is shown in figure 12.

When executing a DIS I instruction, the PCF84CXX clears the enable external interrupt flag, the external interrupt flag and the digital latch, rejecting at any time a pending latched interrupt. However, the MAB84X1 using the same instruction clears only the enable external interrupt flag and the external interrupt flag thus cancelling enabled interrupts only.

For both the PCF84CXX and the MAB84X1, the external interrupt will always be latched in the digital filter/latch, even when the external interrupt is disabled.

There is no internal pull-up or pull-down device connected to the external interrupt input (pin 12). If required it must be implemented using a resistor (R = 100 kΩ). When the external interrupt is not used pin 12 must be connected to V_{DD}.

4.5.5 Interrupt logic

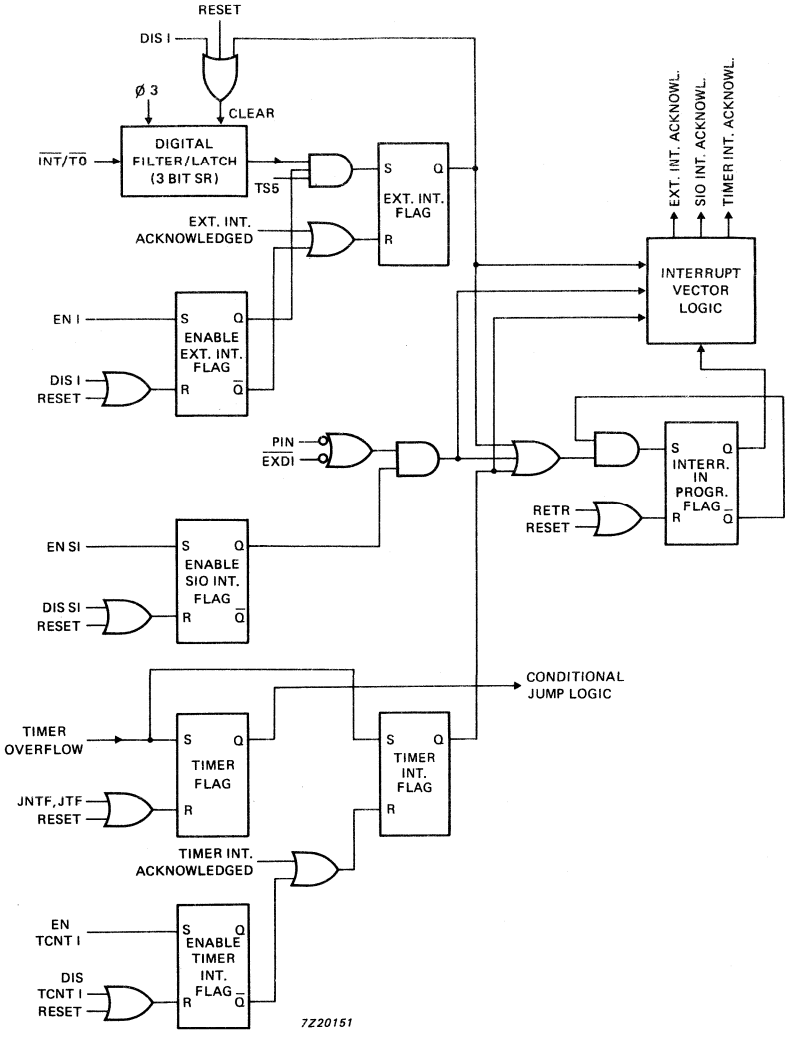


Fig. 12 Interrupt logic PCF84CXX.

NOTE:

1. $\overline{INT}$/TO negative edge is always latched in the digital filter/latch.
2. Correct interrupt timing is ensured when $\overline{INT}$/TO is HIGH for >4 CP followed by a LOW for >7 CP.
3. When the interrupt in progress flag is set, further interrupts are latched but ignored, until RETR is executed.
4. A DIS I instruction always clears a pending external interrupt.
5. For all flip-flops, reset always overules set

4.5.6 Interrupt program examples

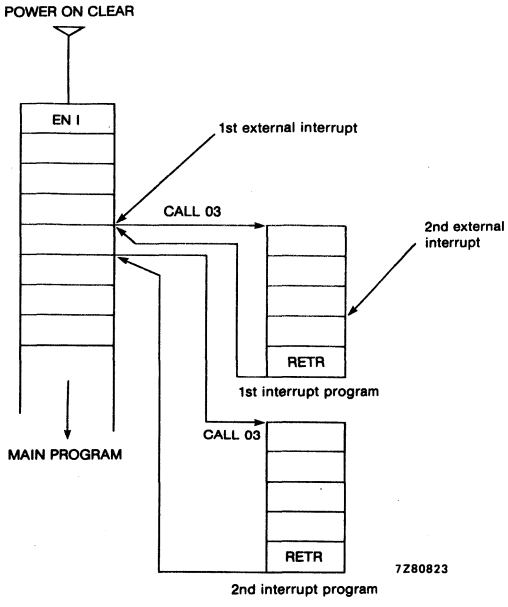


Fig. 13 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

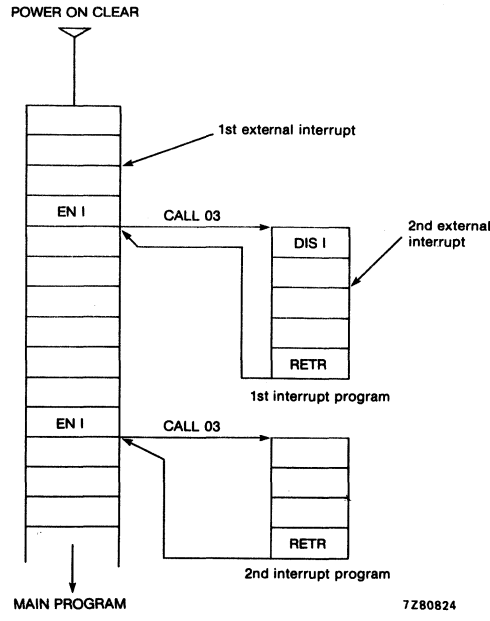


Fig. 14 Second external interrupt is postponed until enabled again.

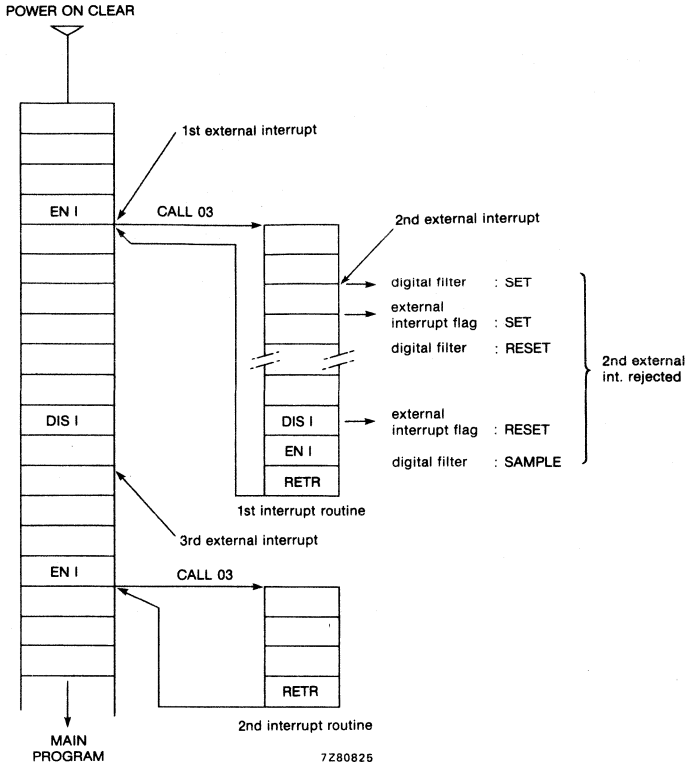
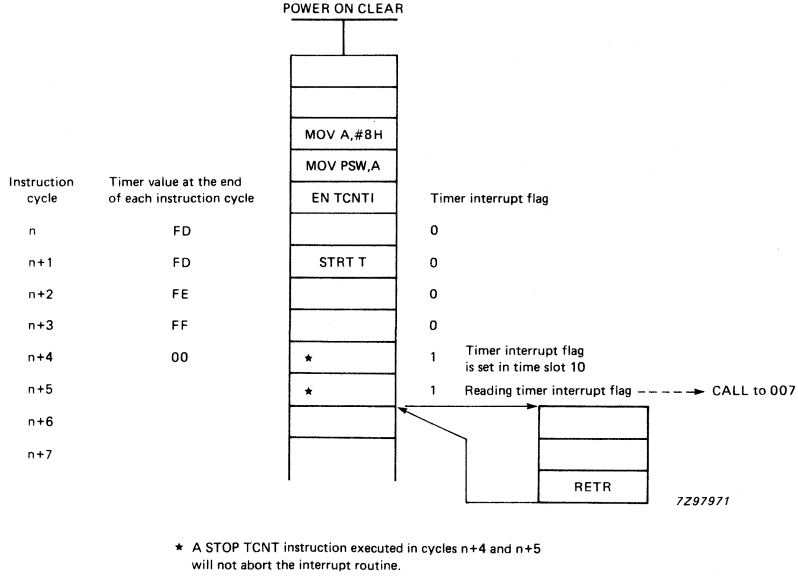


Fig. 15 Clearing of previous external interrupt.



* A STOP TCNT instruction executed in cycles n+4 and n+5 will not abort the interrupt routine.

Fig. 16 Execution of timer interrupt sequence.

4.5.6 Interrupt program examples (continued)

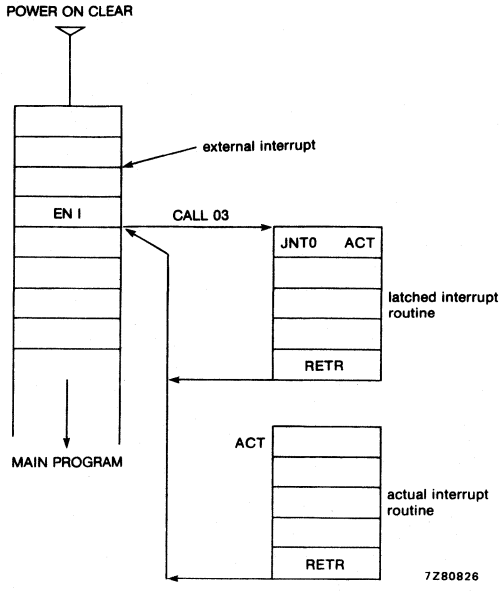


Fig. 17 Detection of previous and actual external interrupt.

4.5.7 Timer interrupt logic

A minor difference exists between the timer interrupt logic in the MAB84X1 and PCF84CXX;

On the PCF84CXX the timer flag TF remains set until a reset or JNTF/JTF is executed. With the MAB84X1 however, the timer flag is reset almost immediately (a few clock cycles) unless the timer interrupt flag is disabled.

4.5.8 Exiting the IDLE mode via an interrupt

The microcontroller will exit the IDLE mode by any one of the following three interrupts or a RESET, provided the concerned interrupt is enabled:-

1) External
2) Serial I/O
3) Timer/event counter

4.6 Test input T1, $\overline{\text{INT}}$/TO

4.6.1 Test input T1

The T1 input line can be used as:

- a test for branch instructions
- an external input to the event counter

When T1 is used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels respectively. A LOW-to-HIGH transition on the T1 input increments the timer/event counter.

There is no internal pull-up or pull-down resistor connected to the T1 input. If required it must be externally implemented using a resistor ($R \geqslant 100$ kΩ). When T1 is not used it must be connected to either V_{DD} or V_{SS}.

When used as an input to the event counter, T1 must be held LOW for at least 4 clock periods, followed by a HIGH for a further 4 clock periods. The maximum repetition rate is once per machine cycle.

4.6.2 Test input $\overline{INT}$/TO

The $\overline{INT}$/TO input may be used as:

- an external interrupt
- an input level that may be tested by the conditional jump instructions JTO and JNTO.

When used as an external interrupt input, the interrupt signal must be held HIGH for a minimum of 4 clock periods and LOW for a minimum of 7 clock periods. The interrupt is detected on its negative going edge. The interrupt facility is discussed in more detail in Chapter 4.5.

4.7 Oscillator and clock

Oscillator

The oscillator can be inhibited by the STOP instruction under software control. It is also inhibited when a low supply voltage condition is present to prevent discharge of a weak back-up battery.

Provided the supply voltage is within the operating range, the oscillator will be restarted after a STOP instruction by a LOW level at the $\overline{INT}$/TO pin or a HIGH level at the RESET pin.

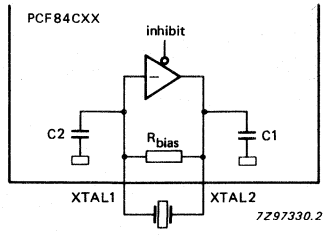


Fig. 18 Oscillator with integrated elements.

Due to the dynamic logic circuitry the minimum oscillator frequency must be 100 kHz. If a quartz crystal is used no additional external components are needed (see Fig. 18).

There are two other possibilities of clock generation (see Fig. 19):

a) Applying an external clock to pin 15, XTAL 1 (see Fig. 19)

Since the internal clock pulses are directly derived from input XTAL 1, duty cycle variations at this point must be small. In order to achieve this, the clock circuit is designed for a peak-to-peak voltage of a maximum of 0.9 V_{DD}. Any higher voltage will be limited by internal clamping diodes. These conditions must be fulfilled using an external generator.

b) Using a L-C oscillator. The resonant frequency is given by the following equation:

$$f = \frac{1}{2\sqrt{L \cdot \frac{C1T \cdot C2T}{C1T + C2T}}}$$

where:

$$C1T = C1 + C1_I + C1_W$$

$$C2T = C2 + C2_I + C2_W$$

$$C1_I \approx 4pF; \quad C2_I \approx 6pF$$

For C_1 and C_2 minimum capacitances of 20 pF are recommended.

C_I is the on-chip integrated capacitance.
C_W is the capacitance of the wiring.

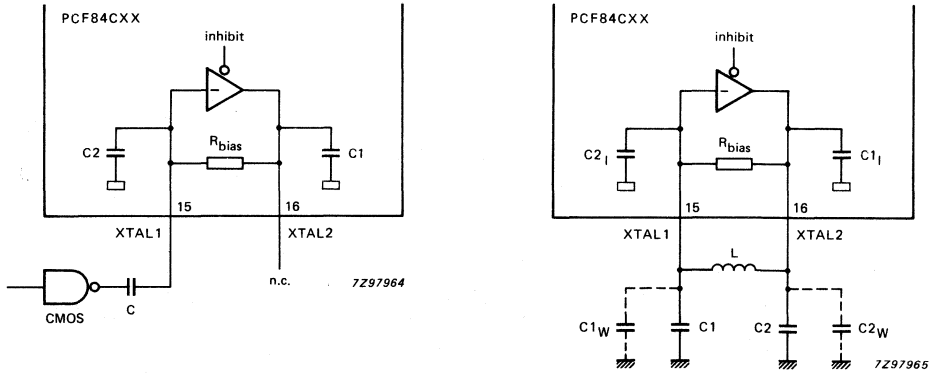


Fig. 19 Two possible clock circuit arrangements.

4.8 Timer/event counter

T1 as an input to the timer/event counter is described in section 4.6.

An internal 8-bit binary up-counter is provided. This can count external events, modulo-32 machine cycles, or machine cycles directly (Fig. 20). Table 1 shows the instructions that control the counter and prescaler and the functions performed.

Table 1 Timer/event counter control

function	timer mode modulo-1, module-32*	counter mode
CLEAR	MOV T,A (A) = 0 or RESET	MOV T,A (A) = 0 or RESET
PRESET	MOV T,A	MOV T,A
START	STRT T	STRT CNT
STOP	STOP TCNT or RESET	STOP TCNT or RESET
TEST	JTF/JNTF	JTF/JNTF
READ**	MOV A,T	MOV A,T

* With prescaler select, PS = 0, the timer counts modulo-32 machine cycles, with PS = 1 it counts modulo-1 cycles (prescaler not used); prescaler cleared with STRT T, prescaler not readable.

** READ does not disturb the counting process.

When used as a timer, the input to the counter is either the overflow or input from a 5-bit prescaler. When used as an event counter, LOW-to-HIGH transitions on T1 (pin 13) are counted. The counter will be incremented in the present instruction cycle if the falling edge occurs during the first 70% of the cycle length, failing this, it will be incremented in the following cycle. The maximum rate at which the counter may be incremented is once every machine cycle (333 kHz for a 3 µs machine cycle). When the counter overflows, the timer flag is set. The flag can be tested and reset using the JTF (jump if timer flag = 1) instruction and JNTF - instruction. Overflow generates an interrupt to the processor when the timer/event counter interrupt is enabled.

The counter will increment if the falling edge at T1 occurs before time slot 8 of any cycle.

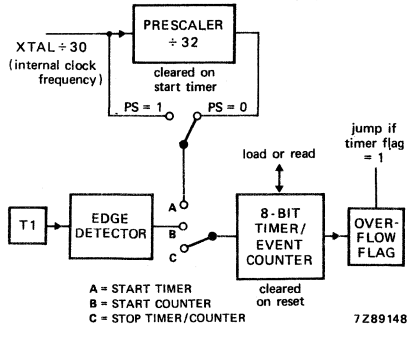


Fig. 20 Timer/event counter.

4.9 Program status word

The program status word (PSW) is an 8-bit word (1 byte) in the CPU which stores information about the current status of the microcontroller (Fig. 21). The PSW bits are:

bits 0, 1 and 2 - stack pointer bits (SP_0, SP_1, SP_2),
bit 3 - prescaler select (PS); 0 = modulo-32; 1 = modulo-1 (no prescaling),
bit 4 - working register bank select (RBS); 0 = register bank 0; 1 = register bank 1,
bit 5 - not used set to (1),
bit 6 - auxiliary carry (AC); half carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A,
bit 7 - carry (CY); the carry flag indicates that the previous operation has resulted in an overflow of the accumulator.

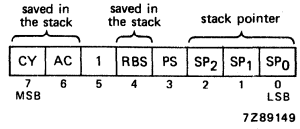


Fig. 21 Program status word.

All bits can be read using the MOV A, PSW instruction. Bits 7 and 6 are set and cleared by CPU operation. Bit 4 can be changed by a SEL RB instruction, bit 3 by the MOV PSW,A instruction, and bits 0, 1 and 2 by the CALL, RET or RETR instructions. Bits 7, 6 and 4 are stored in the program counter stack during subroutines and interrupt calls. These bits are restored in the PSW with a RETR (return and restore) instruction which must be used at the end of an interrupt and can be used at the end of a normal subroutine. The RET instruction has no restore feature and must not be used at the end of an interrupt.

4.10 Program counter

The 13-bit program counter can address up to to 8 K bytes of ROM, figure 22 shows the arrangement of the bits. During an interrupt subroutine PC_{11} and PC_{12} are forced to 0. All 13 bits are saved in the stack during CALL and interrupt routines.

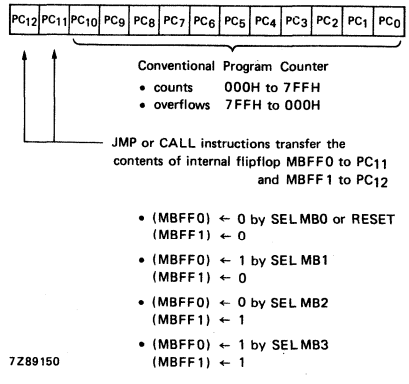


Fig. 22 Program counter.

4.11 Central processing unit

The PCF84CXX family has arithmetic, logic and branch capabilities. The DA A, SWAP A, and XCHD instructions simplify BCD arithmetic and bit handling. The MOVP A,@A instruction permits table look up from the current ROM page.

The conditional branch logic within the processor enables several conditions, to be tested by the user's program, internal and external to the processor. Table 2 lists the conditional jump instructions used to change program sequence. The DJNZ instruction decrements a designated register or data memory location and branches if the contents are non-zero. This instruction is useful for looping control. The JMPP@A instruction allows multiway branches to destinations the address of which are pointered by the accumulator.

Table 2 Conditional branches

test	jump condition	jump instruction
accumulator	all bits zero	JZ
	any bit non-zero	JNZ
accumulator bit test	1	JB0 to JB7
carry flag	1	JC
	0	JNC
timer overflow flag	1	JTF
	0	JNTF
test input T0	1	JT0
	0	JNT0
test input T1	1	JT1
	0	JNT1
register	non-zero	DJNZ

4.12 Reset

- A positive-going signal on the RESET input/output:

- Sets the program counter to zero
- Selects location 0 of memory bank 0 and register bank 0
- Sets the stack pointer to zero (000); pointing to RAM address 8
- Disables the interrupts (external, timer and serial I/O)
- Stops the timer/event counter, then sets it to zero
- Sets the timer prescaler to modulo-32
- Resets the timer flag
- Sets all ports to input mode
- Sets the serial I/O to slave receiver mode and disables the the serial I/O
- Cancels IDLE and STOP mode

4.12.1 Power-on-reset

The internal power-on reset circuit monitors the PCF84CXX supply voltage V_{DD}. For as long as the supply voltage remains below the internal reference level V_{ref} (typically 1,5 V) the oscillator is inhibited and RESET has an undefined level. When V_{DD} rises above the internal reference level, the oscillator is released and RESET is pulled high to V_{DD} by TR1 for a period t_D (typically 50 μs).

N.B. Because of the narrow bandwidth of the crystal, the start-up time of the oscillator is typically 10 ms.

Three cases of power-on reset are possible and are defined below (see figures 23, 25 and 27):

1. If V_{DD} can be switched on fast enough i.e. V_{DD} reaches its minimum operating value (corresponding to the selected oscillator frequency) before the RESET signal (t_D) has finished, then no extra components are required (see Fig. 23 and 24). Note that the first instruction is executed after the oscillator start-up time plus 1866 clock periods have elapsed.

2. If V_{DD} has a slow rise time then the RESET signal should be stretched by an external RC circuit (see Fig. 25 and 26). In the case of a short drop in the supply voltage, the diode path rapidly discharges the capacitor to ensure a reliable power-on reset. To ensure a correct reset, the RESET signal should reach at least 70% of the final value of V_{DD}. Given that the RESET voltage and V_{DD} rise exponentially, the above requirement is satisfied when the time constant T of the RESET pulse is >8 times the time constant of V_{DD}. If V_{DD} rises linearly, then a RESET time constant >2 times the rise time of V_{DD} is required.

   When a reset is completed (RESET goes LOW) before the oscillator has started up, program execution begins after the oscillator start-up time plus 1866 clock periods have elapsed (see Fig. 26). If the oscillator is started-up prior to the completion of RESET, then program execution begins 1866 clock periods after RESET goes LOW.

3. Fig. 27 shows an external reset to the PCF84CXX during power-on. The external reset signal must remain HIGH until V_{DD} has reached its minimum operating value corresponding to the selected oscillator frequency. When a reset is completed (RESET goes LOW) before the oscillator has started up, program execution begins after the oscillator start-up time plus 1866 clock periods have elapsed (see Fig. 28). If the oscillator is started-up prior to the completion of RESET, then program execution begins 1866 clock periods after RESET goes LOW.

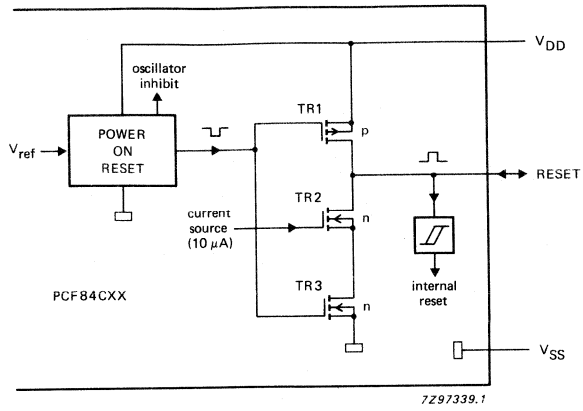


Fig. 23 Power-on-reset configuration.

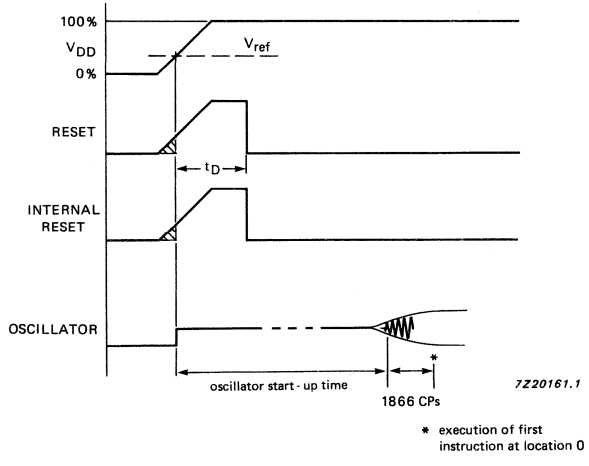


Fig. 24 Timing of power-on-reset with fast rise time.

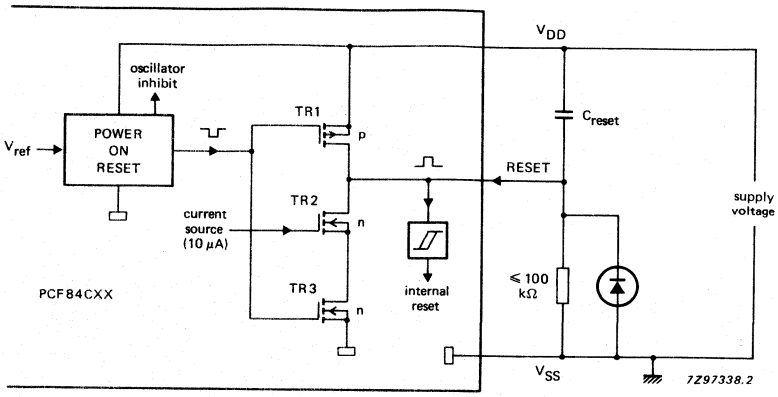


Fig. 25 Stretched power-on-reset with external components.

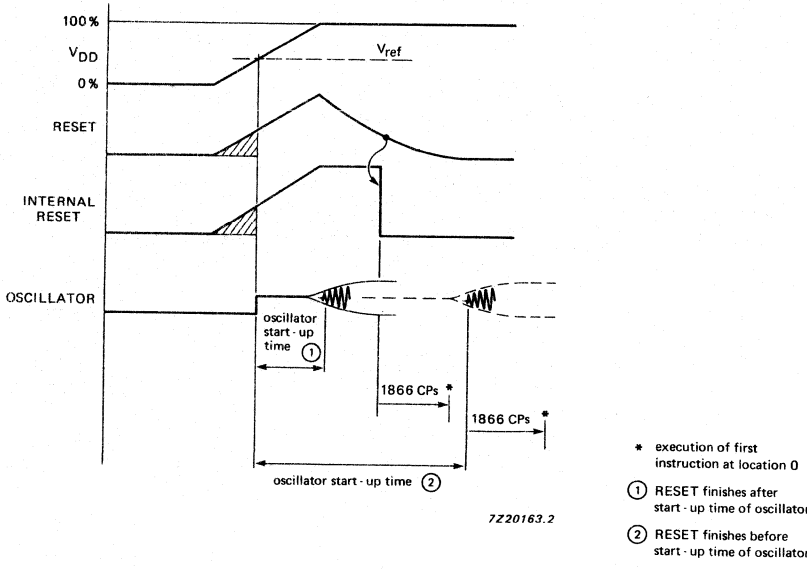


Fig. 26 Timing of power-on-reset with a slowly rising V_{DD} and a stretched RESET pulse.

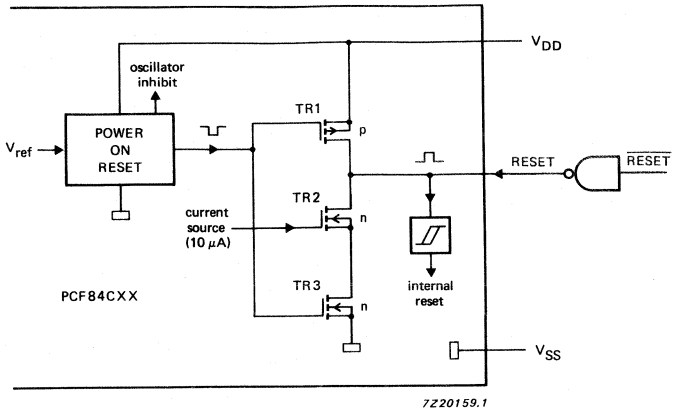


Fig. 27 External power-on-reset configuration.

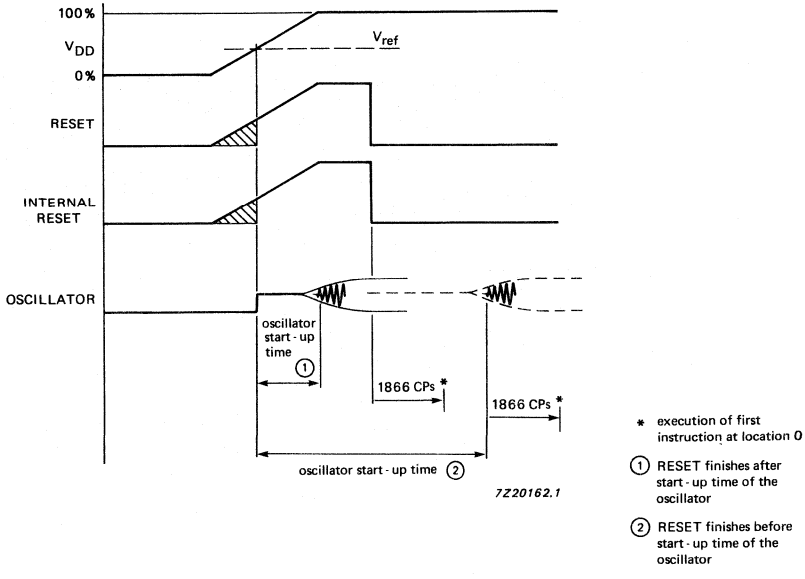


Fig. 28 Timing of external power-on-reset.

4.13 Connection of the PCF84CXX in a typical system

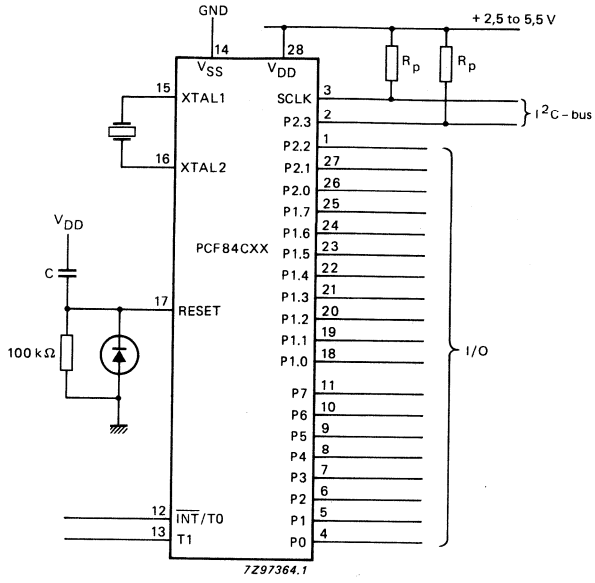


Fig. 29 Connection of PCF84CXX in a typical system.

Note to figure 29: All unused inputs without internal pull-up or pull-down should be connected to V_{SS} or V_{DD}.

The value of pull-up resistors R_p can be found in the I^2C section of this manual.

4.14 The Instruction set

The instruction set of the PCF84CXX is identical to that of the MAB84X1 with the addition of two instructions for the power saving modes;

IDLE: opcode 01 H (1 byte/1 cycle)
STOP: opcode 22 H (1 byte/1 cycle)

The PCF84C00T has an additional 4 instructions to handle the derivative registers, these are:

MOV A,Dx	Move derivative register contents to accumulator	(A) ← (Dx)
MOV Dx,A	Move accumulator contents to derivative register	(Dx) ← (A)
ANL Dx,A	AND derivative register with accumulator	(Dx) ← (Dx) AND (A)
ORL Dx,A	OR derivative register with accumulator	(Dx) ← (Dx) OR (A)

Table 3 shows the instruction map of the PCF84CXX

Table 3 Instruction map

first hexadecimal character of opcode ↓	second hexadecimal character of opcode															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	IDLE		ADD A,#data	JMP page 0	EN I	JNTF addr	DEC A	IN A,Pp 0 1 2				MOV A,Sn 0 1			
1	INC @Rr 0 1		JBO addr	ADDC A,#data	CALL page 0	DIS I	JTF addr	INC A	INC Rr 0 1 2 3 4 5 6 7							
2	XCH A,@Rr 0 1		STOP	MOV A,#data	JMP page 1	EN TCNTI	JNTO addr	CLR A	XCH A,Rr 0 1 2 3 4 5 6 7							
3	XCHD A,@Rr 0 1		JB1 addr		CALL page 1	DIS TCNTI	JTO addr	CPL A	OUTL Pp,A 0 1 2				MOV Sn,A 0 1 2			
4	ORL A,@Rr 0 1		MOV A,T	ORL A,#data	JMP page 2	STRT CNT	JNT1 addr	SWAP A	ORL A,Rr 0 1 2 3 4 5 6 7							
5	ANL A,@Rr 0 1		JB2 addr	ANL A,#data	CALL page 2	STRT T	JT1 addr	DA A	ANL A,Rr 0 1 2 3 4 5 6 7							
6	ADD A,@Rr 0 1		MOV T,A		JMP page 3	STOP TCNT		RRC A	ADD A,Rr 0 1 2 3 4 5 6 7							
7	ADDC A,@Rr 0 1		JB3 addr		CALL page 3			RR A	ADDC A,Rr 0 1 2 3 4 5 6 7							
8				RET	JMP page 4	EN SI			ORL Pp,#data 0 1 2							
9			JB4 addr	RETR	CALL page 4	DIS SI	JNZ addr	CLR C	ANL Pp,#data 0 1 2				MOV Sn,#data 0 1 2			
A	MOV @Rr,A 0 1			MOVP A,@A	JMP page 5	SEL MB2		CPL C	MOV Rr,A 0 1 2 3 4 5 6 7							
B	MOV @Rr,#data 0 1		JB5 addr	JMPP @A	CALL page 5	SEL MB3			MOV Rr,#data 0 1 2 3 4 5 6 7							
C	DEC @Rr 0 1				JMP page 6	SEL RB0	JZ addr	MOV A,PSW	DEC Rr 0 1 2 3 4 5 6 7							
D	XRL A,@Rr 0 1		JB6 addr	XRL A,#data	CALL page 6	SEL RB1		MOV PSW,A	XRL A,Rr 0 1 2 3 4 5 6 7							
E	DJNZ @Rr,addr 0 1				JMP page 7	SEL MB0	JNC addr	RL A	DJNZ Rr, addr 0 1 2 3 4 5 6 7							
F	MOV A,@Rr 0 1		JB7 addr		CALL page 7	SEL MB1	JC addr	RLC A	MOV A,Rr 0 1 2 3 4 5 6 7							

Table 4 Instruction set

	mnemonic	opcode (hex.)	bytes/ cycles	description	function		notes
ACCUMULATOR	ADD A, Rr	6*	1/1	Add register contents to A	(A)←(A) + (Rr)	r = 0–7	1
	ADD A, @Rr	60 61	1/1	Add RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) (A)←(A) + ((R1))		1
	ADD A, #data	03 data	2/2	Add immediate data to A	(A)←(A) + data		1
	ADDC A, Rr	7*	1/1	Add carry and register contents to A	(A)←(A) + (Rr) + (C)	r = 0–7	1
	ADDC A, @Rr	70 71	1/1	Add carry and RAM data, addressed by Rr, to A	(A)←(A) + ((R0)) + (C) (A)←(A) + ((R1)) + (C)		1
	ADDC A, #data	13 data	2/2	Add carry and immediate data to A	(A)←(A) + data + (C)		1
	ANL A, Rr	5*	1/1	'AND' Rr with A	(A)←(A) AND (Rr)	r = 0–7	
	ANL A, @Rr	50 51	1/1	'AND' RAM data, addressed by Rr, with A	(A)←(A) AND ((R0)) (A)←(A) AND ((R1))		
	ANL A, #data	53 data	2/2	'AND' immediate data with A	(A)←(A) AND data		
	ORL A, Rr	4*	1/1	'OR' Rr with A	(A)←(A) OR (Rr)	r = 0–7	
	ORL A, @Rr	40 41	1/1	'OR' RAM data, addressed by Rr, with A	(A)←(A) OR ((R0)) (A)←(A) OR ((R1))		
	ORL A, #data	43 data	2/2	'OR' immediate data with A	(A)←(A) OR data		
	XRL A, Rr	D*	1/1	'XOR' Rr with A	(A)←(A) XOR (Rr)	r = 0–7	
	XRL A, @Rr	D0 D1	1/1	'XOR' RAM, addressed by Rr, with A	(A)←(A) XOR ((R0)) (A)←(A) XOR ((R1))		
	XRL A, #data	D3 data	2/2	'XOR' immediate data with A	(A)←(A) XOR data		
	INC A	17	1/1	increment A by 1	(A)←(A) + 1		
	DEC A	07	1/1	decrement A by 1	(A)←(A) −1		
	CLR A	27	1/1	clear A to zero	(A)←0		
	CPL A	37	1/1	one's complement A	(A)←NOT(A)		
	RL A	E7	1/1	rotate A left	$(A_{n+1})\leftarrow(A_n)$ $(A_0)\leftarrow(A_7)$	n = 0–6	

Table 4 Instruction set

ACCUMULATOR (cont.)	RLC A	F7	1/1	rotate A left through carry	$(A_{n+1})\leftarrow A_n$ $(A_0)\leftarrow(C)$, $(C)\leftarrow(A_7)$	n = 0–6	2
	RR A	77	1/1	rotate A right	$(A_n)\leftarrow(A_{n+1})$ $(A_7)\leftarrow(A_0)$	n = 0–6	
	RRC A	67	1/1	rotate A right through carry	$(A_n)\leftarrow(A_{n+1})$ $(A_7)\leftarrow(C)$, $(C)\leftarrow(A_0)$	n = 0–6	2
	DA A	57	1/1	decimal adjust A			2
	SWAP A	47	1/1	swap nibbles of A	$(A_{4-7})\leftrightarrow(A_{0-3})$		
DATA MOVES	MOV A, Rr	F*	1/1	move register contents to A	(A)←(Rr)	r = 0–7	
	MOV A, @Rr	F0 F1	1/1	move RAM data, addressed by Rr, to A	(A)←((R0)) (A)←((R1))		
	MOV A, #data	23 data	2/2	move immediate data to A	(A)←data		
	MOV Rr, A	A*	1/1	move accumulator contents to register	(Rr)←(A)	r = 0–7	
	MOV @Rr, A	A0 A1	1/1	move accumulator contents to RAM location addressed by Rr	((R0))←(A) ((R1))←(A)		
	MOV Rr, #data	B* data	2/2	move immediate data to Rr	(Rr)←data		
	MOV @Rr, #data	B0 data B1 data	2/2	move immediate data to RAM location addressed by Rr	((R0))←data ((R1))←data		
	XCH A, Rr	2*	1/1	exchange accumulator contents with Rr	(A)↔(Rr)	r = 0–7	
	XCH A, @Rr	20 21	1/1	exchange accumulator contents with RAM data addressed by Rr	(A)↔((R0)) (A)↔((R1))		
	XCHD A, @Rr	30 31	1/1	exchange lower nibbles of A and RAM data addressed by Rr	$(A_{0-3})\leftrightarrow((R0_{0-3}))$ $(A_{0-3})\leftrightarrow((R1_{0-3}))$		
	MOV A, PSW	C7	1/1	move PSW contents to accumulator	(A)←(PSW)		
	MOV PSW, A	D7	1/1	move accumulator bit 3 to PSW_3	$(PSW_3)\leftarrow(A_3)$		3
	MOVP A, @A	A3	1/2	move indirectly addressed data in current page to A	$(PC_{0-7})\leftarrow(A)$, $(A)\leftarrow((PC))$		
FLAGS	CLR C	97	1/1	clear carry bit	(C)←0		2
	CPL C	A7	1/1	complement carry bit	(C)←NOT(C)		2

	mnemonic	opcode (hex.)	bytes/ cycles	description	function		notes
REGISTER	INC Rr	1*	1/1	increment register by 1	(Rr)←(Rr) + 1	r = 0–7	
	INC @Rr	10 11	1/1	increment RAM data, addressed by Rr, by 1	((R0))←((R0)) + 1 ((R1))←((R1)) + 1		
	DEC Rr	C*	1/1	decrement register by 1	(Rr)←(Rr) – 1	r = 0–7	
	DEC @Rr	C0 C1	1/1	decrement RAM data, addressed by Rr, by 1	((R0))←((R0)) – 1 ((R1))←((R1)) – 1		
BRANCH	JMP addr	● 4 address	2/2	unconditional jump within a 2 K bank	(PC_{8-10})←$addr_{8-10}$ (PC_{0-7})←$addr_{0-7}$ (PC_{11-12})←MBFF 0–1		
	JMPP @A	B3	1/2	indirect jump within a page	(PC_{0-7})←((A))		
	DJNZ Rr, addr	E* address	2/2	decrement Rr by 1 and jump if not zero to addr	(Rr)←(Rr) – 1 if (Rr) not zero (PC_{0-7})←addr	r = 0–7	
	DJNZ @Rr, addr	E0	2/2	decrement RAM data, addressed by Rr by 1 and jump if not zero to addr	((R0))←((R0)) – 1 if ((R0)) not zero (PC_{0-7})←addr		
		E1			((R1))←((R1)) – 1 if ((R1)) not zero (PC_{0-7})←addr		
	JBb addr	▲ 2 address	2/2	jump to addr if Acc. bit b = 1	if b = 1 : (PC_{0-7})←addr	b = 0–7	
	JC addr	F6 address	2/2	jump to addr if C = 1	if C = 1 : (PC_{0-7})←addr		
	JNC addr	E6 address	2/2	jump to addr if C = 0	if C = 0 : (PC_{0-7})←addr		
	JZ addr	C6 address	2/2	jump to addr if A = 0	if A = 0 : (PC_{0-7})←addr		
	JNZ addr	96 address	2/2	jump to addr if A is NOT zero	if A≠0 : (PC_{0-7})←addr		
	JT0 addr	36 address	2/2	jump to addr if T0 = 1	if T0 = 1: (PC_{0-7})←addr		
	JNT0 addr	26 address	2/2	jump to addr if T0 = 0	if T0 = 0: (PC_{0-7})←addr		
	JT1 addr	56 address	2/2	jump to addr if T1 = 1	if T1 = 1: (PC_{0-7})←addr		
	JNT1 addr	46 address	2/2	jump to addr if T1 = 0	if T1 = 0: (PC_{0-7})←addr		
	JTF addr	16 address	2/2	jump to addr if Timer Flag = 1	if TF = 1: (PC_{0-7})←addr		4
	JNTF addr	06 address	2/2	jump to addr if Timer Flag = 0	if TF = 0: (PC_{0-7})←addr		

Instruction set (continued)

TIMER/EVENT COUNTER	MOV A, T	42	1/1	move timer/event counter contents to accumulator	(A)←(T)	
	MOV T, A	62	1/1	move accumulator contents to timer/event counter	(T)←(A)	
	STRT CNT	45	1/1	start event counter		
	STRT T	55	1/1	start timer		
	STOP TCNT	65	1/1	stop timer/event counter		
	EN TCNTI	25	1/1	enable timer/event counter interrupt		
	DIS TCNTI	35	1/1	disable timer/event counter interrupt		
CONTROL	EN I	05	1/1	enable external interrupt		
	DIS I	15	1/1	disable external interrupt		
	SEL RB0	C5	1/1	select register bank 0	(RBS)←0	5
	SEL RB1	D5	1/1	select register bank 1	(RBS)←1	5
	SEL MB0	E5	1/1	select program memory bank 0	(MBFF0)←0, (MBFF1)←0	
	SEL MB1	F5	1/1	select program memory bank 1	(MBFF0)←1, (MBFF1)←0	
	SEL MB2	A5	1/1	select program memory bank 2	(MBFF0)←0, (MBFF1)←1	
	SEL MB3	B5	1/1	select program memory bank 3	(MBFF0)←1, (MBFF1)←1	
	STOP	22	1/1	enter STOP mode		
	IDLE	01	1/1	enter IDLE mode		
SUBROUTINE	CALL addr	▲ 4 address	2/2	jump to subroutine	$((SP))\leftarrow(PC), (PSW_{4,6,7})$ 6 $(SP)\leftarrow(SP)+1$ $(PC_{8-10})\leftarrow addr_{8-10}$ $(PC_{0-7})\leftarrow addr_{0-7}$ $(PC_{11-12})\leftarrow$MBFF 0–1	6
	RET	83	1/2	return from subroutine	$(SP)\leftarrow(SP)-1$ $(PC)\leftarrow((SP))$	6
	RETR	93	1/2	return from interrupt and restore bits 4, 6, 7 of PSW	$(SP)\leftarrow(SP)-1$ $(PSW_{4,6,7}) + (PC)\leftarrow((SP))$	6

	mnemonic	opcode (hex.)	bytes/ cycles	description	function	notes
PARALLEL INPUT/OUTPUT	IN A, Pp	08 09 0A	1/2	input port p data to accumulator	(A)←(P0) (A)←(P1) (A)←(P2)	 7
	OUTL Pp, A	38 39 3A	1/2	output accumulator data to port p	(P0)←(A) (P1)←(A) (P2)←(A)	
	ANL Pp, #data	98 data 99 data 9A data	2/2	AND port p data with immediate data	(P0)←(P0) AND data (P1)←(P1) AND data (P2)←(P2) AND data	
	ORL Pp, #data	88 data 89 data 8A data	2/2	OR port p data with immediate data	(P0)←(P0) OR data (P1)←(P1) OR data (P2)←(P2) OR data	
SERIAL INPUT/OUTPUT	MOV A, S_n	0C 0D	1/2	move serial I/O register contents to accumulator	(A)←(S0) (A)←(S1)	 8
	MOV S_n, A	3C 3D 3E	1/2	move accumulator contents to serial I/O register	(S0)←(A) (S1)←(A) (S2)←(A)	 9
	MOV S_n, #data	9C data 9D data 9E data	2/2	move immediate data to serial I/O register	(S0)←data (S1)←data (S2)←data	
	EN SI	85	1/1	enable serial I/O interrupt		
	DIS SI	95	1/1	disable serial I/O interrupt		
	NOP	00	1/1	no operation		

Notes to Table 8

1. PSW CY, AC affected
2. PSW CY affected
3. PSW PS affected
4. Execution of JTF and JNTF instructions resets the Timer Flag (TF).

* : 8, 9, A, B, C, D, E, F
● : 0, 2, 4, 6, 8, A, C, E
▲ : 1, 3, 5, 7, 9, B, D, F

5. PSW RBS affected
6. PSW SP_0, SP_1, SP_2 affected
7. (A) = 1111 P23, P22, P21, P20.
8. (S1) has a different meaning for read and write operation, see serial I/O interface.
9. (S2) is a write only register. Reading S2 will give value FFH.

5.0 IDLE AND STOP MODES

5.1 IDLE mode

When the microcontroller enters the IDLE mode via the IDLE instruction (01 H) the oscillator, timer/counter and serial I/O remain active. The microcontroller exits from IDLE mode by one of three interrupts, if enabled, or by executing a RESET. If the interrupt is not enabled the processor will remain in the IDLE mode. An active signal on the RESET pin restarts the microcontroller and a normal RESET sequence is executed (see Fig. 30).

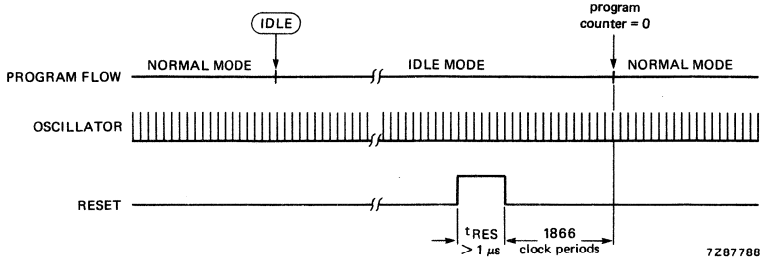


Fig. 30 Exit from IDLE mode via a RESET.

An active signal from an enabled interrupt will invoke execution of the normal interrupt routine since normal interrupt scanning is maintained. A HIGH-to-LOW transition on the external interrupt pin ($\overline{INT}$/T0) reactivates the microcontroller. A LOW level applied to $\overline{INT}$/T0 will reactivate the microcontroller only in the STOP mode. Thus, if $\overline{INT}$/T0 was LOW before the microcontroller entered the IDLE mode, it must return HIGH before the microcontroller can be reactivated (see Fig. 31).

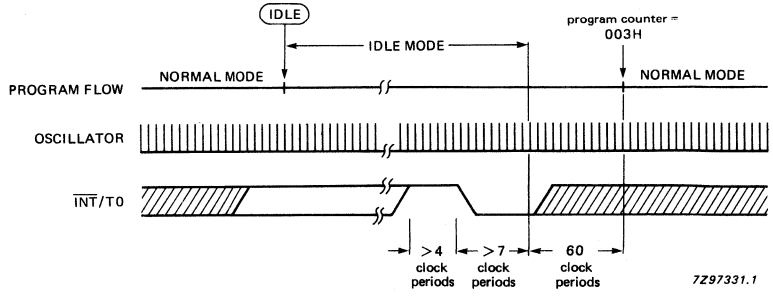


Fig. 31 Exit from IDLE mode via an interrupt.

Exit from the IDLE mode is ensured when INT/T0 is HIGH for > 4 CP (clock periods) followed by a LOW for > 7 CP. After the initial forced CALL; (003 H) operation (60 CP) the program continues with the external interrupt service routine. During IDLE mode, the program memory address bus carries the address of the next instruction following the IDLE instruction.

5.2 STOP mode

The microcontroller enters the STOP mode by the STOP instruction (22H) and the oscillator is switched off. The internal status of the CPU, RAM contents and the state of I/O ports are not affected. The microcontroller can be brought out of the STOP mode by an active signal at the external interrupt input or by an external RESET signal. When one of these two signals is applied, an internal delay of 1866 CP is provided to ensure that all internal clocks are operating correctly before restart (see Fig. 32).

Note: The start-up time of a crystal oscillator is measured in milliseconds, and the 1866 CP count begins after this start-up time.

If the microcontroller exits from the STOP mode by activating RESET, a normal RESET sequence is executed.

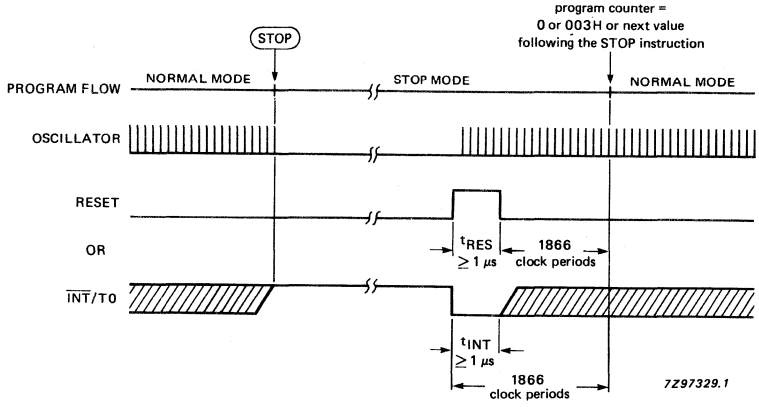


Fig. 32 Entering and exiting the STOP mode.

If the microcontroller exits the STOP mode by pulling the external interrupt input pin LOW, an interrupt sequence is executed only if the external interrupt is enabled. In this event the microcontroller resumes the normal program sequence after returning from the interrupt routine, as in the normal mode. If the interrupt is not enabled, it continues the normal program sequence, executing the instruction following the STOP instruction.

The microcontroller is restarted by a LOW level applied at the $\overline{INT}$/TO pin, and not by a HIGH-to-LOW transition as in a normal interrupt mechanism.

When the $\overline{INT}$/TO level is active during the STOP instruction then no STOP is executed.

A LOW level on the external interrupt input of at least 1 µs will cause the microcontroller to exit the STOP mode. During STOP mode, the program memory address lines are pointing to the address of the instruction immediately following the STOP instruction.

6.0 TIMING

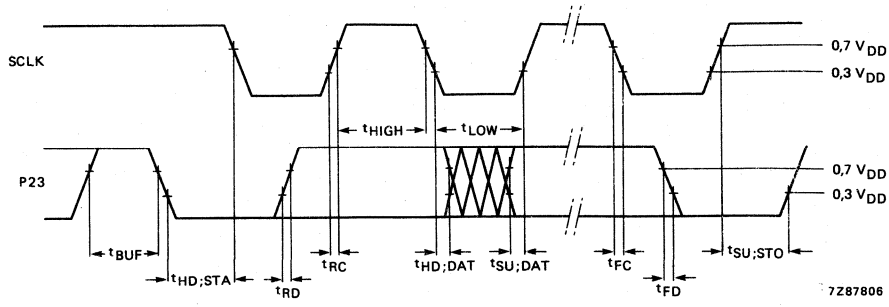


Fig. 33 PCF84CXX timing requirements for the P23 and SCLK input signals.

Table 5 Input timing shown in figure 33

symbol	timing
t_{BUF}	$\geqslant 14t_{XTAL}$
$t_{HD;STA}$	$\geqslant 14t_{XTAL}$
t_{HIGH}	$\geqslant 17t_{XTAL}$
t_{LOW}	$\geqslant 17t_{XTAL}$
$t_{SU;STO}$	$\geqslant 14t_{XTAL}$
$t_{HD;DAT}$	> 0
$t_{SU;DAT}$	$\geqslant 250$ ns
t_{RD}	$\leqslant 1\ \mu s$
t_{RC}	$\leqslant 1\ \mu s$
t_{FD}	$\leqslant 1\ \mu s$
t_{FC}	$\leqslant 0{,}3\ \mu s$

Notes to Table 5

t_{XTAL} = one period of the XTAL input frequency (f_{XTAL})

= 100 ns for f_{XTAL} = 10 MHz.

These figures apply to all modes.

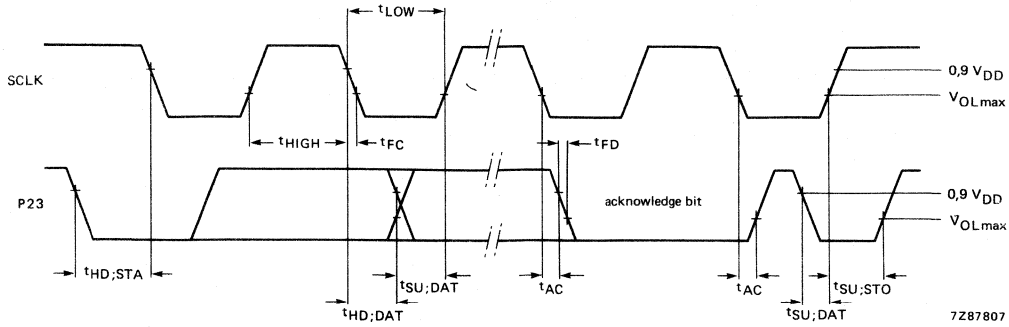


Fig. 34 PCF84CXX timing for the P23 and SCLK output signals.

Table 6 Output timing shown in figure 34.

symbol	timing	
	normal mode (ASC in S2 = 0)	low-speed mode (ASC in S2 = 1)
$t_{HD;STA}$	½ (DF + 9) t_{XTAL}	¾ (DF + 9) t_{XTAL}
t_{HIGH}	½ (DF) t_{XTAL}	¾ (DF) t_{XTAL}
t_{LOW}	½ (DF) t_{XTAL}	¼ (DF) t_{XTAL}
$t_{SU;STO}$	½ (DF − 3) t_{XTAL}	¼ (DF − 3) t_{XTAL}
$t_{HD;DAT}$ (slave transmitter) any DF	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$
$t_{HD;DAT}$ (master transmitter) for DF $\leqslant$ 51	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$	– –
for DF $\leqslant$ 99	– –	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$
$t_{SU;DAT}$ (master transmitter) for DF > 51	$\geqslant 15t_{XTAL}$ $\leqslant 24t_{XTAL}$	– –
for DF > 99	– –	$\geqslant 15t_{XTAL}$ $\leqslant 24t_{XTAL}$
t_{AC}	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$	$\geqslant 9t_{XTAL}$ $\leqslant 12t_{XTAL}$
t_{FD}, t_{FC}	$\leqslant$ 100 ns at C_b = 400 pF	$\leqslant$ 100 ns at C_b = 400 pF

Notes to Table 6

t_{XTAL} = one period of the XTAL input frquency (f_{XTAL})
= 167 ns for f_{XTAL} = 6 MHz.
DF = divisor
C_b = the maximum bus capacitance for each line.

6.1 The PCF84C12 microcontroller

CONTENTS – PCF84C12 8-BIT MICROCONTROLLER

		page
1.0	DESCRIPTION	6.1-5
2.0	FEATURES	6.1-5
3.0	PACKAGE OUTLINES	6.1-5
3.1	Pin designation	6.1-7
4.0	FUNCTIONAL DESCRIPTION	6.1-8
4.1	Program memory	6.1-8
4.2	Data memory	6.1-8
4.3	Working registers	6.1-8
4.4	Interrupts	6.1-10

1.0 DESCRIPTION

The PCF84C12 microcontroller is manufactured in CMOS technology.
It has 13 quasi-bidirectional I/O port lines, one single-level vectored interrupt, an 8-bit timer/event counter and on-board clock oscillator and clock circuits.

This microcontroller is an efficient controller as well as an arithmetic processor. The instruction set is based on that of the PCF84CXX and is pin and software compatible with both the PCF84CXX and MAB84X1 families. The PCF84C12 has bit handling abilities and facilities for both binary and BCD arithmetic.

As the PCF84C12 is based on the PCF84CXX family only outstanding differences are described herein. For more detailed information on the standard functions of PCF84C12, see the previous PCF84CXX section.

2.0 FEATURES

- 8-bit CPU, ROM, RAM, I/O in a single 20-lead DIL or SO package
- 1 K ROM bytes
- 64 RAM bytes
- 13 quasi-bidirectional I/O port lines
- Two test inputs: one of which is also the external interrupt input
- Single-level vectored interrupts: external and timer/event counter
- 8-bit programmable timer/event counter
- Clock frequency 100 kHz to 10 MHz
- Over 80 instructions (based on MAB8048) all of 1 or 2 cycles
- Single supply voltage from 2,5 V to 5,5 V
- STOP and IDLE mode
- Power-on-reset circuit and low supply voltage detection
- Operating temperature range: -40 to + 85 ^{o}C

3.0 PACKAGE OUTLINES

PCF84C12P: 20-lead DIL; plastic (SOT-146).
PCF84C12T: 20-lead mini-pack; plastic (SO-20; SOT-163A).

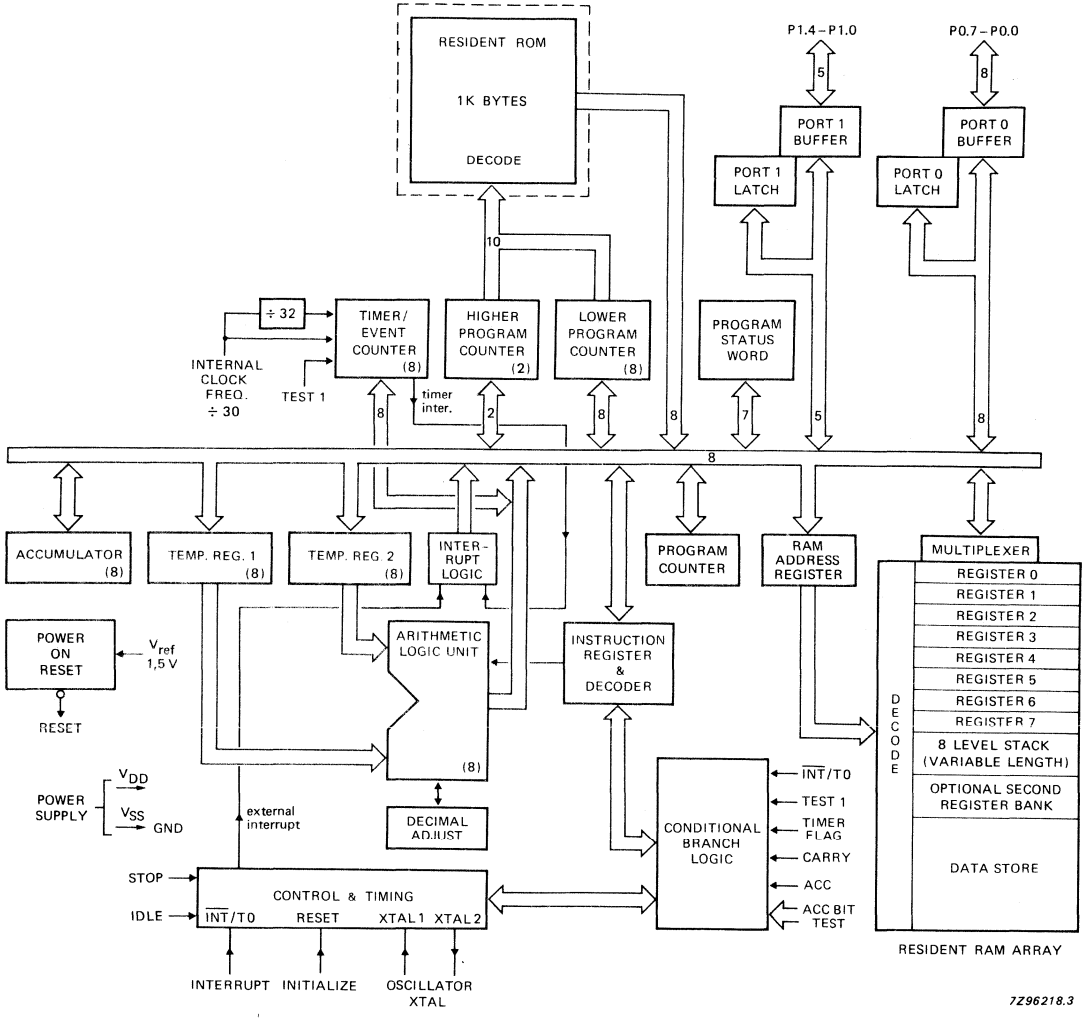


Fig. 1 Block diagram.

3.1 Pin designation

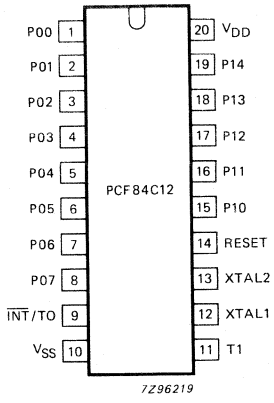


Fig. 2 Pinning diagram.

PIN DESIGNATION

1-8	P00-P07	Port 0: 8-bit quasi-bidirectional I/O port.
9	$\overline{INT}$/T0	Interrupt/Test 0: external interrupt input (sensitive to negative-going edge)/test input pin; when used as a test input directly tested by conditional branch instructions JT0 and JNT0.
10	V_{SS}	Ground: circuit earth potential.
11	T1	Test 1: test input pin, directly tested by conditional branch instructions JT1 and JNT1. T1 also functions as an input to the 8-bit timer/event counter, using the STRT CNT instruction.
12	XTAL 1	Oscillator input: crystal which determines the internal oscillator frequency or the external clock generator.
13	XTAL 2	Oscillator output
14	RESET	Reset input: used to initialize the processor (active HIGH), or output of power-on-reset circuit.
15-19	P10-P14	Port 1: quasi-bidirectional port in parallel port mode.
20	V_{DD}	Power supply: 2,5 V to 5,5 V.

4.0 FUNCTIONAL DESCRIPTION

4.1 Program memory

The program memory consists of 1 K bytes, in a read-only memory (ROM). Each location is directly addressable by the program counter. The memory is mask-programmed at the factory. Figure 3 shows the program memory map.

Three program memory locations are of special importance:

- Location 0; contains the first instruction to be executed after the processor is initialized (RESET),
- Location 3; contains the first byte of an external interrupt service subroutine,
- Location 7; contains the first byte of a timer/event counter interrupt service subroutine.

4.2 Data memory

Data memory consists of 64 bytes, random-access data memory (RAM). All locations are indirectly addressable using RAM pointer registers; up to 16 designated locations are directly addressable. Memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Figure 4 shows the data memory map.

4.3 Working registers

Locations 0 to 7 are designated as working registers, directly addressable by the direct register instructions. Ease of addressing, and a minimum requirement of instruction bytes to manipulate their contents, makes these locations suitable for storing frequently addressed intermediate results. This bank of registers can be selected by the SEL RB0 instruction.

Executing the select register bank instruction SEL RB1, designates locations 24 to 31 as working registers, instead of locations 0 to 7, and these are then directly addressable. This second bank of working registers may be used as an extension of the first or reserved for use during interrupt service subroutines saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations.

All RAM locations make efficient program loop counters when used with the decrement register and test instruction DJNZ.

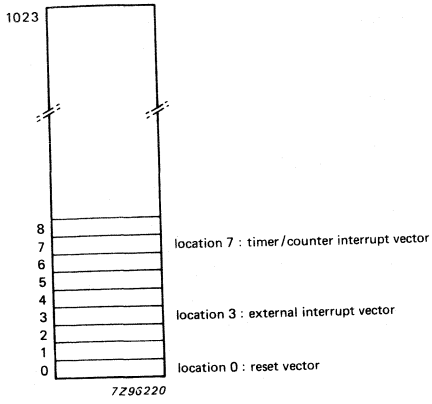


Fig. 3 Program memory map.

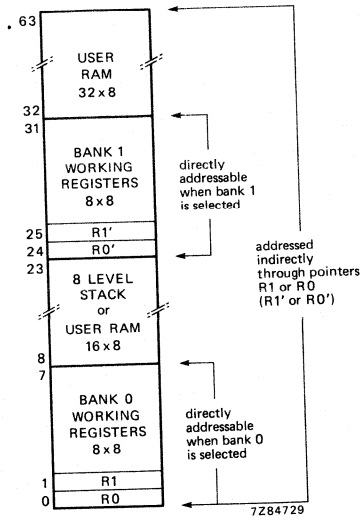


Fig. 4 Data memory map.

4.4 Interrupts (see Fig. 5)

When the external interrupt is enabled, a HIGH-to-LOW transition on the $\overline{INT}$/T0 input initiates an external interrupt subroutine which causes a CALL to program memory location 3 following completion of the current instruction.
The interrupt must remain enabled until the interrupt instruction is completed, otherwise the next instruction of the main program will be executed. A timer/event counter overflow forces a CALL to location 7 when the timer interrupt is enabled.

When an interrupt subroutine starts, the contents of the program counter and bits 4, 6, and 7 of the PSW have been saved in the program counter stack. Accumulator contents have to be saved by software. Interrupt acknowledgement can be carried out by software via port pins. All interrupt subroutines must reside in memory bank 0

Since the interrupt system is single level, once an interrupt is detected, all further interrupt requests are latched, but ignored, pending a RETR instruction to re-enable the interrupt input logic. After executing RETR, the program continues in the main part; this is independent of the occurance of a second interrupt during the running of the first routine. If 2 interrupts occur simultaneously, their priority is:

- (1) external
- (2) timer/event counter

An external interrupt can be generated by using the timer/counter in the event counter mode. The counter is first preset to (FF H), then EN TCNTI instruction is executed. A LOW-to-HIGH transition of the T1 input will then initiate an interrupt subroutine and cause a CALL to location 7.

On execution of a DIS I instruction, the PCF84C12 always clears the digital filter/latch and the External Interrupt Flag.

The Timer Flag (TF) is reset only when the JTF or JNTF instruction is executed or after RESET.

The Timer Interrupt Flag is set when timer overflow occurs, only if the timer interrupt is enabled.

The microcontroller will exit the IDLE mode when one of the following two interrupts is enabled:

- External
- Timer/event counter

There is no internal pull-up or pull-down device connected to the external interrupt input (pin 9). If required pin 9 must be externally connected to a resistor ($R = \leqslant 100\,k\Omega$). When the external interrupt is not used pin 9 must be connected to V_{DD}.

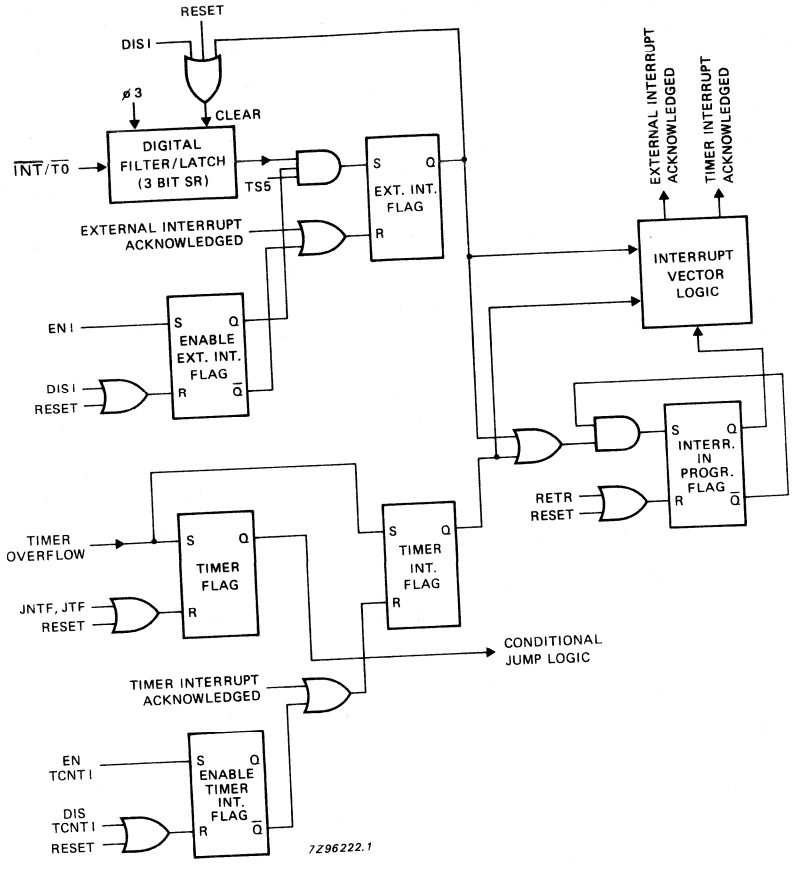


Fig. 5 Interrupt logic.

6.2 PCF84C430 microcontroller

CONTENTS—PCF84C430 8-BIT MICROCONTROLLER WITH LCD DRIVER

		page
1.0	DESCRIPTION	6.2-5
2.0	FEATURES	6.2-5
3.0	PACKAGE OUTLINE	6.2-5
3.1	Pin designation	6.2-7
4.0	FUNCTIONAL DESCRIPTION	6.2-9
4.1	Program memory	6.2-9
4.2	Data memory	6.2-9
4.3	Working registers	6.2-9
4.4	I/O facilities	6.2-10
4.4.1	Parallel ports	6.2-11
4.5	Derivative I/O addresses	6.2-11
4.6	Interrupt logic	6.2-12
4.7	Liquid crystal display driver	6.2-13
4.7.1	LCD bias generation	6.2-14
4.7.2	LCD voltage selector	6.2-15
4.7.3	LCD drive mode waveforms	6.2-16
4.7.4	LCD timing	6.2-21
4.7.5	LCD segment driver outputs	6.2-21
4.7.6	Backplane outputs	6.2-21
4.7.7	LCD segment display registers	6.2-22

1.0 DESCRIPTION

The PCF84C430 microcontroller is a derivative of the PCF84CXX family of microcontrollers and is manufactured in CMOS technology. For detailed information see the PCF84C430 data sheet.

The PCF84C430 contains a PCF84CXX core CPU and is completely software compatible. In addition, the PCF84C430 contains an LCD driver supporting four back planes and a maximum driving capacity of up to 96 segments.

The PCF84C430 has 20 quasi-bidirectional I/O port lines, plus a derivative 4-bit port, a serial I/O interface, a single-level vectored interrupt circuit, an 8-bit timer event counter and on-board clock oscillator and clock circuits.

2.0 FEATURES

- 8-bit CPU, ROM, RAM, I/O in a single 64-lead QFP package
- 4 K ROM bytes
- 128 RAM bytes
- On-chip LCD driver with 24 outputs (max. 96 segments)
- LCD multiplexing rates at; 1:1 (static), 1:2, 1:3, and 1:4
- Low-power oscillator for LCD driver during STOP mode
- 25 quasi-bidirectional I/O port lines are configured as; two 8-bit ports, a 1-bit port (shared with SDA) and an 8-bit derivative port
- Two test inputs: one of which is also the external interrupt input
- Single-level vectored interrupts: external, timer/event counter, serial I/O
- I^2C-bus hardware interface for serial data transfer on two separate lines
- 8-bit programmable timer/event counter
- Clock frequency 100 kHz to 10 MHz
- Over 80 instructions (based on MAB8048) all of 1 or 2 cycles
- Single supply voltage from 4 V to 6 V
- STOP and IDLE mode
- Power-on-reset circuit
- Operating temperature range: -40 to + 85 ^{o}C

3.0 PACKAGE OUTLINE

PCF84C430H: 64-lead quad flat-pack, plastic (SOT-208A1)

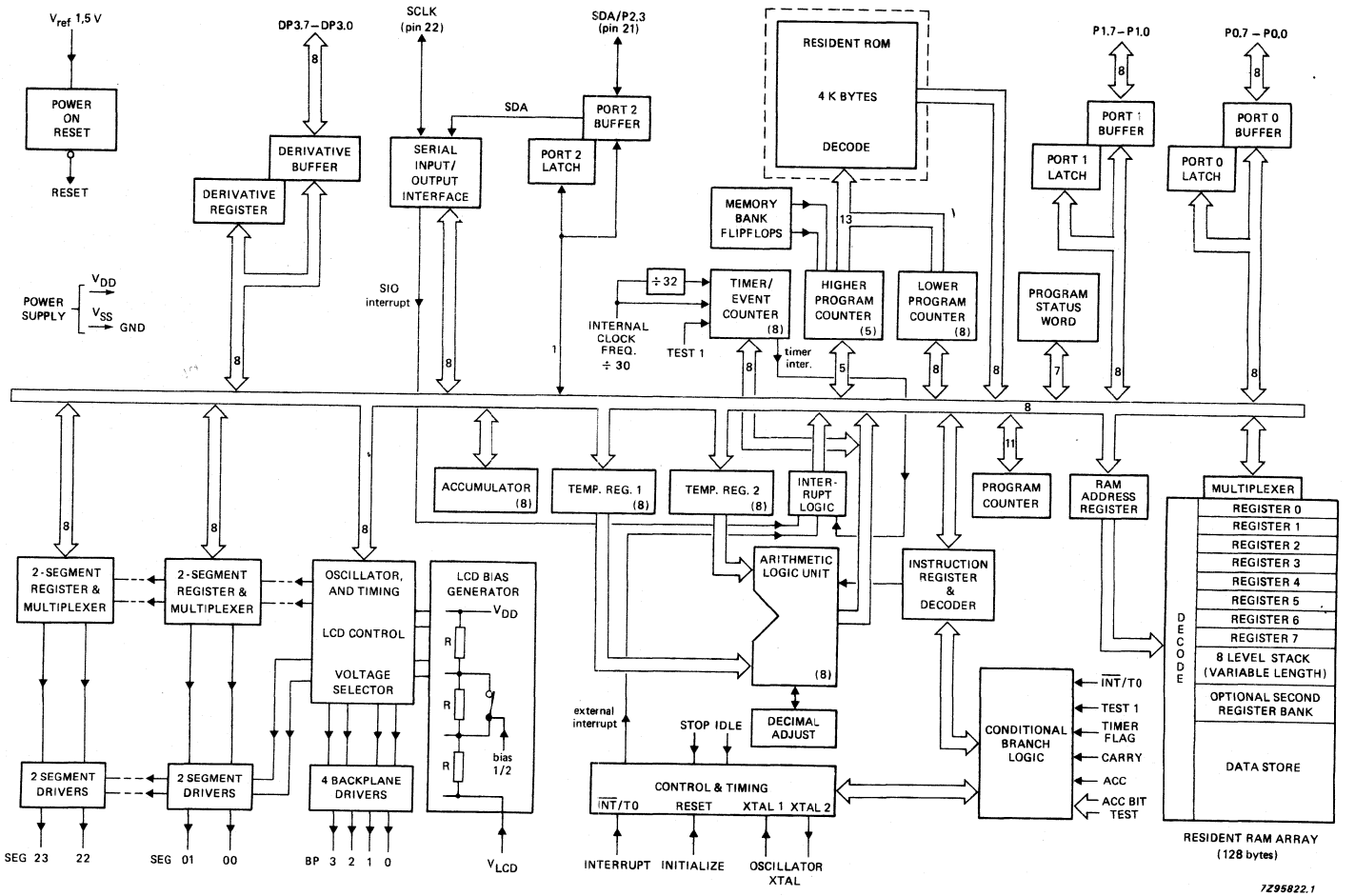


Fig. 1 Block diagram; PCF84C430.

3.1 Pin designation

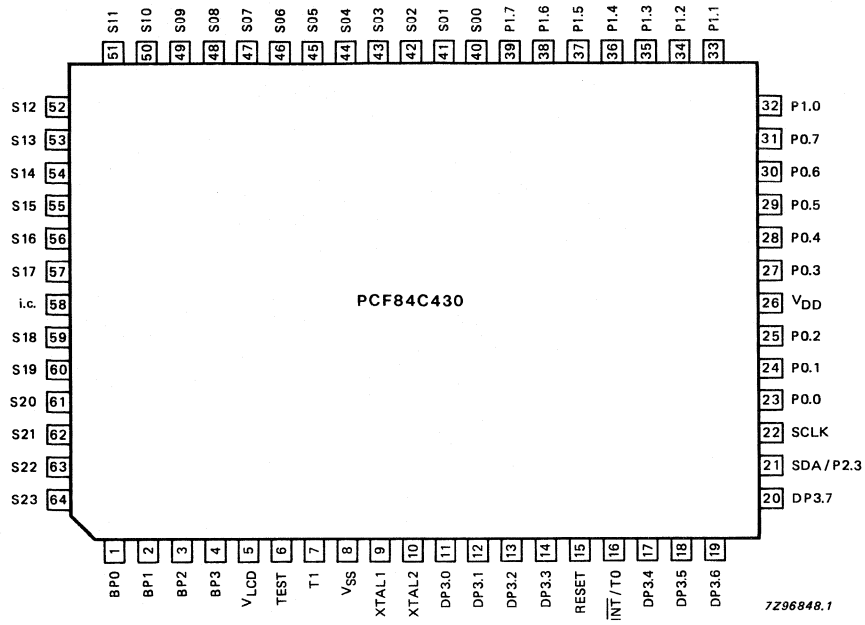


Fig. 2 Pin designation of the PCF84C430

PIN DESIGNATION

Pin	symbol	type	function
1-4	BP0-BP3	O	LCD backplane outputs.
5	V_{LCD}	O	LCD supply voltage.
6	EMU/TEST	I	Emulation control pin. Grounded when unused
7	T1	I	Test 1: test input pin, directly tested by conditional branch instructions JT1 and JNT1 T1 also functions as an input to the 8-bit timer /event counter, using the STRT CNT instruction.
8	V_{SS}	I	Ground: circuit earth potential.
9	XTAL1	I	Oscillator input: crystal which determines the internal oscillator frequency or the external clock generator.
10	XTAL2	I/O	Connection to other side of timing component.
11-14	DP3.0-DP3.3	I/O	Derivative port 3: 8-bit parallel port LSBs.
15	RESET	I/O	Reset input: used to initialize the processor (active HIGH), or output of power-on-reset circuit
16	INT/T0	I	Interrupt/Test 0: external interrupt input (sensitive to negative going edge)/test input; when used as a test input directly tested by conditional branch instructions JT0 and JNT0.
17-20	DP3.4-DP3.7	I/O	Derivative port 3: 8-bit parallel port MSBs.
21	SDA/P23	I/O	Serial data: input/output in serial I/O mode. If not selected as serial data pin, P23 functions as a quasi-bidirectional I/O line.
22	SCLK	I/O	Serial clock: bidirectional clock for serial I/O
23-25 27-31	P0.0-P0.7	I/O	Port 0: 8-bit quasi-bidirectional I/O port.
26	V_{DD}	I	Power supply: 2,5 V to 5,5 V.
32-39	P1.0-P1.7	I/O	Port 1: 8-bit quasi-bidirectional I/O port.
40-57 59-64	S00-S23	O	LCD segment driver outputs.

4.0 FUNCTIONAL DESCRIPTION

4.1 Program memory

The program memory consists of 4 K bytes, in a read-only memory (ROM). Each location is directly addressable by the program counter. The memory is mask-programmed at the factory. Figure 3 shows the program memory map.

Four program memory locations are of special importance:

- Location 0; contains the first instruction to be executed after the processor is initialized (RESET),
- Location 3; contains the first byte of an external interrupt service subroutine,
- Location 5; contains the first byte of a serial I/O service subroutine,
- Location 7; contains the first byte of a timer/event counter interrupt service subroutine.

Program memory is arranged in banks of 2 K bytes, which are selected by SEL MB instructions. The program memory is further divided into location 'pages', each of 256 bytes. This latter division applies only for conditional branches. Memory bank boundaries can be crossed only by using the unconditional branch instructions after the appropriate memory bank has been selected. A CALL instruction can transfer control to a subroutine on any 'page'; RET and RETR instructions can transfer control from a subroutine back to the main program.

4.2 Data memory

Data memory consists of 128 bytes of random-access data memory (RAM). All locations are indirectly addressable using RAM pointer registers; up to 16 designated locations are directly addressable. Memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Figure 4 shows the data memory map.

4.3 Working registers

Locations 0 to 7 are designated as working registers, directly addressable by the direct register instructions. Ease of addressing, and a minimum requirement of instruction bytes to manipulate their contents, makes these locations suitable for storing frequently addressed intermediate results. This bank of registers can be selected by the SEL RB0 instruction.

Executing the select register bank instruction SEL RB1, designates locations 24 to 31 as working registers, instead of locations 0 to 7, and these are then directly addressable. This second bank of working registers may be used as an extension of the first or reserved for use during interrupt service subroutines saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations.

All RAM locations make efficient program loop counters when used with the decrement register and test instruction DJNZ.

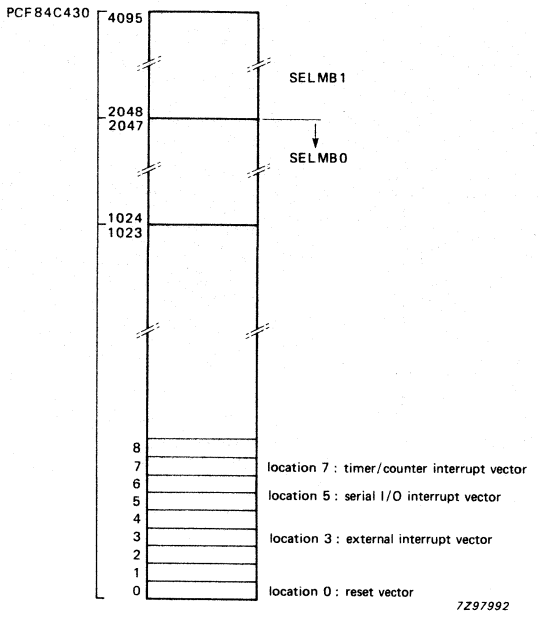


Fig. 3 Program memory map.

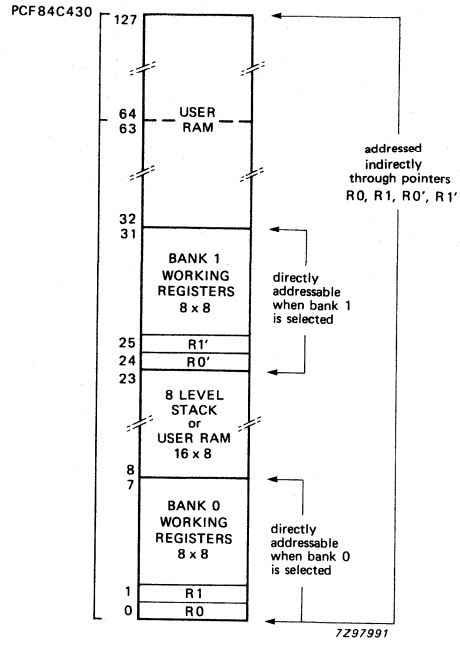


Fig. 4 Data memory map.

4.4 I/O facilities

The PCF84C430 has 24 general I/O lines arranged as:

- Port 0 parallel port of 8 lines (P0.0 to P0.7)
- Port 1 parallel port of 8 lines (P1.0 to P1.7)
- Port 2 I/O port of 1 line (SDA/P23) active when not selected as serial I/O
- Port 3 a derivative 8-bit parallel port (DP3.0 to DP3.7)

In addition 4 specialized I/O lines are comprised:

- SCLK serial I/O clock line
- SDA/P23 serial I/O data line
- INT/T0 external interrupt and test input. When used as a test input T0 can be directly tested by conditional branch instructions JT0 and JNT0
- T1 test input which can alter program sequences when tested by conditional jump instructions JT1 and JNT1. T1 also functions as an input to the 8-bit timer/event counter.

4.4.1 Parallel ports

All parallel ports can be used as outputs or inputs, their structure is quasi-bidirectional. Output data written to a port is latched and remains unchanged until rewritten. Input data is not latched and so must be present until read by an input instruction.

Input lines are fully CMOS compatible, output lines can drive one TTL or CMOS load.

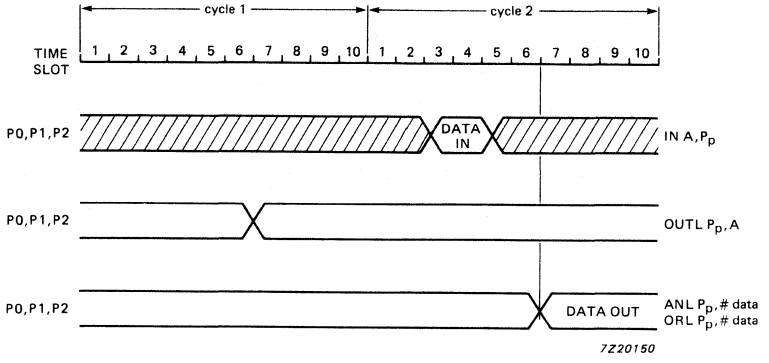


Fig. 5 Shows the timing diagram for all ports using IN, OUTL, ANL and ORL instructions. For the OUTL instruction data changes on time slot 7 of cycle 1. For the ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

4.5 Derivative I/O addresses

For extension of the I/O facilities, instructions MOV A,Dx and MOV Dx,A are added to the instruction set. The derivative register address is the second byte of the instruction. The address map for the derivative registers is given in Table 1.

Table 1 address map of PCF84C430 derivative registers.

Dx	R/W	bits	description
4	R/W	8	PR3 derivative I/O Port 3 register
5	R	8	PP3 derivative I/O Port 3 input pins
16	R/W	8	LCDC LCD control byte
17-28	R/W	8	LCDB1-LCDB12 12 consecutive segment data bytes

4.6 Interrupt logic

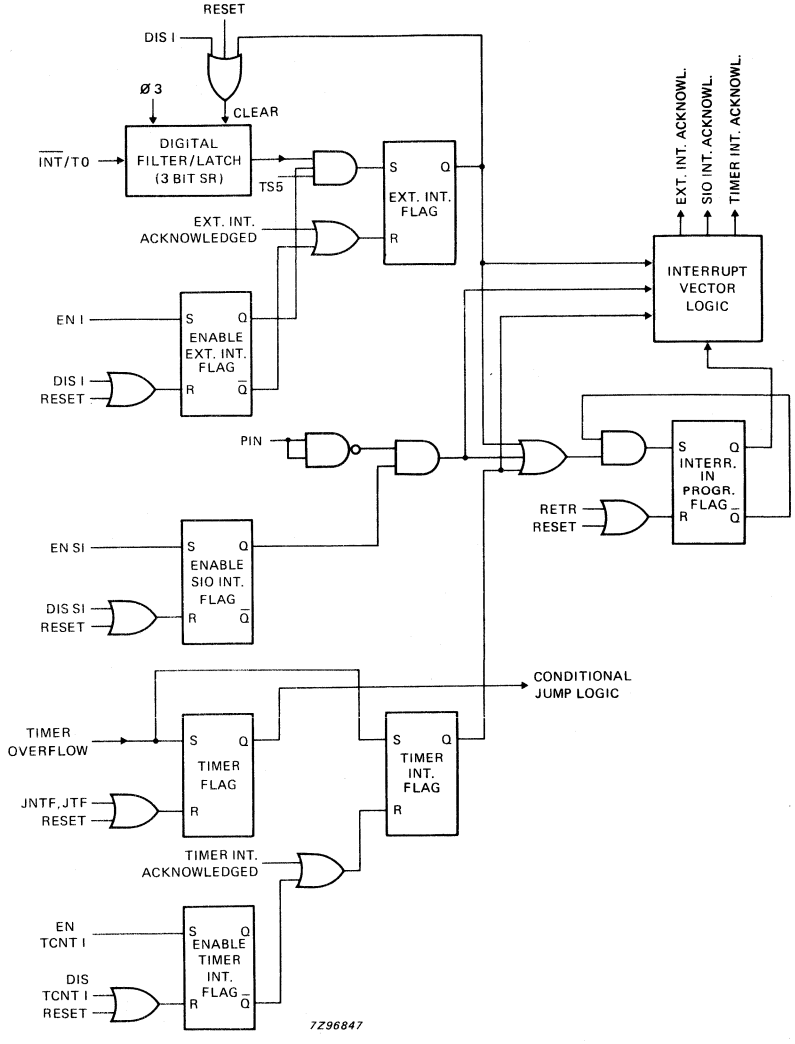


Fig. 6 Interrupt logic.

Notes to figure 6

1. INT/TO negative edge is always latched in the digital filter/latch.
2. Correct interrupt timing is ensured when INT/TO is HIGH for >4 CP followed by a LOW for >7 CP.
3. When the interrupt in progress flag is set, further external and timer interrupts are latched but ignored, until RETR is executed.
4. A DIS I instruction always clears a pending external interrupt.
5. For all flip-flops, RESET overules set.

4.7 Liquid crystal display driver

The PCF84C430 has a display driver which interfaces to almost any liquid crystal display (LCD) which has a low multiplex rate. The interface delivers drive signals for any static or multiplexed LCD panel that contains up to four backplanes and up to 24 segments. Fig. 7 shows a block diagram of the LCD driver.

The following features are incorporated:

- Selectable backplane drive configuration; static or 2/3/4 backplane multiplexing
- Selectable display bias configuration; 1/2 or 1/3 internal LCD bias generation
- 24 individual segment drivers can be used to provide;
  - up to twelve 8-segment numeric characters
  - up to six 15+1 segment alphanumeric characters
  - graphics using up to 96 elements
  - twelve 8-bit derivative registers for display data bits
- LCD and logic voltage supplies may be separated
- An LCD operating voltage range of between 2 V to 5 V
- Oscillator has low-power consumption for display in processor STOP mode.

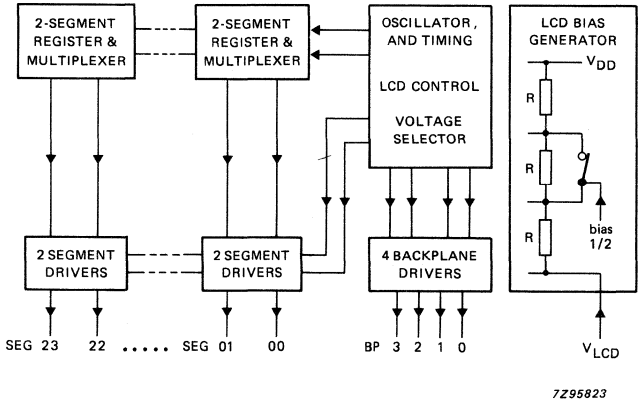


Fig. 7 Block diagram of the LCD driver.

The display configurations possible with the PCF84C430 depend upon the number of active backplane outputs required. A selection of display configurations is given in Table 2.

Table 2 Selection of display configurations

number of backplanes	number of segments	7-segment numeric	14-segment alphanumeric	dot matrix
4	96	12 digits + 12 indicator symbols	6 chars + 12 indicator symbols	96 dots (4 x 24)
3	72	9 digits + 9 indicator symbols	5 chars + 2 indicator symbols	72 dots (3 x 24)
2	48	6 digits + 6 indicator symbols	3 chars + 6 indicator symbols	48 dots (2 x 24)
1	24	3 digits + 3 indicator symbols	1 char + 10 indicator symbols	24 dots (1 x 24)

All of the display configurations given in Table 2 can be implemented in the typical system shown in Fig. 8. The appropriate biasing voltages for the multiplexed LCD wave forms are generated internally.

At power-on all the LCD driver control register bits are cleared. The LCD display is not affected by executing a STOP instruction.

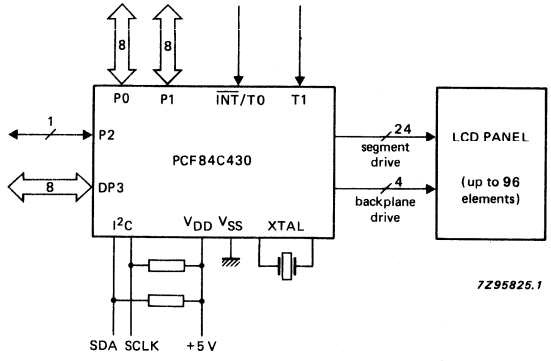


Fig. 8 Typical system configuration

4.7.1 LCD bias generation

The LCD operating voltage (V_{op}) is the result of V_{DD} - V_{LCD}. V_{op} should be chosen so that the off voltage ($V_{off(rms)}$) is just below the threshold voltage (V_{th}), typically when the LCD exhibits 10% contrast. The LCD voltage may be temperature compensated externally through the LCD supply. Fractional LCD biasing voltages are obtained from an internal voltage divider of three resistors connected between V_{DD} and V_{LCD}. The centre resistor may be switched out of circuit to provide a 1/2 bias voltage level for a 1:2 multiplex configuration.

4.7.2 LCD voltage selector

The LCD voltage selector coordinates the multiplexing of the LCD according to the selected drive configuration. The operation of the voltage selector is controlled by the MODE bits in the LCD control byte. The biasing configurations that apply to the preferred mode of operation, together with the biasing characteristics as functions of $V_{op} = V_{DD} - V_{LCD}$ and resulting discrimination ratios (D), are given in Table 3.

Table 3 LCD drive modes and characteristics

LCD drive mode	number of backplanes	LCD bias configuration	number of levels	$\frac{V_{off(rms)}}{V_{op}}$	$\frac{V_{on(rms)}}{V_{op}}$	$D = \frac{V_{on(rms)}}{V_{off(rms)}}$
static	1	static	2	0	1	∞
1:2	2	1/2	3	0,354	0,791	2,236
1:2	2	1/3	4	0,333	0,745	2,236
1:3	3	1/3	4	0,333	0,638	1,915
1:4	4	1/3	4	0,333	0,577	1,7321

Multiplex drive ratios of 1:3 and 1:4 with 1/2 bias are possible, but the discrimination and contrast ratios are reduced thus:

(1,732 for 1:3 or 1,528 for 1:4)

There is an advantage however, that these modes lead to a reduction in V_{op} as follows;

1:3 multiplex (1/2 bias). $V_{op} = 2,449\ V_{off(rms)}$

1:4 multiplex (1/2 bias). $V_{op} = 2,309\ V_{off(rms)}$

This compares with $V_{op} = 3\ V_{off(rms)}$ when 1/3 bias is used.

4.7.3 LCD drive mode waveforms

The static LCD drive mode is used when a single backplane is provided in the LCD. Backplane and segment drive waveforms are shown in Fig. 9.

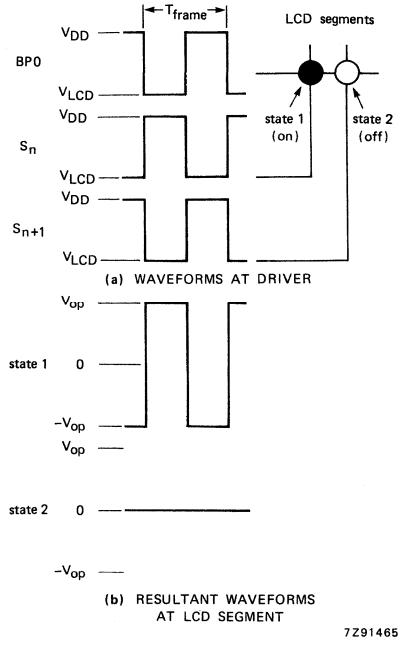


Fig. 9 Static drive mode waveforms.

When two backplanes are provided in the LCD; the 1:2 multiplex drive mode applies. The PCF84C430 allows use of 1/2 or 1/3 bias in this mode as shown in Figs. 10 and 11.

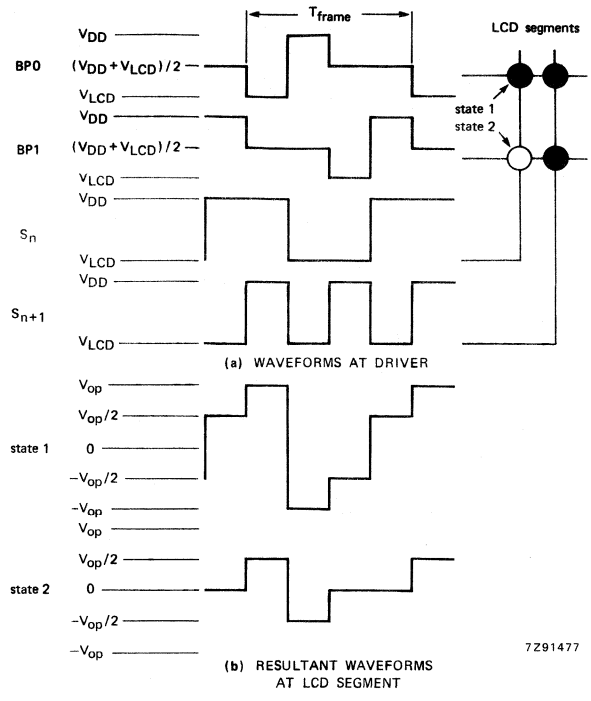


Fig. 10 Waveforms for the 1:2 multiplex drive mode with 1/2 bias.

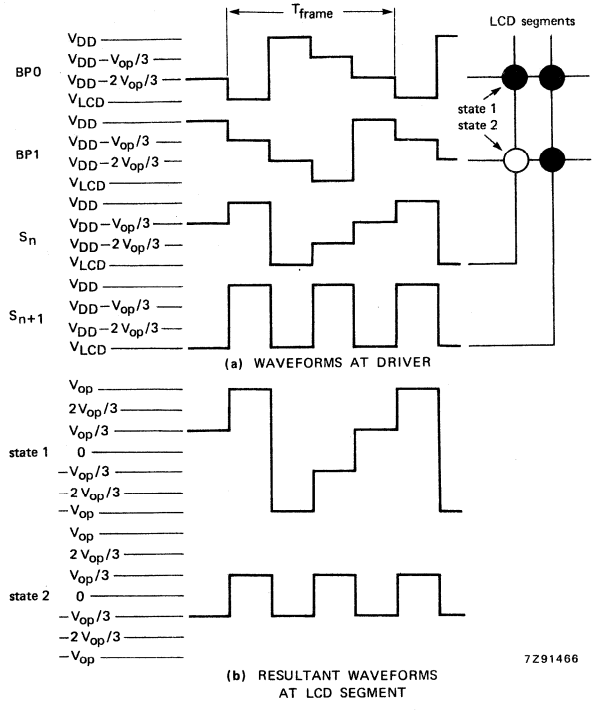


Fig. 11 Wave forms for the 1:2 multiplex drive mode with 1/3 bias.

The backplane and segment drive waveforms for the 1:3 multiplex drive mode (three LCD backplanes) and for the 1:4 multiplex drive mode (four LCD backplanes) are shown in Figs. 12 and 13 respectively.

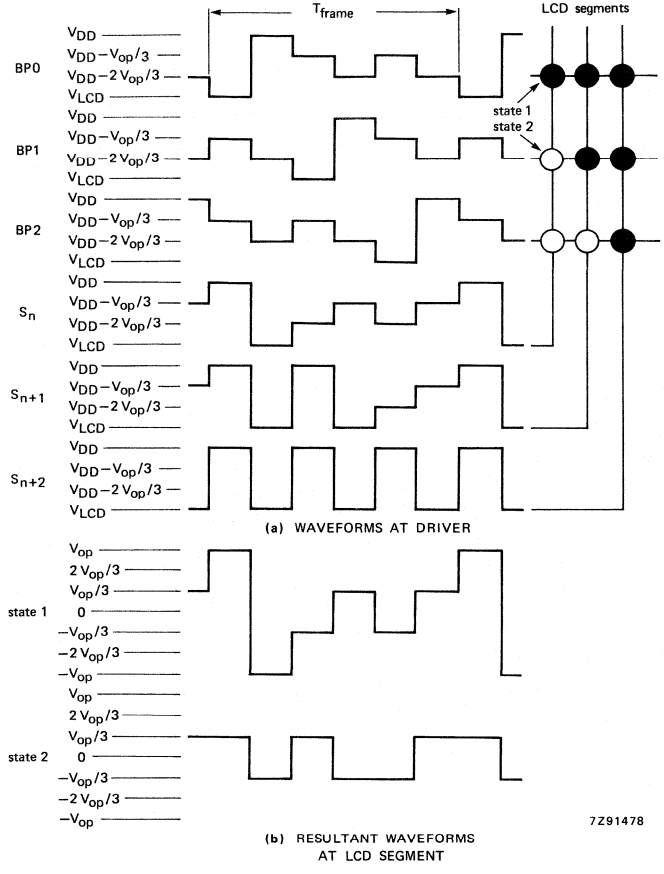


Fig. 12 Waveforms for the 1:3 multiplex drive mode.

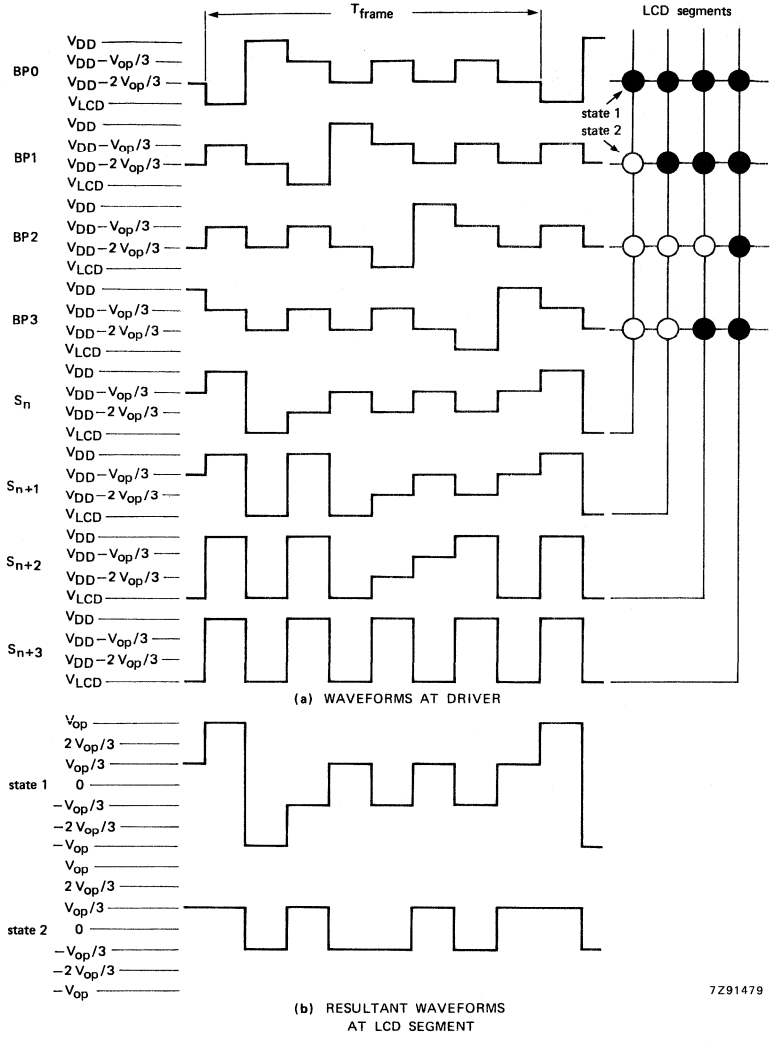


Fig. 13 Waveforms for the 1:4 multiplex drive mode.

4.7.4 LCD timing

The LCD clocking is performed by a separate, self-contained, on-chip oscillator with a nominal frequency of 30 kHz. The display may be kept active while in the STOP mode using a very low suply current. Fig. 14 below shows the LCD timing control.

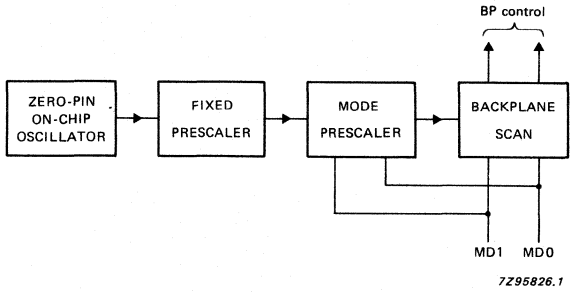


Fig. 14 LCD timing control.

4.7.5 LCD segment driver outputs

The LCD drive section includes 24 segment outputs (S0 to S23) which should be connected directly to the LCD. The segment data bits (one nibble per segment) are multiplexed to the outputs in accordance with the backplane signals. If less than the 24 segment outputs are required then the unused driver outputs should be left open.

4.7.6 Backplane outputs

The LCD drive section includes 4 backplane outputs (BP0-BP3) which should be connected directly to the LCD. These backplane output signals are generated in accordance with the selected LCD drive mode. If less than four backplane outputs are required then the unused backplane outputs should be left open-circuit.

In the 1:3 multiplex drive mode, BP3 carries the same signal as BP0, therefore these two outputs can be tied together to give enhanced drive capabilities. In the 1:2 multiplex drive mode, BP0 and BP3, BP1 and BP2 respectively carry the same signals and may also be paired to increase the drive capabilities. In the static drive mode the same signal is carried by all four backplane outputs and may be connected in parallel to give a very high drive capability.

4.7.7 LCD segment display registers

The 12 segment display registers are 8-bit derivative (read/write) registers which store LCD segment data. A segment register bit which is set to a logic 1 indicates the 'on' state of the corresponding LCD segment, similarly, a logic '0' indicates the 'off' state. There is a one-to-one relationship between the LCD segment register bits and the segment outputs. Every byte is divided into two nibbles. Each nibble corresponds to a segment driver; the first byte contains the bits earmarked for segment 1 and 2 etc. The first bit of a nibble will correspond to backplane 0, and the second to backplane 1 and so on. Fig. 16 shows the display register bit map. Each bit is shown in the form Sxx.n, where (xx) is the segment output and (n) the backplane output.

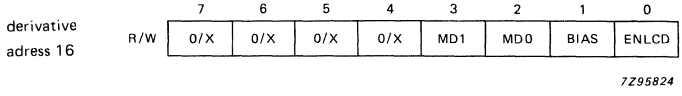


Fig. 15 LCDC: LCD control byte

MDS	MDO	LCD drive mode
0	0	static
0	1	1 : 2
1	0	1 : 3
1	1	1 : 4

BIAS	LCD bias configuration
1	1/3
0	1/2

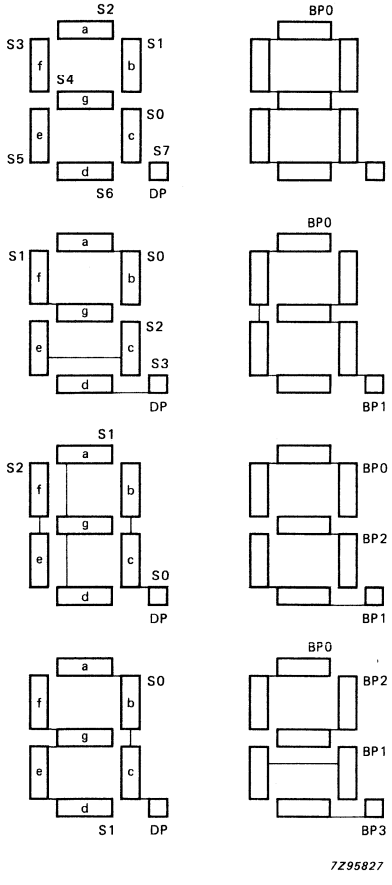
ENLCD	Display status
0	LCD display blank
1	LCD dsisplay on

LCDD : LCD display data

derivative address									
17	R/W	S 0.0	S 0.1	S 0.2	S 0.3	S 1.0	S 1.1	S 1.2	S 1.3
18	R/W	S 2.0	S 2.1	S 2.2	S 2.3	S 3.0	S 3.1	S 3.2	S 3.3
19	R/W	S 4.0	S 4.1	S 4.2	S 4.3	S 5.0	S 5.1	S 5.2	S 5.3
20	R/W	S 6.0	S 6.1	S 6.2	S 6.3	S 7.0	S 7.1	S 7.2	S 7.3
21	R/W	S 8.0	S 8.1	S 8.2	S 8.3	S 9.0	S 9.1	S 9.2	S 9.3
22	R/W	S10.0	S10.1	S10.2	S10.3	S11.0	S11.1	S11.2	S11.3
23	R/W	S12.0	S12.1	S12.2	S12.3	S13.0	S13.1	S13.2	S13.3
24	R/W	S14.0	S14.1	S14.2	S14.3	S15.0	S15.1	S15.2	S15.3
25	R/W	S16.0	S16.1	S16.2	S16.3	S17.0	S17.1	S17.2	S17.3
26	R/W	S18.0	S18.1	S18.2	S18.3	S19.0	S19.1	S19.2	S19.3
27	R/W	S20.0	S20.1	S20.2	S20.3	S21.0	S21.1	S21.2	S21.3
28	R/W	S22.0	S22.1	S22.2	S22.3	S23.0	S23.1	S23.2	S23.3

Fig. 16 Display register bit map

In the static drive mode the eight display data bits are placed in bit 0 of the nibbles of four succesive derivative display registers. In the 1:2 multiplex drive mode, these eight bits are placed in bits 0 and 1 of the nibbles of two succesive registers. In the 1:3 multiplex drive mode the eight bits are placed in bits 0, 1 and 2 of three succesive nibbles. In the 1:4 multiplex drive mode the eight bits are placed in bits 0,1,2 and 3 of the nibbles of the first display register. Fig. 16 shows the relationship between LCD segment layout, drive mode and the LCD segment derivative register bit pattern.



byte

17	c	–	–	–	b	–	–	–
18	a	–	–	–	f	–	–	–
19	g	–	–	–	e	–	–	–
20	d	–	–	–	Dp	–	–	–

static mode

byte

17	a	b	–	–	f	g	–	–
18	e	c	–	–	d	Dp	–	–
19	–	–	–	–	–	–	–	–
20	–	–	–	–	–	–	–	–

1:2 multiplex mode

byte

17	b	Dp	c	–	a	d	g	–
18	f	e	–	–	–	–	–	–
19	–	–	–	–	–	–	–	–
20	–	–	–	–	–	–	–	–

1:3 multiplex mode

byte

17	a	c	b	Dp	f	e	g	d
18	–	–	–	–	–	–	–	–
19	–	–	–	–	–	–	–	–
20	–	–	–	–	–	–	–	–

1:4 multiplex mode

Fig. 17 LCD segment layout and register bit pattern.

6.3 The PCF84C85 microcontroller

CONTENTS – PCF84C85 8-BIT MICROCONTROLLER

		page
1.0	DESCRIPTION	6.3-5
2.0	FEATURES	6.3-5
3.0	PACKAGE OUTLINES	6.3-5
3.1	Pin designation	6.3-7
4.0	FUNCTIONAL DESCRIPTION	6.3-8
4.1	Program memory	6.3-8
4.2	Data memory	6.3-8
4.3	Working registers	6.3-8
4.4	I/O facilities	6.3-9
4.5	Interrupts	6.3-10

1.0 DESCRIPTION

The PCF84C85 microcontroller is manufactured in CMOS, and is designed to be an efficient controller as well as an arithmetic processor. The instruction set is based on that of the MAB8048 and is software compatible with the PCF84CXX family. The PCF84C85 has two additional derivative ports and the microcontroller has bit handling abilities and facilities for both binary and BCD arithmetic.

For detailed information on the PCF84CXX see both the data sheet and the separate section in this manual.

2.0 FEATURES

- 8-bit CPU, ROM, RAM, I/O in a single 40-lead DIL or SO package
- 8 K ROM
- 256 RAM bytes
- 32 quasi-bidirectional I/O port lines
- Two test inputs: one of which is also the external interrupt input
- Single-level vectored interrupts: external, timer/event counter, serial I/O
- I^2C hardware interface for two-line serial data transfer (serial I/O data via an existing port line and clock via a dedicated line)
- 8-bit programmable timer/event counter
- Clock frequency 100 kHz to 10 MHz
- Over 80 instructions (based on MAB8048) all of 1 or 2 cycles
- Single supply voltage from 2,5 V to 5,5 V
- STOP and IDLE mode
- Power-on-reset circuit
- Operating temperature range: -40 to + 85 ^{o}C

3.0 PACKAGE OUTLINES

PCF84C85 P: 40-lead DIL; plastic (SOT-129).
PCF84C85 T: 40-lead; mini-pack (SOT-158).

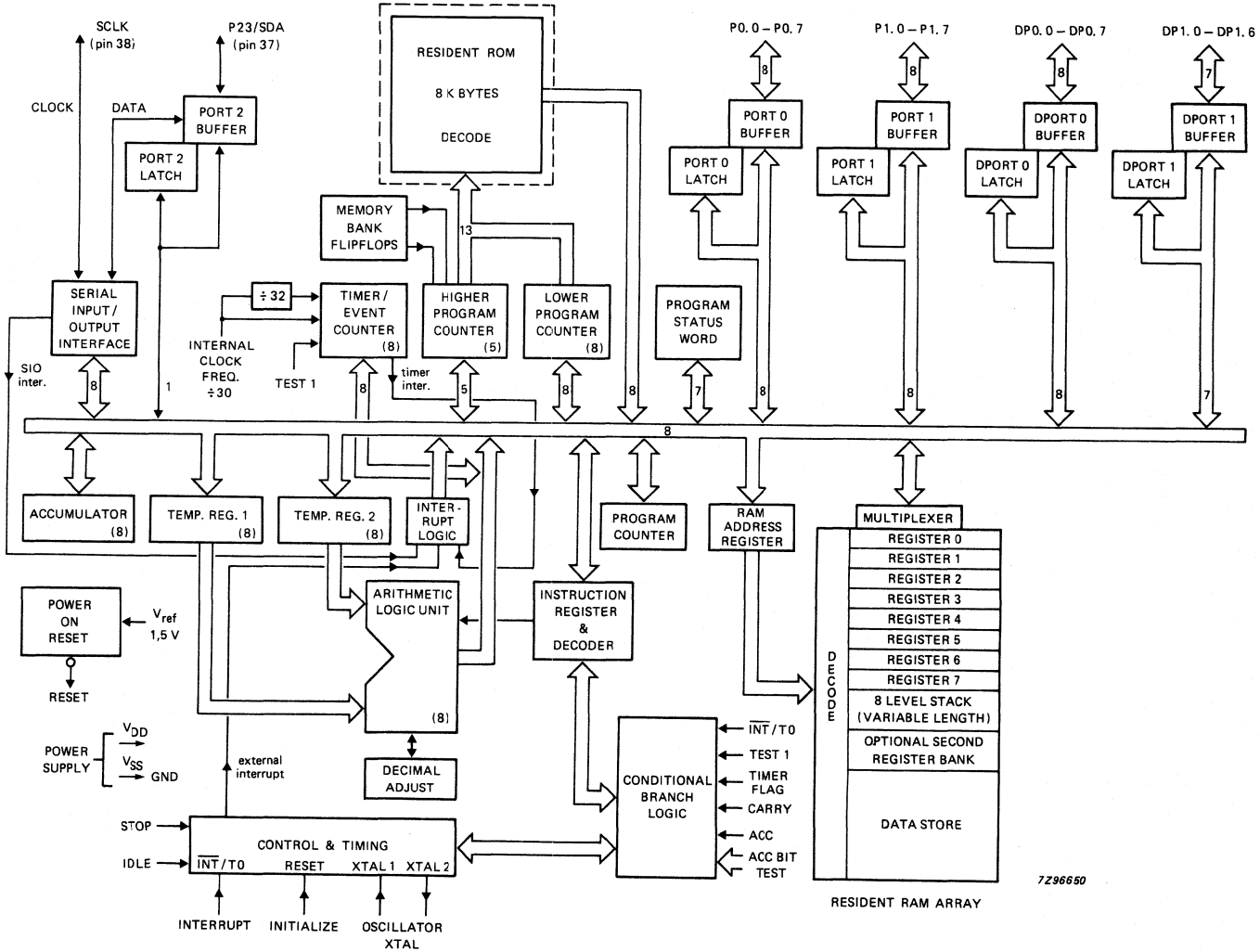


Fig. 1 Block diagram; PCF84C85.

3.1 Pin designation

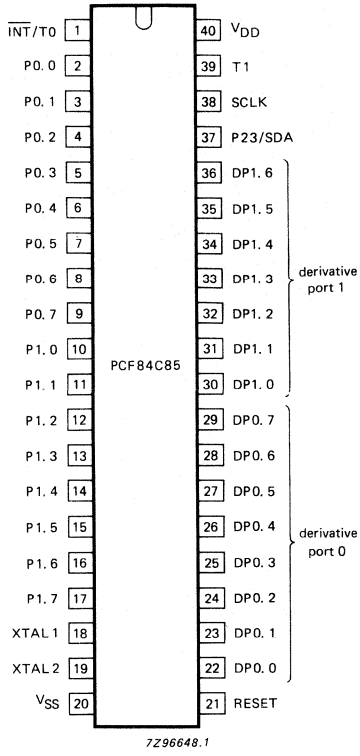


Fig. 2 Pinning diagram; PCF84C85.

PIN	DESIGNATION	
38	SCLK	Clock: bidirectional clock for serial I/O.
1	$\overline{\text{INT}}$/T0	Interrupt/Test 0: external interrupt input (sensitive to negative-going edge)/test input pin; when used as a test input directly tested by conditional branch instructions JT0 and JNT0.
39	T1	Test 1: test input pin, directly tested by conditional branch instructions JT1 and JNT1. T1 also functions as an input to the 8-bit timer/event counter, using the STRT CNT instruction.
18	XTAL 1	Crystal input: connection to timing component (crystal) which determines the frequency of the internal oscillator; also the input for an external clock source.
19	XTAL 2	connection to the other side of the timing component
21	RESET	Reset input: used to initialize the processor (active HIGH), or output of power-on-reset circuit.
2-9	P0.0-P0.7	Port 0: 8-bit quasi-bidirectional I/O port.
10-17	P1.0-P1.7	Port 1: 8-bit quasi-bidirectional I/O port.
37	P23-SDA	Port 2: 1-bit quasi-bidirectional I/O port or serial data input/output in serial I/O mode.
22-29	DP0.0-DP0.7	Derivative port 0: 8-bit quasi-bidirectional I/O port.
30-36	DP1.0-DP1.6	Derivative port 1: 7-bit quasi-bidirectional I/O port.
20	V_{SS}	Ground: circuit earth potential.
40	V_{DD}	Power supply: 2,5 V to 5,5 V.

4.0 FUNCTIONAL DESCRIPTION

4.1 Program memory

The program memory consists of 8 K bytes, in a read-only memory (ROM). Each location is directly addressable by the program counter. The memory is mask-programmed at the factory. Figure 3 shows the program memory map.

Four program memory locations are of special importance:

- Location 0; contains the first instruction to be executed after the processor is initialized (RESET),
- Location 3; contains the first byte of an external interrupt service subroutine,
- Location 5; contains the first byte of a serial I/O interrupt service subroutine,
- Location 7; contains the first byte of a timer/event counter interrupt service subroutine.

Program memory is arranged in banks of 2 K bytes, which are selected by SEL MB instructions. The program memory is further divided into location 'pages', each of 256 bytes. This latter division applies only for conditional branches. Memory bank boundaries can be crossed only by using the unconditional branch instructions after the appropriate memory bank has been selected. A CALL instruction can transfer control to a subroutine on any 'page'; RET and RETR instructions can transfer control from a subroutine back to the main program.

4.2 Data memory

Data memory consists of 256 bytes, random-access data memory (RAM). All locations are indirectly addressable using RAM pointer registers; up to 16 designated locations are directly addressable. Memory also includes an 8-level program counter stack addressed by a 3-bit stack pointer. Figure 4 shows the data memory map.

4.3 Working registers

Locations 0 to 7 are designated as working registers, directly addressable by the direct register instructions. Ease of addressing, and a minimum requirement of instruction bytes to manipulate their contents, makes these locations suitable for storing frequently addressed intermediate results. This bank of registers can be selected by the SEL RB0 instruction.

Executing the select register bank instruction SEL RB1, designates locations 24 to 31 as working registers, instead of locations 0 to 7, and these are then directly addressable. This second bank of working registers may be used as an extension of the first or reserved for use during interrupt service subroutines saving the first bank for use in the main program. If the second bank is not used, locations 24 to 31 may serve as general purpose RAM.

The first locations of each bank contain the RAM pointer registers R0, R1, R0' and R1', which indirectly address all RAM locations.

All RAM locations make efficient program loop counters when used with the decrement register and test instruction DJNZ.

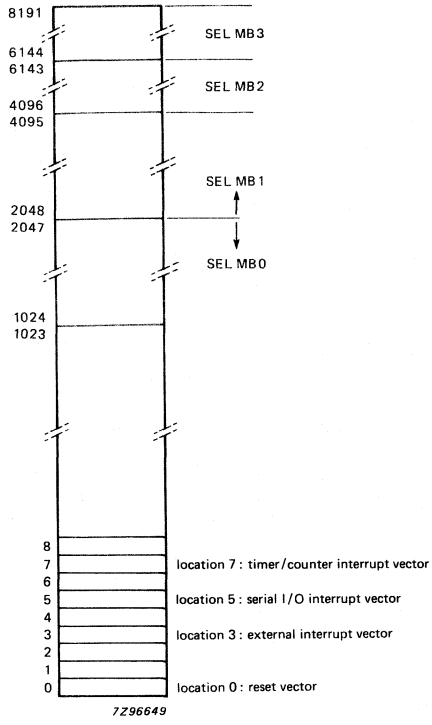


Fig. 3 Program memory map.

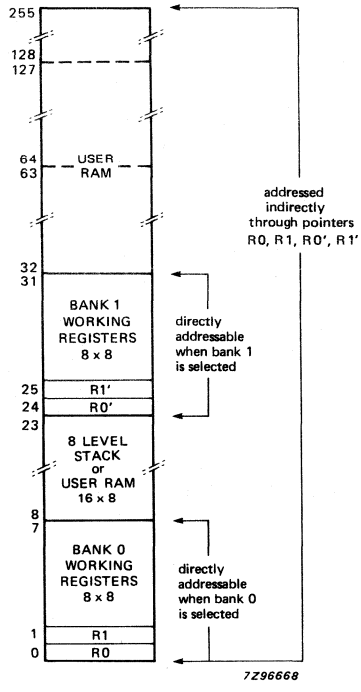


Fig. 4 Data memory map.

4.4 I/O facilities

The PCF84C85 family has 32 I/O lines arranged as:

- Port 0 parallel port of 8 lines (P0.0 to P0.7)
- Port 1 parallel port of 8 lines (P1.0 to P1.7)
- Port 2 parallel port of 1 line (P2.3)
- D Port 0 parallel port of 8 lines (DP0.0-DP.7)
- D Port 1 parallel port of 7 lines (DP1.0-DP1.6)

In addition to these the PCF84C85 also comprises four specialized I/O lines:

- SCLK I^2C-bus serial clock line
- SDA I^2C-bus serial data line (shared with P2.3)
- $\overline{INT}$/T0 external interrupt and test input. When used as a test input can be directly tested by conditional branch instructions JT0 and JNT0
- T1 test input which can alter program sequences when tested by conditional jump instructions JT1 and JNT1. T1 also functions as an input to the 8-bit timer/event counter.

4.5 Interrupts

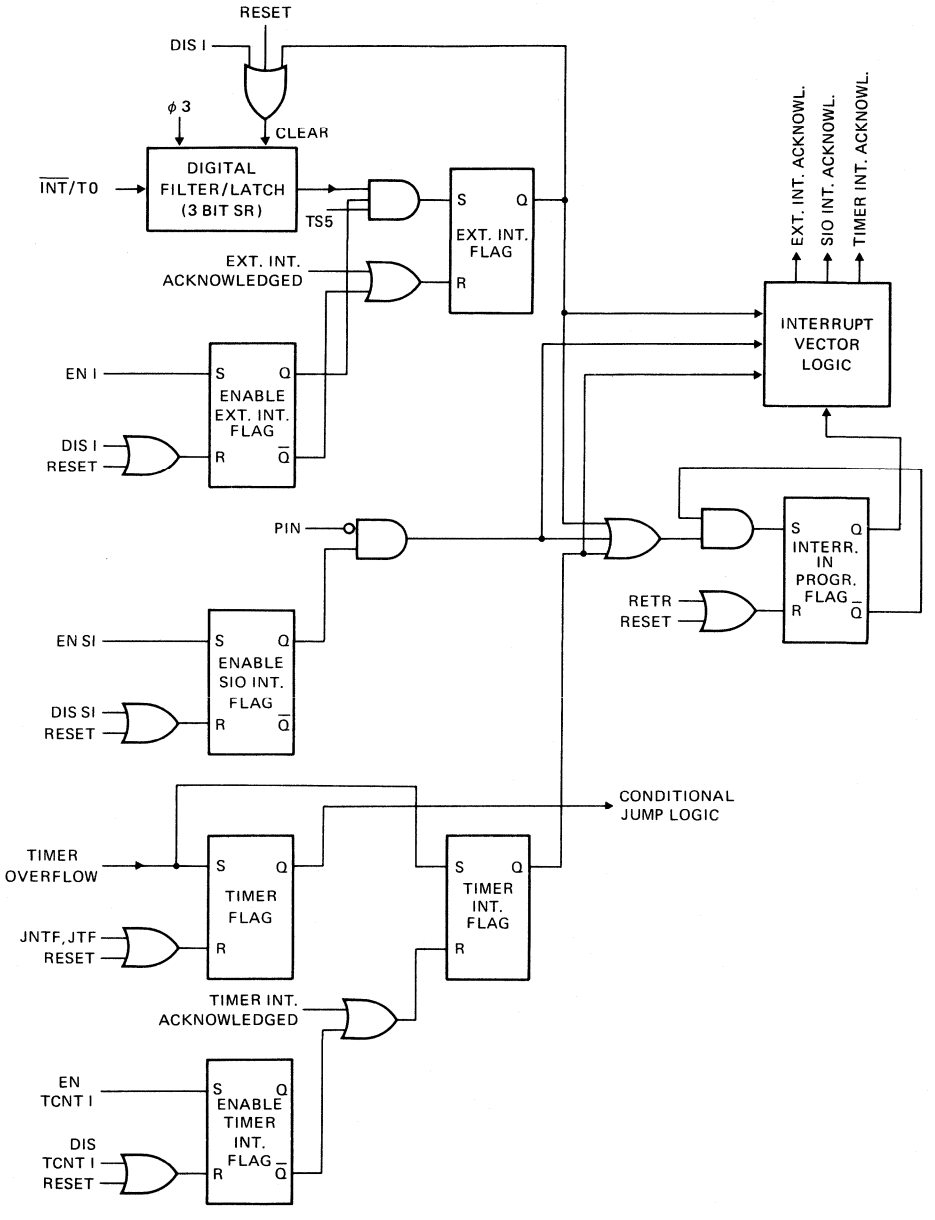


Fig. 5 Interrupt logic.

7. The MAB84X1/PCF84CXX serial I/O

CONTENTS – MAB84XX SERIAL I/O

		page
1.0	INTRODUCTION	7-4
2.0	SERIAL INPUT/OUTPUT	7-6
3.0	SERIAL BUS STRUCTURE	7-6
4.0	SERIAL DATA TRANSFER	7-6
5.0	SERIAL DATA FORMATS	7-7
6.0	OPERATING MODES OF THE SERIAL I/O INTERFACE	7-9
6.1	Master transmitter	7-9
6.2	Master receiver	7-9
6.3	Slave receiver	7-10
6.4	Slave transmitter	7-10
7.0	SERIAL I/O INTERFACE	7-10
7.1	Data shift register S0	7-13
7.2	Address register S0'	7-13
7.3	Status register S1	7-14
7.4	Clock control register S2	7-17
8.0	SERIAL I/O OPERATIONS	7-20
8.1	Arbitration procedure	7-20
8.2	Clock synchronization	7-20
9.0	PROGRAMMING	7-21
9.1	Machine state following RESET	7-21
9.2	Initialization procedure	7-21
9.2.1	Clock control register S2	7-22
9.2.2	Address register S0'	7-22
9.2.3	Status register S1	7-22
9.2.4	Enable/disable serial I/O interrupt	7-22
9.2.5	Example of an initialization procedure	7-22
9.3	Generation of a 'start' condition and the first byte	7-24
9.4	Software responses after transmission or reception of a byte	7-24
9.5	Generation of the 'stop' condition	7-25
9.6	Generation of a repeated 'start' condition	7-26
9.7	Generation of the 'stop' condition by a 'master receiver'	7-27
9.8	Flow charts for the 'master' and 'slave' functions of the SIO interface	7-28
9.9	Solutions to temporary software problems	7-13

1.0 INTRODUCTION

The increasing demand for microcontrollers in domestic and industrial applications has led to the development of the MAB84X1 family. More widespread use of distributed intelligence in today's controller sytems, has demanded the introduction of simpler, more efficient data exchange. To the MAB84X1 family this has meant the addition of a serial communication interface (SIO). The MAB8422/42 has not the full SIO hardware on-chip, but can easily handle the I^2C-bus with software.

Normally, serial data transfer imposes a heavy processing load on microcontrollers, with the serial data bus having to be regularly monitored for information. To overcome this problem the Serial I/O (SIO) interface was introduced. The MAB84X1 (SIO) interface can detect, receive and convert the serial data stream to parallel format without interrupting current program execution.

The use of serial data communication between devices reduces the number of necessary connections, leading to simpler circuit layouts and economies in both connector size and circuit board area. A hardware serial I/O allows the interconnection of any number of devices by the two-wire serial bus.

To demonstrate how serial communication works, consider two microcontrollers connected via an I^2C (Inter-IC) bus to a serial memory and a display (Fig. 1). In a typical system such as this, the device which controls the transfer of messages is called a 'master', the device(s) controlled by the master is named a 'slave'. So, in our example, the master would be either of the two microcontrollers, with the serial memory and display acting as slaves. The master device generates the timing signals for data transmission. Only one master can control the I^2C-bus at any one time. The controlling device can act as either a master transmitter or master receiver. The slave, under control of the master, can also transmit or receive.

An important feature of the I^2C-bus is its true multi-master capabilities, which means that more than one master can try to control the bus at the same time without risking data clash. During transmission, an arbitration procedure decides priority. Therefore there is no central master in the bus structure and any master transceiver, if it loses the arbitration, can be addressed as a slave by the elected master.

To explain what is meant by slave transmitters and receivers, master receivers and transmitters a definition is given below.

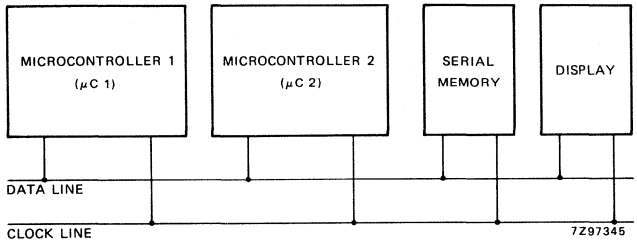
Transmitter	- The device which sends data to the bus
Receiver	- The device which receives data from the bus
Master	- The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	- The device addressed by a master
Multi-master	- More than one master can attempt to control the bus at the same time without corrupting the message

Arbitration - Procedure to ensure that if more than one master simultaneously tries to control the bus, only one is allowed to do so and there is no data clash.

If either of the microcontrollers is sending information to the serial memory then the memory is a 'slave receiver' and the microcontroller is the 'master transmitter'.

If a microcontroller is receiving information from the serial memory then the memory is a 'slave transmitter' and the microcontroller is the 'master receiver'.

Of course, if after arbitration the losing microcontroller is addressed by the winner, then the loser is considered a 'slave transmitter' or 'slave receiver' and the winner, a 'master transmitter' or 'master receiver'.



	uC 1	uC 2	Memory	Display
Master Tx	0	X	/	/
Slave Rx	X	0	0	0
			X	X
Master Rx	0	X	/	/
Slave Tx	X	0	0 X	/
Slave Rx			0	0
			X	X

X = First instance

0 = Second instance

NOTE: Display can never become a transmitter

Fig. 1 Example of typical serial I/O connection.

2.0 SERIAL INPUT/OUTPUT

The ability of any two devices to communicate, without interruption to any other devices tied to the bus, is an outstanding attribute of the serial I/O system. Communication is achieved by allocating a specific 7-bit address to each device and providing a system whereby a device reacts only to messages prefixed with its own address or the 'general call' address (00 H). Address recognition is performed by the interface hardware so that operation of the microcontroller need only be interrupted when a valid address has been received. This results in a significant saving of processing time and memory space compared with a conventional microcontroller employing a software serial interface. The addressing facilitiy is not required, for instance in a system with only two microcontrollers, direct data transfer without addressing can be performed i.e. the receiving device reacts after each received byte from the serial bus (see free data format Chapter 6).

3.0 SERIAL BUS STRUCTURE

The serial data (SDA) and serial clock (SCL) lines are both bidirectional. Each is connected to a positive supply voltage via a pull-up resistor (see Fig. 2). When the bus is free, both lines are HIGH. The output stages of peripheral IC's connected to the bus must have an open drain or open collector to perform the wired-AND function. The number of ICs that can be connected to the bus, and its length, are solely limited by the maximum bus capacitance of 400 pF.

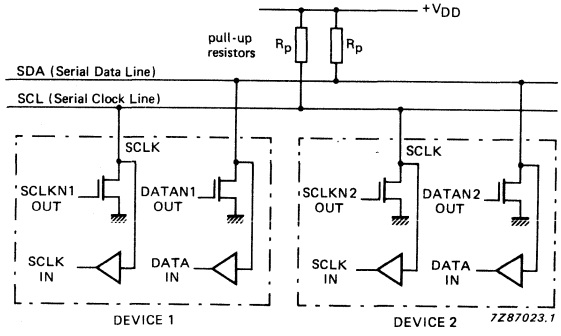


Fig. 2 Connection of two microcontrollers to the serial bus.

R_p = 1 to 5 ko (depending upon current required)

Each microcontroller can be software programmed to function as a 'transmitter' or a 'receiver' operating in either a 'master' or a 'slave' mode.

4.0 SERIAL DATA TRANSFER

Fig. 3 shows the serial data transfer sequence for the SIO interface.
The following conditions can be distinguished:

F (free): The bus is free; data line SDA and clock line SCL are both HIGH

S (start): A data transfer commences with a 'start' condition during which the level of data line SDA changes from HIGH to LOW, and clock line SCL remains HIGH. The bus is now busy.

C (change): During the LOW perion of the clock, the data bit to be transmitted is applied to data line SDA. The level on the SDA line may therefore change during this period

D (data): One data bit is transmitted during the HIGH period of the clock. As SDA line is sampled on an SCL HIGH pulse, the level (bit state) on line SDA must remain stable during this period to prevent it being interpreted as a 'start' or 'stop' condition

P (stop): A data transfer concludes with a 'stop' condition during which the level on data line SDA changes from LOW to HIGH and clock line SCL remains HIGH. The bus is now free again.

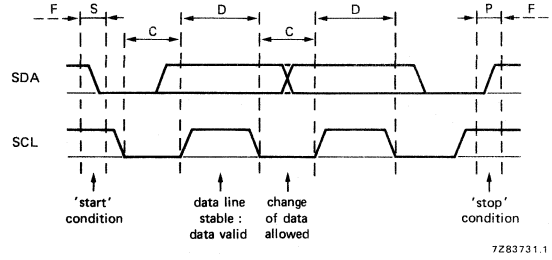


Fig. 3 Sequence of bit transfer on the serial bus.

Every byte transferred on the SDA line must contain 8-bits. The number of bytes that can be transmitted per transfer is unrestricted. Each byte must be followed by an acknowledge bit. If a receiving device cannot receive another complete byte of data until it has performed some other function, for example serviced an internal interrupt, it may hold the clock line SCL LOW to force the transmitter into a wait state. Transfer then continues when the receiver is ready for another byte and releases clock line SCL.

5.0 SERIAL DATA FORMATS

The data transfer format is shown in figure 4. After the start condition, a 7-bit slave address is transmitted which is follwed by a data direction bit (R/W); a '0' indicates a write action, a '1' indicates a read. A data transfer is always terminated by a stop condition generated by the master. However, if a master still wishes to communicate on the bus, it can generate another start condition, and address a further slave without first generating a stop condition. Various combinations of read/write formats are then possible within such a transfer. Figures 5(a), (b), (c) and (d) illustrate the four basic types of transfer formats used:

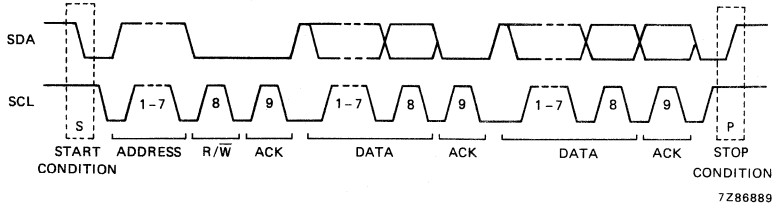


Fig. 4 A complete data transfer.

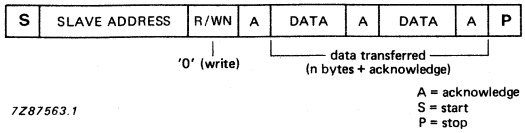


Fig. 5(a) Master transmitter transmits to slave, direction unchanged.

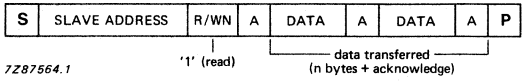


Fig. 5(b) Master reads slave immediatly after first byte.

At the moment of the first acknowledge, the master transmitter becomes a master receiver and the slave receiver becomes a slave transmitter. This acknowledge is still generated by the slave. The stop condition is generated by the master.

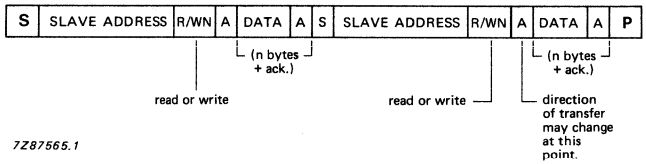


Fig. 5(c) Combined formats.

During a change of direction within a transfer, the start condition and slave address are both repeated, but with the R/W bit reversed (See ALS='0').

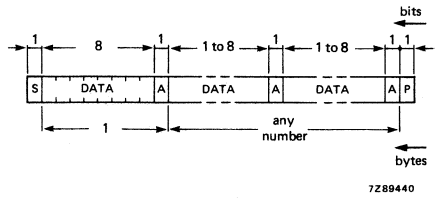


Fig. 5(d) Free data format.

In the free data format, the direction of transmission remains constant throughout the data transfer (See ALS ='1').

In the first three formats (Figs. 5(a), (b) and (c)), byte SLA is the address of the target slave to be selected by the master. The least significant bit of the address byte indicates the direction of transmission of the following byte(s).

If R/W is 0 the master will write (transmit) data to the selected slave; if it is set to 1 the master will read data from the slave. In the last case the direction will change after transmission of the first byte (Fig. 5(c)).

6.0 OPERATING MODES OF THE SERIAL I/O INTERFACE

The serial I/O functions in four basic operating modes to service all facilities of the I^2C (Inter IC) bus, P/C bus or point to point connections:

- 'master transmitter'
- 'master receiver'
- 'slave receiver'
- 'slave transmitter'

6.1 Master transmitter

In this mode, data assembled in one of the previously described data formats is shifted-out on the serial data line synchronous with the clock pulses on SCL. The clock pulses are inhibited and SCL held LOW when the intervention of the processor is required (see address format (a) or (c)).

6.2 Master receiver

This mode can only be entered from the 'master transmitter' mode. Using the address formats ((b) or (c)), the 'master receiver' mode is entered after the transmission of address byte SLA with the R/W bit set to 1. Serial data bits received on bus line SDA by a 'master receiver', are shifted-in synchronized with the self-generated clock pulses on SCL. Again clock pulses are inhibited and SCL held LOW when the intervention of the processor is required after reception of a byte. At the end of a transfer, the 'master receiver' generates the 'stop' condition P.

6.3 Slave receiver

Serial data bits received on bus line SDA by a 'slave receiver', are shifted-in synchronized with the clock pulses at SCL generated by the 'master' device. A 'slave receiver' does not generate clock pulses. The 'slave receiver' holds clock line SCL LOW whilst intervention of the processor is required following the reception of a byte. 'Start' and 'stop' conditions are recognized and interpreted accordingly.

Note; When a slave receiver does not acknowledge its address, for example if it is not able to receive because it is performing some real-time function, the data line (SDA) is left HIGH by the slave. The master can then generate a stop condition to abort transfer.

If a slave receiver acknowledges its slave address, but later in the transfer cannot receive further data, the master must again abort the transfer. This is indicated by the slave not acknowledgeing a data byte by leaving the data line HIGH. The master then generates a stop condition.

6.4 Slave transmitter

The 'slave transmitter' mode can only be entered from the 'slave receiver' mode. Using either of the address formats ((b) or (c)), the 'slave transmitter' mode is entered when the address received in byte SLA matches its own slave address and the R/W bit is a 1. A 'slave transmitter' shifts out the serial data on data line SLA, in synchrony with the clock pulses generated on clock line SCL by the 'master' device. A 'slave transmitter' does not generate clock pulses. The 'slave transmitter' holds clock line SCL LOW if invervention of the processor is required after transmission of a byte. 'Start' and 'stop' conditions are recognized and interpreted accordingly.

7.0 SERIAL I/O INTERFACE

A block diagram of the SIO interface is shown in figure 6. The clock line of the serial bus has exclusive use of pin 3, while the data line shares pin 2 with I/O signal P23 of port 2. Consequently, only three I/O lines are available for Port 2 when the SIO interface is enabled. When the SIO interface is enabled, P23 is disabled as a parallel port line (P23 as SDA open drain only).

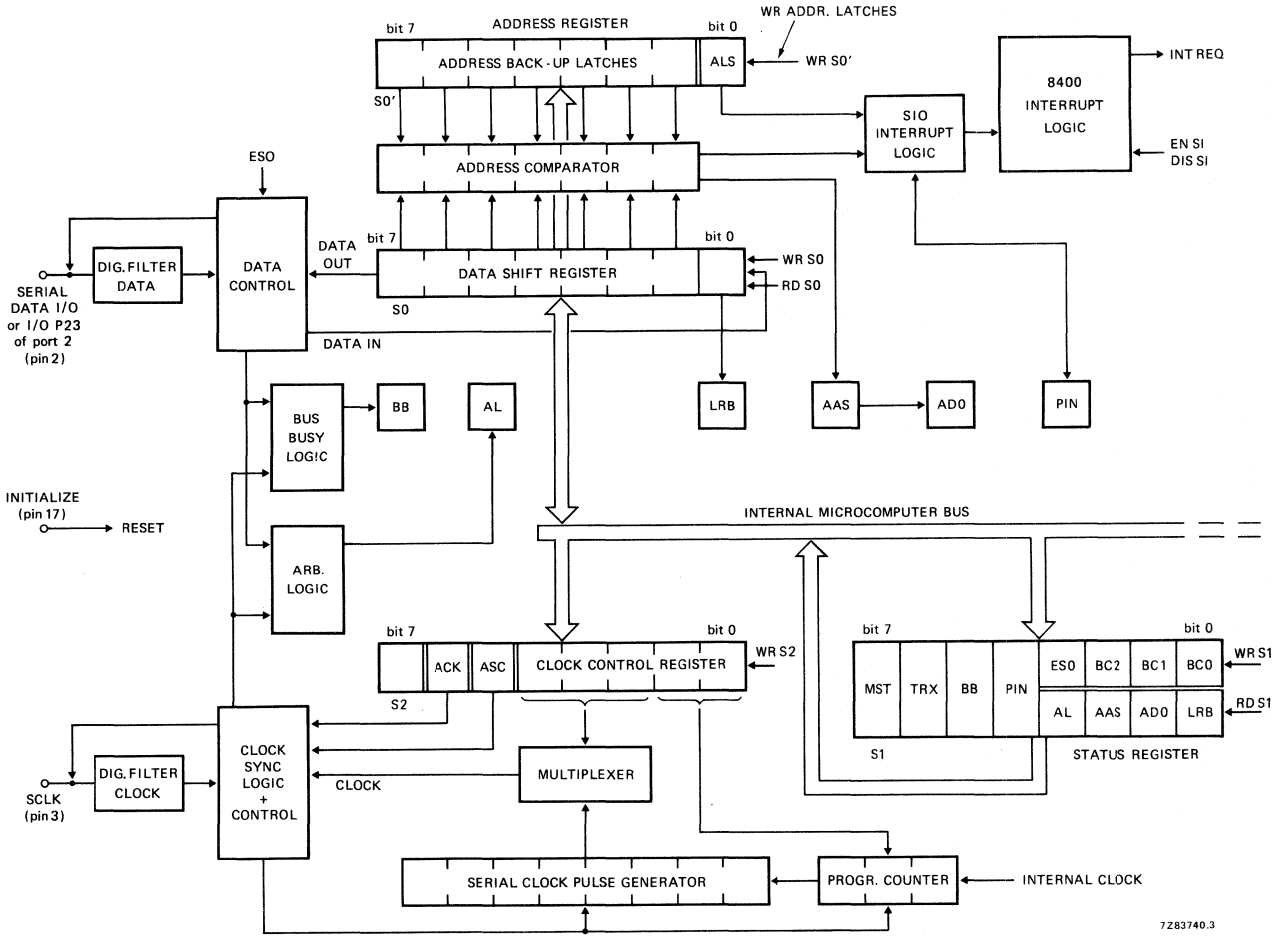


Fig. 6 Block diagram of the MAB84X1 SIO interface.

Communication between the microcontroller and interface takes place via the internal bus of the microcontroller and the Serial Interrupt Request line. Four registers are used to store data and information controlling the operation of the interface.

- data shift register S0
- address register S0'
- status register S1
- clock control register S2.

Eight instructions control operation on these four registers enabling data to be written into or read via the internal microcontroller bus. All bits with the exception of PIN (pending interrupt not) in the status register are cleared to 0 by a RESET. PIN is set to 1 by a RESET. The address comparator shown in figure 5, can instruct the interrupt logic block (if enabled) to generate an SIO interrupt request to the microcontroller.

Communications with the four registers and the function of the address comparator will now be described in more detail.

7.1 Data shift register SO

SO is the data shift register used to perform the conversion between serial and parallel data format. Data to be transmitted is loaded in parallel into SO and shifted out serially, most significant bit first. Data received on the serial bus is shifted into SO, most significant bit first. After transmission orreception of a complete data byte, the specific address or the general address, a pending interrupt is generated to the microcontroller.

To address this 8-bit data shift register, the ESO (enable serial output) bit in status register S1 must be be set to 1. The write instruction MOV SO,A or MOV SO,# data, causes data shift register SO to be parallel-loaded via the internal bus with the data for transmission. During the transmission, the contents of the register are shifted out, most significant bit first, on to data line SDA.

During reception, the serial data is shifted into the data shift register SO. The last received data bit present on data line SDA is shifted into the least-significant bit position in the data shift register. While in the acknowledgement mode, the acknowledgement bit is not shifted into register SO but into the LRB (last received bit) position of S1. The read instruction MOV A,SO causes the contents of register SO to be parallel-loaded into the accumulator via the internal bus.

7.2 Address register SO

The address register contains the 7-bit address back-up latches and the ALS flag, as shown in Fig. 7. The address latches hold the address allocated to the slave device, while the ALS (Always Selected) flag is used to enable/disable the address recognition mode.

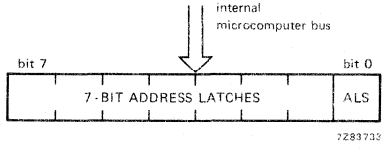


Fig. 7 Address register in the SIO interface.

ALS = 1, The address recognition has been disabled. The SIO interface will respond to the free data formats (Fig 5 (d)), i.e. it reacts after each received byte from the serial bus. This mode may be used when only two devices or memory peripherals share the bus.

ALS = 0 The address recognition has been enabled. The SIO interface will respond to the addressing format i.e. it reacts only to messages containing its own slave address or general address (00 H). In this mode, the least significant bit of the first byte received performs the function of read/write command (write = 0).

The address register can only be loaded from the internal bus of the microcontroller when the SIO interface is disabled (ESO = 0). Under this condition, a MOV SO,A instruction will move the contents of the accumulator to the address register instead of SO, as would normally be the case.

7.3 Status register S1

The status word in status register S1 is shown in figure 8, it contains all information regarding the status of the SIO interface. Read instruction MOV A,S1 causes the status word to be read via the internal bus. To control the SIO interface, information is written into the status register with the write instruction MOV S1,A or MOV S1,# data. Two latches at each of the four least significant bit positions in the status register, enable two separate status bits to be stored at each position. The four least-significant bits ES0, BC2, BC1, and BC0 are write only control bits. The four low order bits AL, AAS and LRB are read only status bits. The four high order bits MST, TRX, BB and PIN which can be both written and read. The function of each bit in the status register will now be described.

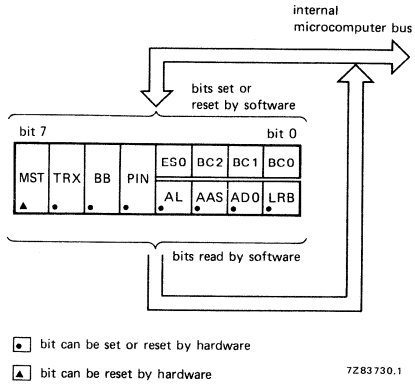


Fig. 8 Bit allocation of the status word in register S1.

MST: 'Master' bit. Setting MST = 1 places the SIO in the 'master' mode and generates the pulses on clock line SCL for timing the transmission or reception of the serial data. Setting MST = 0 places the SIO in the 'slave' mode and the clock pulses are received fram the 'master' device on clock line SCL. The MST bit is reset to 0 by the hardware if an arbitration of this master device is lost. When a data transfer has been completed, a 'stop' condition is generated by the 'master' device and the hardware resets the MST bit of this master device to 0.

TRX: 'Transmitter' bit. Setting TRX = 1 places the SIO interface in the 'transmitter' mode and the data in register S0 is shifted out on data line SDA in synchrony with the pulses on clock line SCL. If TRX = 0, the SIO interface is in the 'receiver' mode and the data on bus line SDA is shifted into data register S0 in synchrony with the pulses on clock line SCL. The TRX bit is reset to 0 by hardware if an arbitration is lost. When a data transfer has been completed, a 'stop' condition is generated by the 'master' device and the hardware resets the TRX bit of this master device to 0. In either of the addressing formats (Fig. 5(a),(b) or (c)), The TRX bit is set by the hardware in accordance with the read/write direction bit R/W. The operating modes set by the MST and TRX bits of the status word are summarized in Table 1.

Table 1 Operating modes set by bits MST and TRX

MST	TRX	operating code
0	0	'slave receiver'
1	0	'master receiver'
0	1	'slave transmitter'
1	1	'master transmitter'

BB: 'bus busy' bit. This bit indicates the state of the serial bus. If it is 0, the bus is free. If it is 1, the bus is occupied. On reception of a 'start' condition, the bus busy logic sets BB to 1. BB is reset to 0 one LOW period of the internal serial clock after reception of a 'stop' condition. In the 'master' mode, BB is controlled by software. To start a transmission with a 'start' condition, bits MST, TRX and BB of the status word are set to 1. To finish a transmission with a 'stop' condition, bit BB is reset to 0 and bits MST and TRX are set to 1.

PIN: 'pending interrupt not' bit. Setting this bit to 0 initiates a serial I/O interrupt only if the EN SI (enable serial interrupt) instruction has been previously executed. As long as PIN is 0, the clock pulses are inhibited and clock line SCL is held LOW. The PIN bit is reset to zero only when:

- A complete byte has been transmitted (even if abitration has been lost).
- The address comparator has recognized its own slave address, or the general address of all 8 zeros has been received (ALS flag = 0).
- Another byte has been received following a previous recognition
- A byte has been received in the free data format shown in Fig. 5 (d).

The PIN bit is reset to 1 by:

- Read instruction MOV A,S0 or write instruction MOV S0,A or MOV S0,# data.
- Loading a 1 into the PIN bit position in status register S1 as a result of write instruction MOV S1,A or MOV S1,# data.

ESO: 'Enable serial output' bit. With ESO = 1, the SIO interface is enabled and can receive or transmit on serial data line SDA, in synchrony with the pulses on clock line SCL. When the ESO bit = 0, the SIO interface is disabled, pin 2 reverts to its P23 (I/O line of port 2) function and pin 3 (SCLK) is in the high impedance state. Whilst ESO is 0, a 7-bit slave address, plus ALS bit, can be loaded into address register S0' with write instruction MOV S0,A or MOV S0,# data. As long as ESO remains 0, PIN is held HIGH, AL is held LOW by hardware and the contents of data shift register S0 are not affected by a write instruction to S0.

BC2, BC1 and BC0: 'bit count'. As shown in Table 2, the binary-coded number preset in bit position BC2, BC1 and BC0 is the number or bits (excluding the acknowledge bit) of the next byte which are yet to be received or transmitted.

Table 2 Binary numbers in bit-count locations BC2, BC1 and BC0

BC2	BC1	BC0	bits/byte without ACK	bits/byte with ACK
0	0	1	1	2
0	1	0	2	3
0	1	1	3	4
1	0	0	4	5
1	0	1	5	6
1	1	0	6	7
1	1	1	7	8
0	0	0	8	9

A 'start' condition resets the bit count to 000, so the first byte to be received or transmitted will always consist of eight bits excluding the acknowledge bit. After each complete byte has been received or transmitted, the bit count has reduced to 000.

AL: 'Arbitration lost' bit. When in the 'master transmitter' mode, this bit is set to 1 when the arbitration logic senses the SIO interface has lost an arbitration. It is also set to 1 if an attempt is made to occupy the bus while BB is set to 1. After a byte has been transmitted, PIN is reset to 0 and the status word in register S1 indicates in which mode the SIO has been addressed. AL is reset to 0 by read instruction MOV A,S0 or any of the write instructions;

```
MOV S0,A
MOV S0,#data
MOV S1,A
MOV S1,#data
```

AAS: 'Addressed as slave' bit. AAS is set to 1 when the address comparator has recognized its own slave address or an address of all (8) zeros. AAS is also set to 1 if the first byte has been received in the free data format (ALS =1). The AAS bit is reset to 0 by the read instruction MOV A,S0 or write instruction MOV S0,A or MOV S0,# data. The device remains selected until the STOP condition.

ADO: 'Address zero' bit. This bit is set to 1 if the address comparator detects the address of all (8) zeros. The ADO bit is reset to 0 when a 'start' or 'stop' condition is detected.

LRB: 'Last received bit' If a byte is transmitted with an acknowledge (ACK) bit, the LRB bit position in status register S1 of the 'transmitter' contains the acknowledgement of the 'receiver'. When LRB is 0, reception of the transmission has been acknowledged. If a transmission does not include an acknowledgement bit, it is the last bit of the transmitted or received byte that will be in the LRB position.

7.4 Clock control register S2

This is an 8-bit write-only register with bit positions as shown in figure 9. The seven bits S20 to S26 can be parallel-loaded with the write instruction MOV S2,A or MOV S2,# data. The eighth bit position (S27) is not used. Bits S20 to S24 control the frequency of the clock pulses for the SIO interface. Bit S25 (ASC) determines whether the mark-space ratio of the clock pulses is 1:1 or 3:1. Bit S26 (ACK) determines whether or not the device is operating in the acknowledgement mode.

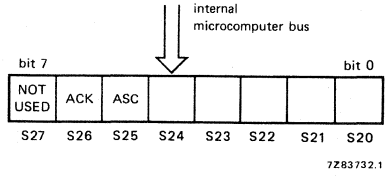


Fig. 9 Bit allocation in clock control register S2

S20 to S24: clock frequency control bits. These bits form a binary-coded frequency control number (0 to 31) which determines the factor by which the microcontroller clock frequency will be devided to obtain the required frequency of pulses on clock line SCL. Table 3 lists the divisors together with hexadecimal equivalents of the frequency control numbers, and shows the resulting frequency of the clock pulses on SCL if the microcontroller clock is controlled by a 4,43 or 6 MHz crystal.

ASC: 'Asymmetrical clock' bit at S25. If ASC = 1 the clock pulses on SCL have a mark-space ratio of 3:1 and the clock frequency control number may must be HEX'18', HEX'19', HEX'1A' or HEX'1B' which gives a clock-pulse frequency range of 1,4 kHz to 2,3 kHz. This is the low-speed mode of the SIO interface. For example, if the clock-frequency control number HEX'19' is in bit positions S20 to S24, the frequency of the pulses on clock line SCL will be about 1,9 kHz (1/f = 520 us for a clock input frequency of 4,4 MHz). Since ASC is 1, the mark/space ratio will be 3:1 so that the HIGH time of the clock pulses is 390 us and the LOW time is 130 us.

If ASC is cleared to 0, the pulses on clock line SCL have a mark-space ratio of 1:1 and any of the clock frequency control numbers, except 0, may be loaded into bit positions S20 to S24. The clock-pulse frequency may therefore be selected within the range 700 Hz to 114 kHz. This is the high-speed mode of the SIO interface.

Table 3 Clock pulse frequency control when using a 4,43 or 6 MHz crystal

HEX S20-S24 code	4,43 MHz crystal		6 MHz crystal
	divisor	approx. f_{clock}(kHz)	approx. f_{clock}(kHz)
0	NOT ALLOWED		NOT ALLOWED
1	39	114	154
2	45	98	133
3	51	87	118
4	63	70	95
5	75	59	80
6	87	51	70
7	99	45	61
8	123	36	49
9	147	30	41
A	171	26	35
B	195	23	31
C	243	18	25
D	291	15	21
E	339	13	18
F	387	11	16
10	483	9,2	12
11	579	7,7	10,4
12	675	6,6	8,9
13	771	5,8	7,8
14	963	4,6	6,2
15	1155	3,8	5,2
16	1347	3,3	4,5
17	1539	2,9	3,9
18*	1923	2,3	3,1
19*	2307	1,9	2,6
1A*	2691	1,7	2,2
1B*	3075	1,4	2,0
1C	3843	1,2	1,6
1D	4611	1,0	1,3
1E	5379	0,8	1,1
1F	6147	0,7	0,98

* Only values that may be used in the low-speed mode (ASC=1)

ACK: 'Acknowledge' bit at S26. For operation in the acknowledgement mode, the ACK bit in register S2 must be set to 1. The acknowledgement procedure is illustrated in figure 10. The operation of devices in each of the four SIO interface modes will now be described.

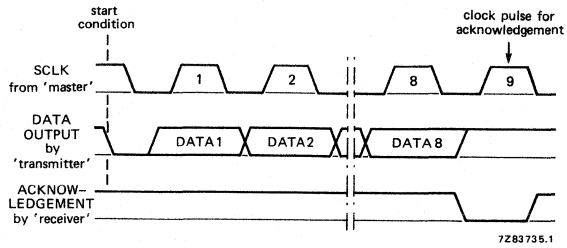


Fig. 10 Acknowledgement between 'receiver' and 'transmitter'.

If the ACK bit in register S2 of a 'master transmitter' is 1, the device generates an extra clock pulse on SCL at the end of each transmitted byte. During this clock pulse, the SERIAL DATA I/O (pin 2) becomes an input and, if a 'receiver' generates an acknowledgement (LOW level on data line SDA), it resets the LRB flag to 0. If acknowledgement is not received, the level on data line SDA remains HIGH setting the LRB flag to 1.

A 'slave transmitter' reacts to an acknowledgement in the same manner as a 'master transmitter' except it receives the extra clock pulse from the 'master' device instead of generating it.

If the ACK bit in clock control register S2 of a 'master receiver' is set to 1, the device generates an extra clock pulse after each received byte and applies it to clock line SCL. During this clock pulse, the SERIAL DATA I/O (pin 2) becomes an output where a LOW level is generated on data line SDA to acknowledge receipt of the transmission.

A 'slave receiver' generates an acknowledgement in the same manner as a 'master receiver' except that it receives the extra clock pulse from the 'master' device instead of generating it.

It should be noted that, if a device changes from a 'receiver' to a 'transmitter' after the R/$\overline{W}$ bit, an acknowledgement will be generated for the first byte by that device. If a device is changed from a 'transmitter' to a 'receiver' after the R/$\overline{W}$ bit, the acknowledgement for the first byte will be received by that device.

If the ACK bit position in clock control register S2 is loaded with a 0, The SIO interface is no longer in the acknowledgement mode. A 'master' device does not then generate an extra clock pulse after each byte and a 'receiver' does not generate an acknowledgement LOW level on bus line SDA.

8.0 SERIAL I/O OPERATIONS

8.1 Arbitration procedure

If two or more master transmitters start a transmission on the same bus almost simultaneously, arbitration procedure will be invoked. The arbitration procedure uses the data presented on the serial data line by the competing transmitters, hence it does not delay transfer of information. When a transmitter senses that a HIGH signal it has presented on the line has been overruled by a LOW signal, it switches to the slave receiver mode, sets the AL flag and generates a pending interrupt at the end of a byte. Figure 11 shows the arbitration procedure between two devices. As soon as the master generating DATA 1 generates an internal HIGH level whilst the data on the SDA line is LOW, it switches its output off so that the master generating DATA 2 wins the arbitration and its data stream continues on the SDA line. Should two or more devices send identical first bytes, arbitration will continue on the subsequent bytes. Since arbitrations are decided solely on the basis of addresses and data transmitted by competing masters, there is no central master, or any order of priority on the bus.

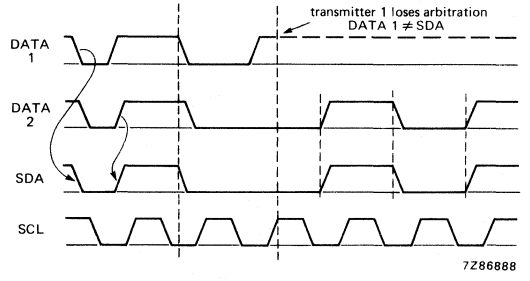


Fig. 11 Arbitration procedure between two master transmitters.

8.2 Clock synchronization

All masters generate their own clock on the SCL line during transfer of data. Data is only valid during the clock HIGH period. A defined clock is therefore needed to allow the bit-by-bit arbitration procedure to take place. Clock synchronization is performed using the wired-AND configuration of the bus. As shown in figure 12, a HIGH to LOW transition on the SCL line causes all competing devices to count off their LOW periods and the SCL line is held low until the device with the longest LOW period finishes its count. Devices with shorter LOW periods remain in a HIGH wait state during this time. When all devices concerned have counted off their LOW periods the SCL line is released and goes HIGH. There is then no difference between the state of the device clocks and the SCL line, and all devices start counting their HIGH periods. The first device to complete its HIGH period drags the SCL line LOW. In this way a synchronized SCL clock is generated, the LOW period of which is determined by the device with the longest LOW period and the HIGH period determined by the device with the shortest clock HIGH period.

If a device pulls down the clock line for a longer time, the result will be that all clock generators enter the WAIT state. In this way a slave can slow down a fast master and the slow device can create enough time to store a received byte, or to prepare a byte to be transmitted. An illustration of clock synchronization is given in figure 12.

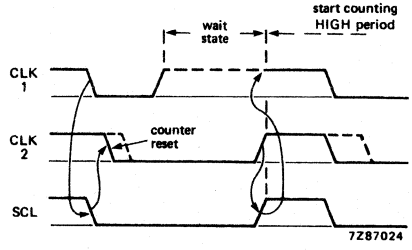


Fig. 12 Synchronization of two SIO clock generators.

9.0 PROGRAMMING

9.1 Machine state following RESET

A reset to pin 17 clears all SIO interface registers (S0, S0', S1 and S2) except for the PIN (pending interrupt not) bit in status register S1 which is set to 1. Because the ESO (enable serial output) bit in status register S1 is 0, the SIO interface is disabled, pin 2 functions as P23 of port 2 (HIGH after RESET) and pin 3 (SCLK) is in the high impedance state. An initialization procedure must therefore be carried out before the SIO interface can be used to transfer serial data.

9.2 Initialization procedure

A flow chart of the initialization procedure is given in Fig. 14. Each step of the procedure will now be explained in more detail.

9.2.1 Clock control register S2

The first step of the initialization procedure is to set the bit pattern in Register S2. Bit position S20 to S24 are loaded, as indicated in Table 3, to set the frequency of the pulses on clock line SCL. Bit position ASC and ACK are loaded as previously described to determine whether the SIO interface operates with a symmetrical or asymetrical clock, and whether or not it operates in the acknowledgement mode. After S2 has been loaded, the clock pulse generator requires one LOW period of its programmed clock cycle-time to stabilize. Since a data transfer must not be started during this stabilization period, the BB flag is set to indicate such an occurence. The BB flag is reset when the clock generator has stabilized.

9.2.2 Address register SO

The next step is to load address register SO'. To address this register, the SIO interface must remain disabled by leaving the ESO (enable serial output) bit in status register S1 at 0. Address register SO' is loaded with the 7-bit slave address plus the ALS bit. If the ALS bit is 1, the SIO will service free data format so the slave address in SO' is immaterial.

9.2.3 Status register S1

The final step of the initialization procedure is to change the ESO bit in status register S1 to 1. All the other bits in register S1 remain as they were following the RESET, i.e. all bits 0 except PIN which is 1. The status register therefore contains HEX'18' and the SIO interface is in the 'slave receiver' mode and ready to receive serial data.

9.2.4 Enable/disable serial I/O interrupt

The instruction EN SI must be performed to enable the serial interrupt logic. If the serial interrupt logic is not enabled, the SIO interface can be serviced by polling PIN.

9.2.5 Example of an initialization procedure

After a RESET and a jump to label INSI, initialization of the SIO interface can be achieved as follows.

```
INSI MOV S2,# H'43'      ; WITH ACK, FSCLK   87 kHz at 4,43 MHz
     MOV SO,# H'84'      ; LOAD SLAVE ADD HEX '42'(bit7-1) IN SO' ALS=0(bit 0)
     MOV S1,# H'18'      ; ENABLE SIO, SELECT SLAVE RECEIVER MODE
     EN SI               ; ENABLE SERIAL I/O INTERRUPT
```

The SIO interface is now in the 'slave receiver' mode. If address HEX'42' is received, the device will generate an acknowledge bit and cause a serial I/O interrupt call. Furthermore software responses depend on the contents of status register S1 as shown in Table 4.

Table 4 Status word bit patterns

Serial I/O status word S1

MST	TRX	BB	PIN	AL	AAS	ADO	LRB	Mode	Data received or transmitted	Software response
0	0	1	0	0	1	1/0	0	SLV/REC	Own slave/all 0´s address received with R/W = 0	A dummy read of S0
0	0	1	0	0	0	1/0	0	SLV/REC	Data received in S0	00101XXX to S1 and/or read S0
0	1	1	0	0	1	0	0	SLV/TRX	Own slave address received with R/W = 1	Write 01101XXX to S1 and/or load S0
0	1	1	0	0	0	0	1/0	SLV/TRX	Transmitted out S0 LRB = 1/0 -> NO ACK/ACK returned	Write 01101XXX to S1 and/or load S0, or write 00111000 to S1
1	1	1	0	0	0	0	1/0	MST/TRX	Transmitted out S0. LRB = 1/0 -> NO ACK/ACK returned	Write 11101XXX to S1 and/or load S0, or * write 11011000 to S1
1	0	1	0	0	0	0	1/0	MST/REC	Slave address with R/W = 1 transmitted LRB = 1/0 -> NO ACK/ACK returned	Write 10111XXX to S1 or a dummy read of S0
1	0	1	0	0	0	0	0	MST/REC	Data received in S0	Write 10101XXX to S1 and/or read S0, or * write 11011000 to S1
0	0	1	0	1	0	0	X	MST/TRX lost arbit.	Slave address or data transmitted and arbitration lost	A dummy read of S0
0	0	1	0	1	1	1/0	0	MST/TRX lost arbit. now SLV/REC	During transmission of slave address arbitration lost and own slave/all 0´s address received with R/W = 0	A dummy read of S0
0	1	1	0	1	1	0	0	MST/TRX lost arbit. now SLV/TRX	During transmission of slave address arbitration lost and own slave address received with R/W = 1	Write 01101XXX to S1 and/or load S0
0	0	0	1	0	0	0	X		The bus is free, a transmission may be started	
X	X	1	X	X	X	X	X		The bus is busy, a transmission may not be started	
X	X	X	1	X	X	X	X		No serial I/O pending i.e. no complete byte received or transmitted (polling PIN)	

Note:

X = 1 or 0. Underlined bits do not contribute to the status information

XXX = BC2, BC1, BC0 according to table 2.

* Generation of a STOP condition.

9.3 Generation of a 'start condition' and the first byte.

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' modes as shown by the flow chart in figure 15. If the device is connected in a multi-transmitter bus system, the state of the BB flag must be tested to verify whether the serial bus is free. If the bus is free (BB = 0), data register S0 is loaded with the first byte for transmission. Bit position MST, TRX, and BB in status register S1 must each be set to 1 so as to begin transmission with a 'start' condition. When the first byte has been transmitted, the PIN flag is reset to 0. A serial interrupt call is initiated if the instruction EN SI has been performed previously. The status word in register S1 determines the software responses that will ensue as listed in Table 4. An example of a program which generates the 'start' condition and transmits the contents of working register Rr is shown below:

```
MSTX MOV A,S1          ; LOAD STATUS WORD TO ACCU
     JB5 MSTX          ; TEST IF BUS IS FREE
     MOV A,Rr          ; LOAD BYTE TO BE TRANSMITTED FROM Rr TO ACCU
     MOV S0,A          ; LOAD BYTE TO BE TRANSMITTED FROM ACCU TO S0
     MOV S1,#H'F8'     ; SELECT MASTER TRANSMITTER MODE AND
                       ; START TRANSMISSION
```

After checking that the serial bus is free, but before the transmission starts, another device may engage the bus. If this occurs, the SIO interface will not start its transmission. To prevent received data from being corrupted, The SIO hardware inhibits data from being written by software into data register S0 and status register S1. If an instruction then attempts to write into register S1, the AL (arbitration lost) flag is set to indicate that the attempt to transmit has failed. When the AL flag is set, a serial interrupt request is always generated at the end of the serial byte.

9.4 Software responses after transmission or reception of a byte

After transmission or reception of a byte, the PIN flag is reset to 0. If a serial I/O interrupt call to program memory location 5 (serial interrupt enabled) is made or if the PIN flag is polled (serial interrupt disabled), a software response has to be initiated to service the SIO interface. As long as the PIN flag is 0, the transfer of serial data is halted because clock bus line SCL is held LOW. Reading out or writing information to data shift register S0 sets the PIN flag to 1 and allows the transfer of serial data to continue.

The status word in register S1 contains all the information to determine the required software response. It also indicates the operating mode of the SIO interface, and gives information about the transmitted or received serial byte. Table 4 lists all possible bit combinations for the status word. It must be remembered that the four least-significant bits of the status word that can be read are not the same as those that can be written (see Fig. 8). The least-significant bit (LRB) of the status word during a read operation only indicates an acknowledgement if the SIO interface is programmed to operate in the acknowledgement mode, i.e. the ACK bit in clock control register S2 is set to 1. If the ACK bit is 0, the LRB bit position in the status register contains the last bit of the byte that has been transmitted or received.

The following is an example of a software response by a 'master transmitter'. The next 8-bit byte to be transmitted is in working register Rq.

```
     ORG H'05'          ; SERIAL INTERRUPT VECTOR
     JMP SIR            ; JUMP TO SERIAL INTERFACE ROUTINE
*
SIR  SEL RB1            ; SELECT REGISTER BANK 1
     MOV Rp,A           ; SAVE ACCU CONTENTS IN Rp
     MOV A,S1           ; LOAD SERIAL STATUS IN ACCU
     JB7 TXTS           ; TEST MST BIT
     JMP SLVR           ; JUMP TO SLAVE ROUTINE IF MST=0
TXTS JB6 TNBY           ; TEST TRX BIT
     JMP MRRC           ; JUMP TO MST/REC ROUTINE IF TRX=0
TNBY MOV A,Rq           ; LOAD NEXT BYTE TO BE TRANSMITTED
*                       ; FROM Rq TO ACCU
     MOV S0,A           ; LOAD IT TO S0=START TRANSMISSION
     MOV A,Rp           ; RESTORE CONTENTS OF ACCU
     RETR               ; RETURN TO MAIN PROGRAM
```

Note that bit counter bits BC0, BC1 and BC2 in the status register need not be preset; they all remain 0 because 8 data bits have to be transmitted (see Table 2). The state of the LRB flag can be tested to check whether the transmitted data has been acknowledged. The result will be interpreted by the programmer.

9.5 Generation of the 'stop ' condition

A data transfer ends with a 'stop' condition generated by the 'master' device. To generate the 'stop' condition, the program in section 9.4 is followed until TXTS. The number of bytes to be transmitted is stored in a RAM memory location or working register. Each time a byte is transmitted this register is decremented. After checking that the last byte has been transferred the program continues with:

```
  MOV S1,#H'D8'         ; RESET BB TO 0, MST, TRX, PIN TO 1
*                       ; THIS GENERATES THE STOP CONDITION
  MOV A,Rp              ; RESTORE CONTENTS OF ACCU
  RETR                  ; RETURN TO MAIN PROGRAM
```

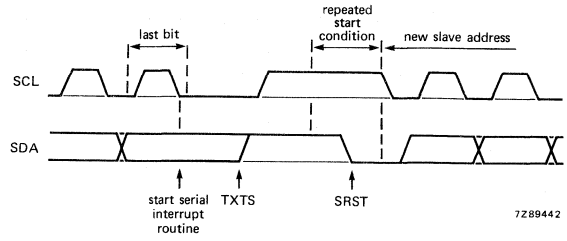


Fig. 13 Repeated 'start' condition on the serial bus.

Figure 13 shows the generation of a repeated 'start' condition followed by a new slave address. The addressing format with repeated 'start' condition is shown in figure 5(c). The serial bus remains busy at the end of the preceding data transfer because a 'stop' condition is not generated. The repeated 'start' condition is generated by a program which follows the program in section 9.4 until TXTS. After a check to verify that a repeated 'start' condition must be transmitted, the program continues with:

```
     MOV S1,#H'18'       ; RESET MST, SLV, BB TO 0
*                        ; SET PIN TO 1
*                        ; NOW BOTH SDA AND SCL
*                        ; BECOME HIGH (NO STOP CONDITION), PROVIDED SCL IS
*                        ; NOT PULLED DOWN BY ANOTHER DEVICE
MSTX MOV A,S1            ; LOAD STATUS WORD TO ACCU **
     JB5 MSTX            ; TEST BB. AS LONG AS THE INTERNAL SERIAL
*                        ; CLOCK IS LOW, BB = 1 **
     CPL A               ; COMPLEMENT ACCU
     JBO MSTX            ; TEST LRB=TEST IF SCL IS HIGH *** LRB IS SET TO 1
*                        ; AT THE POSITIVE GOING EDGE OF SCL (SDA=1)
     MOV A,Rq            ; LOAD NEW SLAVE ADD. TO ACCU
     MOV SO,A            ; LOAD NEW SLAVE ADD. TO SO
SRTS MOV S1,#H'F8'       ; SELECT MASTER TRANSMITTER MODE
*                        ; START TRANSMISSION
     MOV A,Rp            ; RESTORE CONTENTS OF ACCU
     RETR                ; RETURN TO MAIN PROGRAM
```

** This instruction may be omitted if it is certain that the LOW period of the serial clock finishes before label SRTS is reached.

*** As long as a slow slave pulls down the SCL line, the master is not able to generate a repeated 'start condition.

9.7 Generation of the 'stop' condition by a 'Master Receiver'

If a 'master receiver' wants to conclude a data transfer, it must instruct the 'slave transmitter' to free data bus line SDA so that it can generate the 'stop' condition. The 'master receiver' therefore generates a negative acknowledge (data line SDA held HIGH) after reception of the current data byte. The 'slave transmitter' recognizes the negative acknowledge by testing the LRB flag and must then be changed to the 'slave receiver' mode by software. The generation of the 'stop' condition consists of the following three stages:

1. An 8-bit byte is received without generating an acknowledgement bit because bit position ACK in clock control register S2 is loaded with a 0.

2. The 'master receiver' gives a negative acknowledgement by generating an extra clock pulse, i.e. it generates a single HIGH-state bit on data line SDA.

3. The 'stop' condition is generated as a 'master transmitter'.

The 'master receiver' program for the last received byte is the same as that given in section 9.4 until TNBY. After a check has verified that the last byte is to be transferred, the program continues with:

```
     MOV S2,#H'03'       ; NO ACK, FSCL = 87 kHz
     MOV A,SO            ; READ SO AND START
*                        ; WITH LAST BYTE TRANSFER
     MOV @RO,A           ; STORE RECEIVED DATA
     MOV A,Rp            ; RESTORE CONTENTS OF ACCU
     RETR                ; RETURN TO MAIN PROGRAM
```

During the next serial interrupt routine, the program in section 9.4 is again followed until TNBY. After a check has verified that the last byte has been transferred, a negative acknowledge and a 'stop' condition must be transmitted. The program continues with:

```
     MOV S1,#H'A9'       ; LOAD BIT COUNTER FOR
*                        ; ONE BIT TRANSFER
     MOV A,SO            ; READ SO AND START
*                        ; ONE BIT TRANSFER =
*                        ; NEGATIVE ACKNOWLEDGE
     MOV @RO,A           ; STORE RECEIVED DATA
TSPI MOV A,S1            ; LOAD STATUS WORD TO ACCU **
     JB4 TSPI            ; TEST END OF ONE BIT TRANSFER **
     MOV S2,#H'43'       ; RESTORE ACK BIT IN S2
     MOV S1,#H'D8'       ; GENERATE STOP CONDITION
     MOV A,Rp            ; RESTORE ACCU
     RETR                ; RETURN TO MAIN PROGRAM
```

** These instructions can be omitted if it is certain that the negative acknowledge has been transferred before instruction MOV S2,# H'43' is given.

9.8 Flow charts for the 'master' and 'slave' functions of the SIO interface

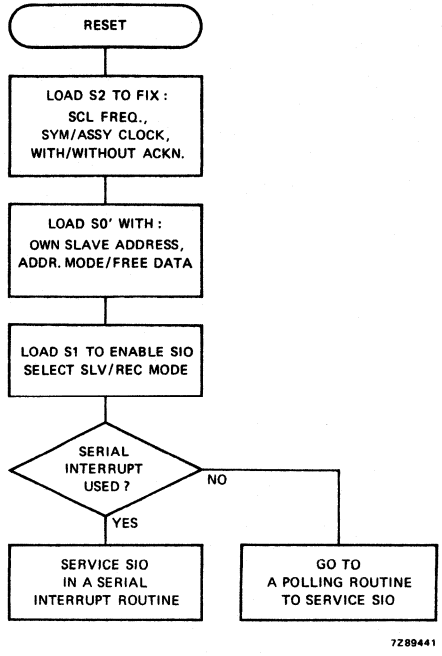


Fig. 14 Initialization flow chart. A prior RESET signal has cleared all bits of the SIO registers except PIN = 1 in status register S1.

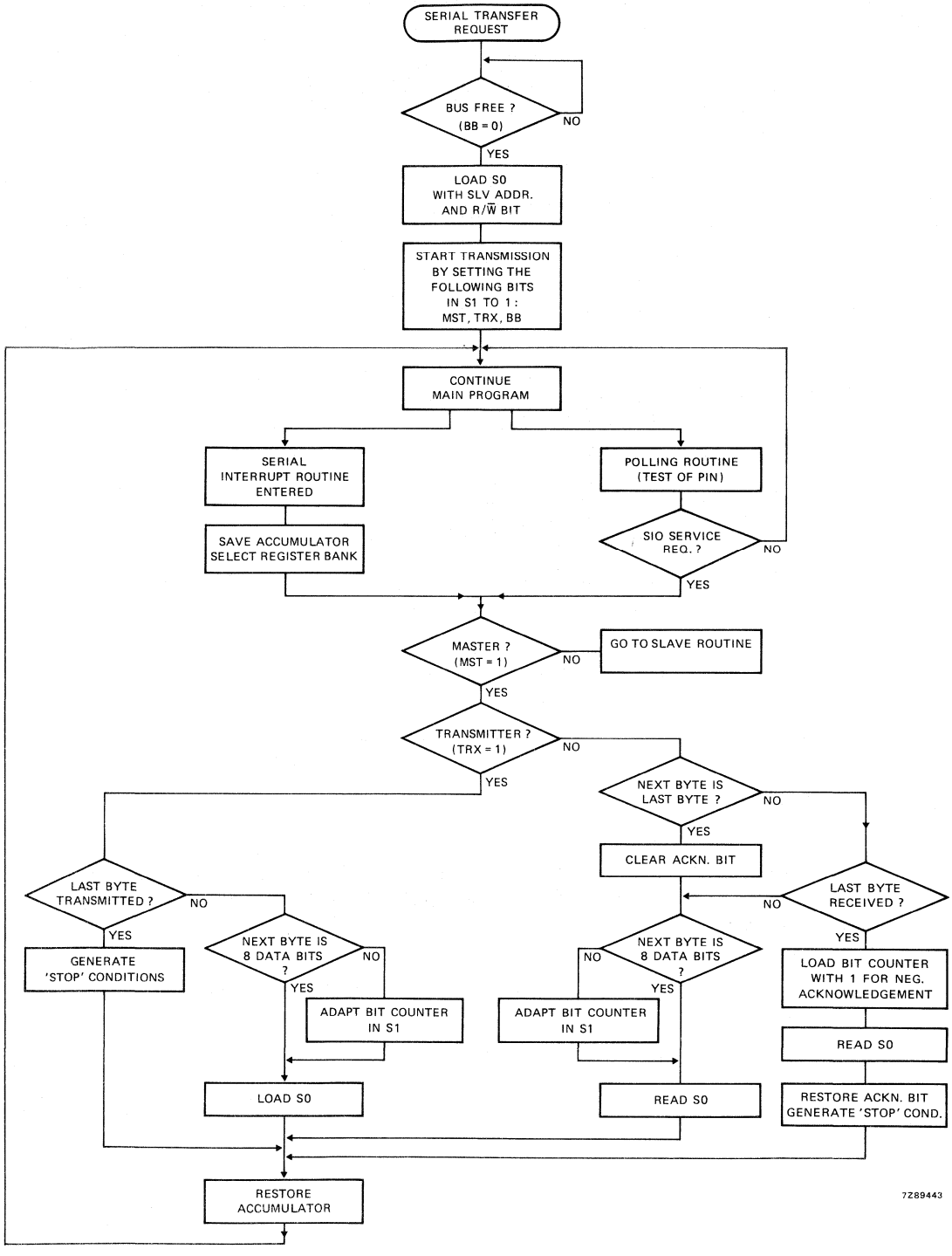


Fig. 15 Flow chart of the 'master' mode after initialization.

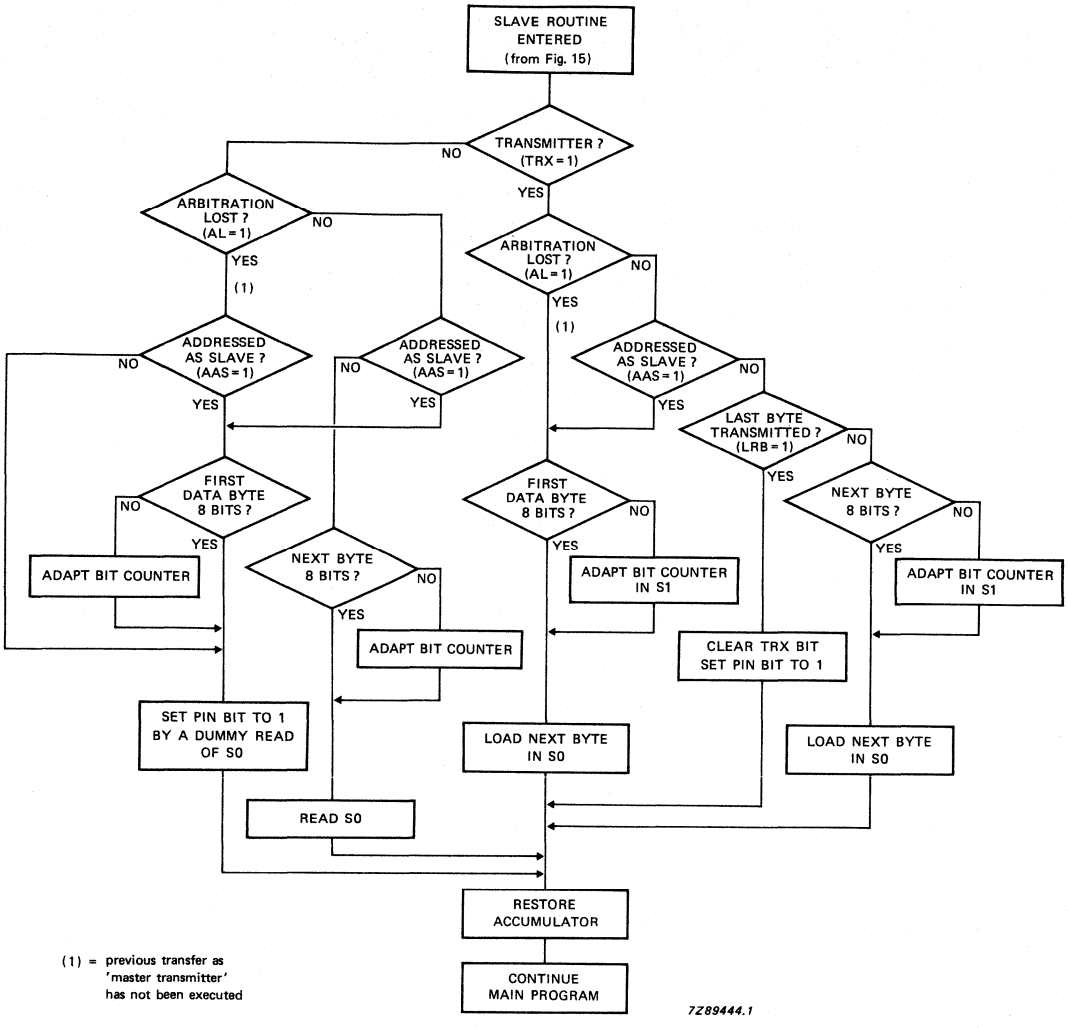


Fig. 16 Flow chart of the 'slave' mode after initialization.

9.9 Solutions to bus-error software problems

The following are solutions to an undefined bus-error state, due to noise or interference, that the SIO interface may suffer while operating in the I^2C mode.

1. Problem:

   In multi-master operation it is feared that the SIO interface has been disturbed during transmission of the acknowledge bit.

   Solution:

   This can be solved by rewriting register S2:

```
MOV S2, #H'02'      ; Clear ACK bit
MOV S2, #H'42'      ; Set ACK bit and frequency
```

2. Problem:

   To recover succesfully from a bus-error state, register S1 must also be reinitialized. When writing to S1, not all the bits are set as expected.

   Solution:

   Write to S1 twice:

```
MOV S1, #ES0 + PIN
MOV S1, #ES0 + PIN
```

While operating in the 'Master transmitter' mode and the end of a byte is reached, the following subroutine illustrates a possible bus-error recovery program incorporating the above instructions.

```
MERROR   MOV R1,A            ; Save accumulator
         MOV A,S1            ; Get status
         JB2 ADDSLV          ; Jump if addressed as slave
         JB7 I2CSST          ; Jump if MST = 1
         JB3 MERR2           ; Jump if AL = 1
         MOV S2,#H'02'       ; Release bus at error condition
                             ; AAS + MST + AL = 0
         MOV S2,#H'42'       ;
         MOV S1,#ES0+PIN     ;
         MOV S1,#ES0+PIN     ;
         JMP MERR1           ;
MERR2    MOV A,S0            ; Release bus method if AL = 1
         JMP MERR1           ;
I2CSST   MOV A,Rq            ; Send next data byte
         MOV S0,A            ;
 MERR1   MOV A,R1            ; Restore accumulator
         RET                 ; Leave
ADDSLV   CALL SLAVE          ; Call slave routine
         MOV A,R1            ; Restore accumulator
         RET                 ; Leave
```

8. The I^2C-Bus specification

CONTENTS – I²C-BUS SPECIFICATION

		page
1.0	INTRODUCTION	8-4
2.0	THE I²C-BUS SPECIFICATION	8-4
3.0	GENERAL CHARACTERISTICS	8-7
4.0	BIT TRANSFER	8-7
4.1	Data validity	8-8
4.2	Start and stop conditions	8-8
5.0	TRANSFERRING DATA	8-9
5.1	Byte format	8-9
5.2	Acknowledge	8-10
6.0	ARBITRATION AND CLOCK GENERATION	8-11
6.1	Synchronization	8-11
6.2	Arbitration	8-12
6.3	Use of clock synchronizing mechanism as a handshake	8-13
7.0	FORMATS	8-13
8.0	ADDRESSING	8-15
8.1	Definition of bits in the first byte	8-15
8.1.1	General call address	8-17
8.1.2	Start byte	8-19
8.1.3	CBUS compatibility	8-20
9.0	ELECTRICAL SPECIFICATIONS OF INPUTS AND OUTPUTS OF I²C DEVICES	8-21
10.0	TIMING	8-24
APPENDIX A		8-26
Maximum and minimum values of the pull-up resistors R_p and the series resistor R_s.		
APPENDIX B		8-28
Note to chapter 6.2		

1.0 INTRODUCTION

For 8-bit applications, such as those requiring single-chip microcontrollers, certain design criteria can be established:

- A complete system usually consists of at least one microcontroller and other peripheral devices - such as memories and I/O expanders.

- The cost of connecting the various devices within the system must be kept to a minimum.

- Such a system usually performs a control function and does not require high-speed data transfer.

- Overall efficiency depends on the devices chosen and the interconnecting bus structure.

To produce a system that satisfies these criteria, a serial bus structure is needed. Although serial buses don't have the throughput capability of parallel buses, they do require less wiring and fewer connecting pins. However, a bus is not merely an interconnecting wire, it embodies all the formats and procedures for communication within the system.

Devices communicating with each other on a serial bus must have some form of protocol which avoids all possibilities of confusion, data loss and blockage of information. Fast devices must be able to communicate with slow devices. The system must not be dependent on the devices connected to it, otherwise modifications or improvements would be impossible. A procedure has also to be resolved to decide which device will be in control of the bus and when. And if different devices with different clock speeds are connected to the bus - the bus clock source has to be defined.

All these criteria are involved in the specification of the I^2C-bus.

2.0 THE I^2C-BUS SPECIFICATION

Any manufacturing process (NMOS, CMOS, I^2L) can be supported by the I^2C-bus. Two wires (SDA - serial data, SCL - serial clock) carry information between the devices connected to the bus. Each device is recognised by a unique address - whether it is a microcontroller, LCD driver, memory or keyboard interface - and can operate as either a transmitter or receiver, depending on the function of the device we're considering. Obviously an LCD driver is only a receiver, while a memory can both receive and transmit data. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers (see table 1). A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

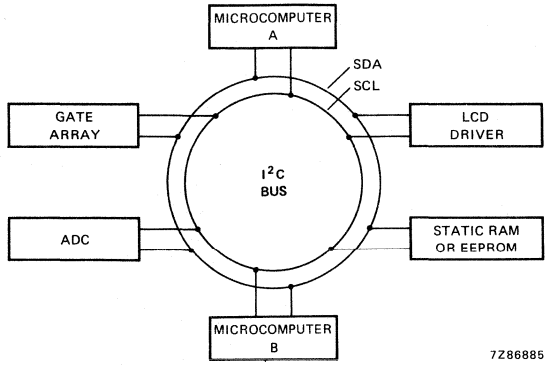


Fig. 1 Typical I^2C-bus configuration.

The I^2C-bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it. As masters are usually microcontrollers, let's consider the case of a data transfer between two microcontrollers connected to the I^2C-bus (Fig. 1). This highlights the master-slave and receiver-transmitter relationships to be found on the I^2C-bus. It should be noted that these relationships are not permanent, but only depend on the direction of data transfer at that time. The transfer of data would follow in this way:

1) Suppose microcontroller A wants to send information to microcontroller B
   - microcontroller A (master) addresses microcontroller B (slave)
   - microcontroller A (master transmitter) sends data to microcontroller B (slave receiver)
   - microcontroller A terminates the transfer.

2) If microcontroller A wants to receive information from microcontroller B
   - microcontroller A (master) addresses microcontroller B (slave)
   - microcontroller A (master receiver) receives data from microcontroller B (slave transmitter)
   - microcontroller A terminates the transfer.

Even in this case, the master (microcomputer A) generates the timing and terminates the transfer.

The possibility of more than one microcontroller being connected to the I^2C-bus means that more than one master could try to initiate a data transfer at the same time. To avoid the chaos that might ensue from such a event - an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all devices to the I^2C-bus.

If two or more masters try to put information on to the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line (for more detailed information concerning arbitration see section 6.0).

Generation of clock signals on the I^2C-bus is always the responsibility of master devices; each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can only be altered when they are stretched by a slow slave device holding down the clock line or by another master when arbitration takes place.

Table 1 Definition of I^2C-bus terminology

Transmitter	- The device which sends data to the bus
Receiver	- The device which receives data from the bus
Master	- The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	- The device addressed by a master
Multi-master	- More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	- Procedure to ensure that if more than one master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted
Synchronization	- Procedure to synchronize the clock signals of two or more devices

3.0 GENERAL CHARACTERISTICS

Both SDA and SCL are bidirectional lines, connected to a positive supply voltage via a pull-up resistor (see Fig. 2). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open drain or open collector in order to perform the wired-AND function. Data on the I^2C-bus can be transferred at a rate up to 100 kbit/s. The number of devices connected to the bus is solely dependent on the limiting bus capacitance of 400 pF.

4.0 BIT TRANSFER

Due to the variety of different technology devices (CMOS, NMOS, I^2L), which can be connected to the I^2C-bus, the levels of the logic '0' (LOW) and '1' (HIGH) are not fixed and depend on the appropriate level of V_{DD} (see Section 9.0 for Electrical specifications). One clock pulse is generated for each data bit transferred.

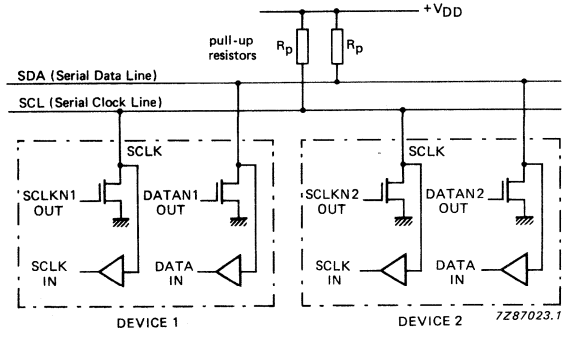


Fig. 2 Connection of devices to the I^2C-bus.

4.1 Data validity

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (Fig. 3).

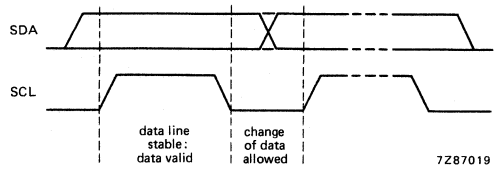


Fig. 3 Bit transfer on the I^2C-bus.

4.2 Start and stop conditions

Within the procedure of the I^2C-bus, unique situations arise which are defined as start and stop conditions (see Fig. 4).

A HIGH-to-LOW transition of the SDA line while SCL is HIGH is one such unique case - this situation indicates a start condition.

A LOW-to-HIGH transition of the SDA line while SCL is HIGH defines a stop condition.

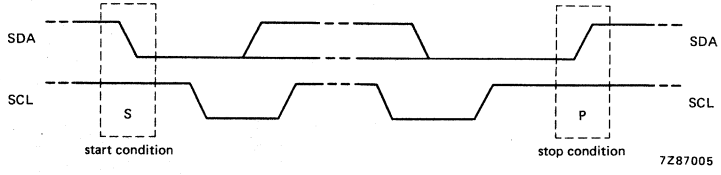


Fig. 4 Start and stop conditions.

Start and stop conditions are always generated by the master. The bus is considered to be busy after the start condition. The bus is considered to be free again a certain time after the stop condition. This bus free situation will be described later in detail.

Detection of start and stop conditions by devices connected to the bus is easy if they possess the necessary interfacing hardware. However, microcontrollers with no such interface have to sample the SDA line at least twice per clock period in order to sense the transition.

5.0 TRANSFERRING DATA

5.1 Byte format

Every byte put on the SDA line must be 8-bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (Fig. 5). If a receiving device cannot receive another complete byte of data until it has performed some other function, for example, to service an internal interrupt, it can hold the clock line SCL LOW to force the transmitter into a wait state. Data transfer then continues when the receiver is ready for another byte of data and releases the clock line SCL.

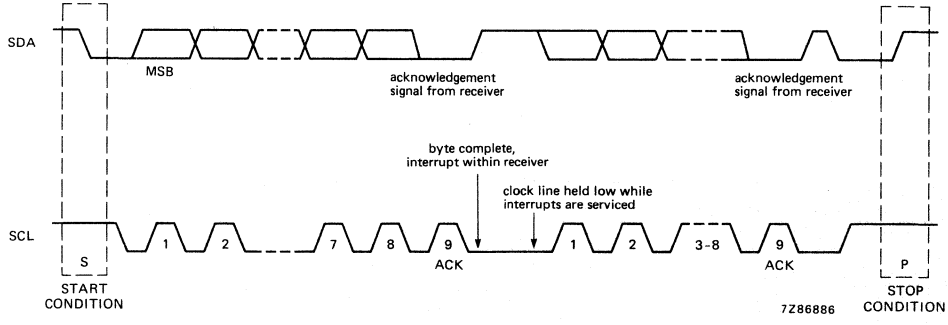


Fig. 5 Data transfer on the I^2C-bus.

In some cases, it is permitted to use a different format from the I^2C-bus format (for example for CBUS compatible devices). A message which starts with such an address can be terminated by the generation of a stop condition, even during the transmission of a byte. In this event, no acknowledge is generated.

5.2 Acknowledge

Data transfer with acknowledge is obligatory. The acknowledge-related clock pulse is generated by the master. The transmitting device releases the SDA line (HIGH) during the acknowledge clock pulse.

The receiving device has to pull down the SDA line during the acknowledge clock pulse so that the SDA line during the acknowledge clock pulse so that the SDA line is stable LOW during the high period of this clock pulse (Fig. 6). Of course, set-up and hold times must also be taken into account and these will be described in section 10.0.

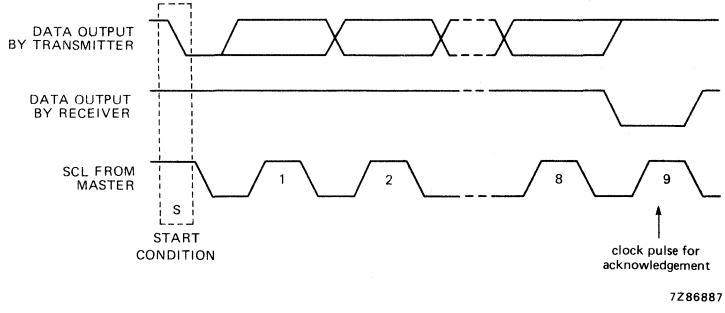


Fig. 6 Acknowledge on the I^2C-bus.

Usually, a receiver which has been addressed is obliged to generate an acknowledge after each byte has been received (except when the message starts with an START BYTE or CBUS address - see section 8.1.2 and 8.1.3)

When a slave receiver does not acknowledge on the slave address, for example because it is not able to receive due to it performing some real-time function, the data line has to be left HIGH by the slave. The master can then generate a STOP condition to abort the transfer.

If a slave receiver does acknowledge the slave address, but some time later in the transfer cannot receive any more data bytes, the master must again abort the transfer. This is indicated by the slave not generating the ackowledge on the first byte following. The slave leaves the data line HIGH during the clock pulse for acknowledgement and the master must generate the STOP condition there after.

In the case of a master receiver involved in a transfer - it must signal an end of data to the slave transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave transmitter must release the data line to allow the master to generate the STOP condition.

6.0 ARBITRATION AND CLOCK GENERATION

6.1 Synchronization

All masters generate their own clock to the SCL line to transfer messages on the I^2C-bus. Data is only valid during the clock HIGH period on the SCL line. A defined clock is needed therefore, if the bit-by-bit arbitration procedure is to take place.

Clock synchronization is performed using the wired-AND connection of devices to the SCL LINE. This means that a HIGH-to-LOW transition on the SCL line will affect the devices concerned, such that they start counting off their LOW period - and once a device clock has gone LOW it will hold the SCL line in that state until the clock HIGH state is reached (Fig. 7). However, the LOW-to-HIGH change in this device clock may not change the state of the SCL line, if another device clock is still within its LOW period. Therefore SCL will be held LOW by the device with the longest LOW period. Devices with shorter LOW periods enter a HIGH wait state during this time.

When all devices concerned have counted off their LOW period, the clock line will be released and go HIGH. There will then be no difference between the device clocks and the state of the SCL line and all of them will start counting their HIGH periods. The first device to complete its HIGH period will again pull the SCL line LOW.

In this way, a synchronized SCL clock is generated for which the LOW period is determined by the device with the longest clock LOW period while the HIGH period on SCL is determined by the device with the shortest clock HIGH period.

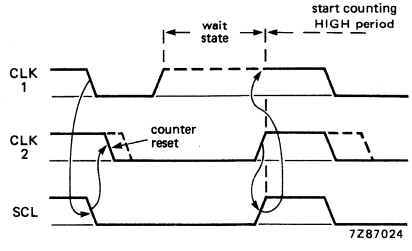


Fig. 7 Clock synchronization during the arbitration procedure.

6.2 Arbitration (See also appendix B)

Arbitration takes place on the SDA line in such a way that the master which transmits a HIGH level, while another master transmits a LOW level, will switch off its DATA output stage since the level on the bus does not correspond to its own level.

Arbitration can carry on through many bits. The first stage of arbitration is the comparison of the address bits (information on addressing can be found in section 8.0). If the masters are each trying to address the same device - arbitration continues into a comparison of the data. Because address and data information is used on the I^2C-bus for the arbitration, no information is lost during this process.

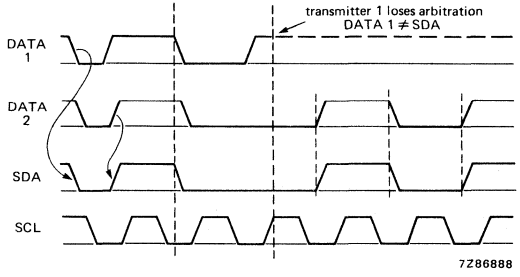


Fig. 8 Arbitration procedure of two masters.

A master which loses the arbitration can generate clock pulses until the end of the byte in which it loses the arbitration.

If a master does lose arbitration during the addressing stage - it is possible that the winning master is trying to address it. The losing master must therefore switch over immediately to its slave receiver mode.

Fig. 8 shows the arbitration procedure for two masters. Of course more may be involved depending upon how many masters are connected to the bus. The moment there is a difference between the internal data level of the master generating DATA 1 and the actual level on the SDA line, its data output is switched off, which means that a HIGH output level is then connected to the bus - this will not affect the data transfer initiated by the winning master. As control of the I^2C-bus is decided solely on the address and data sent by competing masters, there is no central master, nor any order of priority on the bus.

If the arbitration procedure reaches the repeated start condition then the master involved must produce the same format (i.e. they should each generate a repeated start condition in the same position of the format frame see appendix).

6.3 Use of the clock synchronizing mechanism as a handshake

In addition to being used during the arbitration procedure, the clock synchronization mechanism can be used to enable receiving devices to cope with fast data transfers, either on a byte or bit level.

On the byte level, a device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slave devices can then hold the SCL line LOW, after reception and acknowledge of a byte, to force the master into a wait state until the slave is ready for the next byte transfer in a type of handshake procedure.

On the bit level, a device such as a microcontroller without a hardware I^2C interface on-chip can slow down the bus clock by extending each clock LOW period. In this way, the speed of any master is adapted to the internal operating rate of this device.

7.0 FORMATS

Data transfers follow the format shown in Fig. 9. After the start condition, a slave address is sent. This address is 7 bits long, the eighth bit is a data direction bit (R/W) - a 'zero' indicates a transmission (WRITE), a 'one' indicates a request for data (READ). A data transfer is always terminated by a stop condition generated by the master. However, if a master still wishes to communicate on the bus, it can generate another start condition, (see the restirictions for the arbitration procedure in 6.2) and address another slave without first generating a stop condition. Various combinations of read/write formats are then possible within such a transfer.

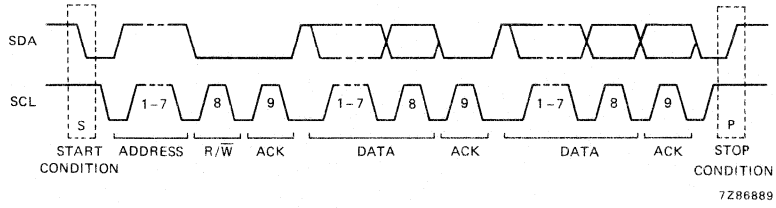
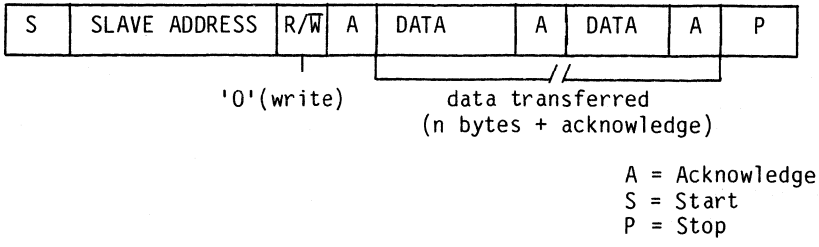


Fig. 9 A complete data transfer.

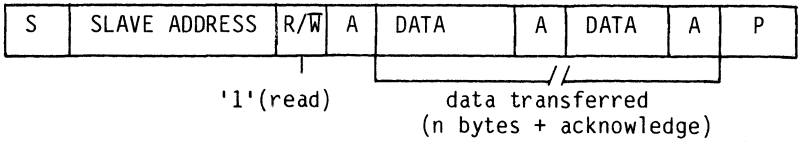
Possible data transfer formats are:

a) Master transmitter transmits to slave receiver. Direction is not changed.



A = Acknowledge
S = Start
P = Stop

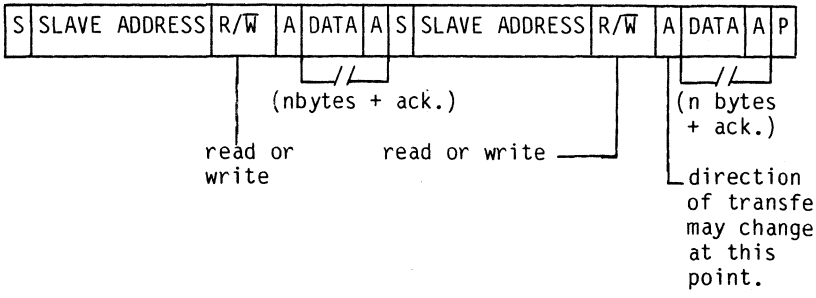
b) Master reads slave immediately after first byte.



At the moment of the first acknowledge, the master transmitter becomes a master receiver and the slave receiver becomes a slave transmitter. This acknowledge is still generated by the slave.

The stop condition is generated by the master.

c) Combined formats.



During a change of direction within a transfer, the start condition and the slave address are both repeated, but with the R/W bit reversed.

NOTE:

1) Combined formats can be used, for example, to control a serial memory. During the first data byte, the internal memory location has to be written. After the start condition is repeated, data can then be transferred.

2) All decisions on auto-increment or decrement of previously accessed memory locations etc. are taken by the designer of the device.

3) Each byte is followed by an acknowledge or a not-acknowledge as indicated by the A and A blocks respectively in the sequence.

4) I^2C devices have to reset their bus logic on receipt of a start condition such that they all anticipate the sending of a slave address.

8.0 ADDRESSING

The addressing procedure for the I^2C-bus is such that the first byte after the start condition determines which slave will be selected by the master. Usually, this first byte follows that start procedure. The exception is the 'general call' address which can address all devices. When this address is used, all devices should, in theory, respond with an acknowledge - although devices can be made to ignore this address. The second byte of the general call address then defines the action to be taken.

8.1 Definition of bits in the first byte

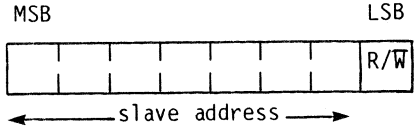


Fig. 10 The first byte after the start procedure.

The first seven bits of this byte make up the slave address (Fig. 10). The eighth bit (LSB - least significant bit) determines the direction of the message. A 'zero' on the least significant position of the first byte means that the master will write information to a selected slave; a 'one' in this position means that the master will read information from the slave.

When an address is sent, each device in a system compares the first 7 bits after the start condition with its own address; if there is a match, the device will consider itself addressed by the master as a slave receiver or slave transmitter, depending on the R/W bit.

The slave address can be made up of a fixed and a programmable part. Since it is expected that identical ICs will be used more than once in a system, the programmable part of the slave address enables the maximum possible number of such devices to be connected to the I^2C-bus. The amount of programmable address bits of a device depends on the amount of pins available. For example, if a device has 4 fixed and 3 programmable address bits, a total of eight identical devices can be connected to the same bus.

The I^2C bus committee is available to coordinate allocation of I^2C addresses.

The bit combination 1111XXX of the slave address is reserved for future extension purposes.

The address 1111111 is reserved as the extension address. This means that the addressing procedure will be continued in the next byte(s). Devices that do not use the extended addressing do not react at the reception of this byte. The seven other possibilities in group 1111 will also only be used for extension purposes but are not yet allocated.

The combination 0000XXX has been defined as a special group. The following addresses have been allocated and are shown in Table 2.

Table 2 Slave address formats

first byte			
SLAVE ADDRESS		R/$\overline{W}$	
0000	000	0	general call address
0000	000	1	start byte
0000	001	X	CBUS address
0000	010	X	Address reserved for different bus format
0000	011	X	to be defined
0000	100	X	
0000	101	X	
0000	110	X	
0000	111	X	

See NOTES

NOTES:

- NO device is allowed to acknowledge at the reception of the start byte.

- The CBUS address has been reserved to enable the inter-mixing of CBUS and I^2C-bus devices in one system. I^2C-bus devices are not allowed to respond at the reception of this address.

- The address reserved for a different bus format is included to enable the mixing of I^2C and other protocols. Only I^2C devices that are able to work with such formats and protocols are allowed to respond to this address.

8.1.1 General call address

The general call address should be used to address every device connected to the I^2C-bus. However, if a device does not need any of the data supplied within the general call structure, it can ignore this address by not acknowledging. If a device does require data from a general call address it will acknowledge this address and behave as a slave receiver. The second and following bytes will be acknowledged by every slave receiver capable of handling this data. A slave which cannot process one of these bytes must ignore it by not acknowledging.

The meaning of the general call address is always specified in the second byte (Fig. 11).

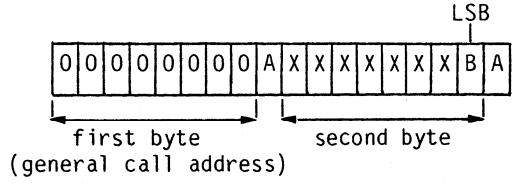


Fig. 11 General call address format.

There are two cases to consider:

- When the least significant bit B is a zero
- When the least significant bit B is a one.

When B is a zero; the second byte has the following definition:

00000110 (H'06') Reset and write programmable part of slave address by software and hardware. On receiving this two byte sequence, all devices (designed to respond to the general call address) will reset and take in the programmable part of their address.

Precautions have to be taken to ensure that a device is not pulling down the SDA or SCL line after applying the supply voltage, since these low levels would block the bus.

00000010 (H'02') Write slave address by software only. All devices which obtain the programmable part of their address by software (and which have been designed to respond to the general call address) will enter a mode in which they can be programmed. The device will not reset.

An example of a data transfer of a programming master is shown in Fig. 12 (ABCD represents the fixed part of the address)

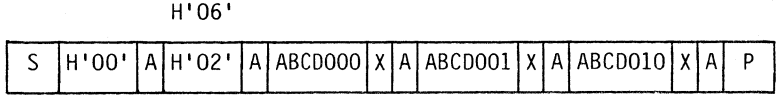


Fig. 12 Sequence of a programming master.

000000100 (H'04') Write slave address by hardware only. All devices which define the programmable part of their address by hardware (and which respond to the general call address) will latch this programmable part at the reception of this two byte sequence. The device will not reset.

00000000 (H'00') This code is not allowed to be used as the second byte.

Sequences of programming procedure are published in the appropriate device data sheets.

The remaining codes have not been fixed and devices must ignore these codes.

When B is a one; the two byte sequence is a 'hardware general call'. This means that the sequence is transmitted by a hardware master device, such as a keyboard scanner, which cannot be programmed to transmit a desired slave address. Since a hardware master does not know in advance to which device the message has to be transferred it can only generate this hardware general call and its own address - identifying itself to the system (Fig. 13).

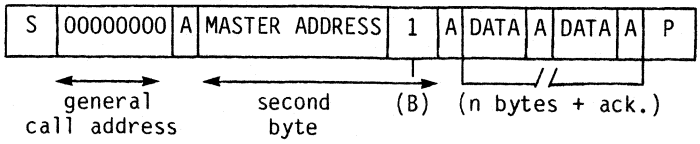


Fig. 13 Data transfer from hardware master transmitter.

The seven bits remaining in the second byte contain the device address of the hardware master. This address is recognized by an intelligent device, such as a microcontroller, connected to the bus which will then direct the information coming from the hardware master. If the hardware master can also act as a slave, the slave address is identical to the master address.

In some systems an alternative could be that the hardware master transmitter is brought in the slave receiver mode after the system reset. In this way, a system configuring master can tell the hardware master transmitter (which is now in slave receiver mode) to which address data must be sent (Fig. 14). After this programming procedure the hardware master remains in the master transmitter mode.

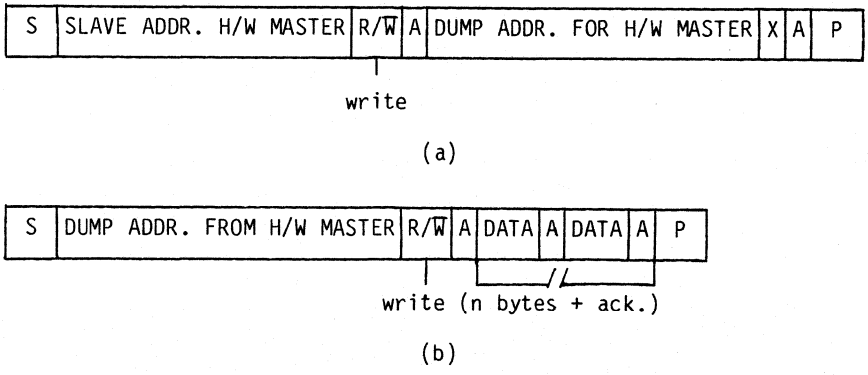


Fig. 14 Data transfer of hardware master transmitter capable of dumping data directly to slave devices, (a) configuring master sends dump address to hardware master, (b) hardware master dumps data to selected slave device.

8.1.2 Start byte

Microcontrollers can be connected to the I^2C-bus in two ways. If an on-chip hardware I^2C-bus interface is present, the microcontroller can be programmed to be only interrupted by requests from the bus. When the device possesses no such interface, it must constantly monitor the bus via software. Obviously, the more times the microcontroller monitors, or polls, the bus - the less time it can spend carrying out its intended function.

Therefore, there is a difference in speed between fast hardware devices and the relatively slow microcontroller which relies on software polling.

In this case, data transfer can be preceded by a start procedure which is much longer than normal (Fig. 15). The start procedure consists of:

a) A start condition S
b) A start byte 00000001
c) An acknowledge clock pulse
d) A repeated start condition Sr

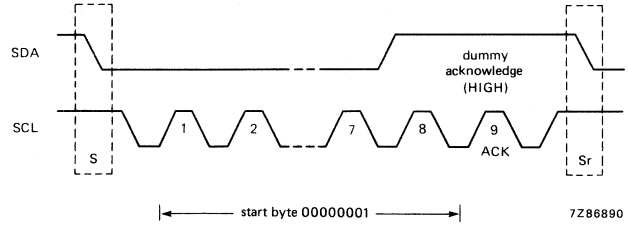


Fig. 15 Start byte procedure.

After the start condition S has been transmitted by a master requiring bus access, the start byte (00000001) is transmitted. Another microcontroller can therefore sample the SDA line on a low sampling rate until one of the seven zeros in the start byte is detected. After detection of this LOW level on the SDA line, the microcontroller is then able to switch to a higher sampling rate in order to find the second start condition Sr which is then used for synchronization.

A hardware receiver will reset at the reception of the second start condition Sr and will therefore ignore the start byte.

After the start byte, an acknowledge-related clock pulse is generated. This is present only to conform with the byte handling format used on the bus. No device is allowed to acknowledge the start byte.

8.1.3 CBUS compatibility

Existing CBUS receivers can be connected to the I^2C-bus. However, in this case, a third line called DLEN has to be connected and the acknowledge bit omitted. Normally, I^2C transmissions are multiples of 8-bit bytes; CBUS devices have different formats however.

In a mixed bus structure, I^2C devices are not allowed to respond on the CBUS message. For this reason, a special CBUS address (0000001X) has been reserved. No I^2C device will respond to this address. After the transmission of the CBUS address, the DLEN line can be made active and transmission, according to the CBUS format, can be performed (Fig. 16).

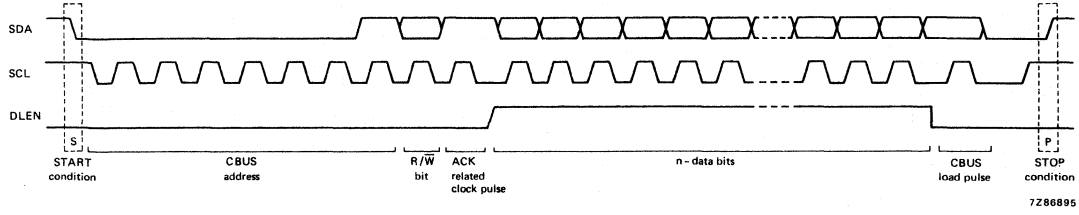


Fig. 16 Data format of transmissions with CBUS receiver/transmitter.

After the stop condition, all devices are again ready to accept data.

Master transmitters are allowed to generate CBUS formats after having sent the CBUS address. Such a transmission is terminated by a stop condition, recognized by all devices.

NOTE: If the CBUS configuration is known and no expansion with CBUS devices is foreseen, the user is allowed to adapt the hold time to the specific requirements of device(s) used.

9.0 ELECTRICAL SPECIFICATIONS OF INPUTS AND OUTPUTS OF I^2C DEVICES

The I^2C-bus allows communication between devices made in different technologies which might also use different supply voltages.

For devices with fixed input levels, operating on a supply voltage of 5 V $\pm$ 10%, the following levels have been defined:

V_{ILmax}= 1,5 V (maximum input LOW voltage)
V_{IHmin}= 3,0 V (minimum input HIGH voltage)

Devices operating on a fixed supply voltage different from 5 V (e.g. I^2L), must also have these input levels of 1,5 V and 3,0 V for V_{IL} and V_{IH} respectively.

For devices operating over a wide range of supply voltages (e.g. CMOS), the following levels have been defined:

$V_{IL\ max}$= 0,3 V_{DD} (maximum input LOW voltage)
$V_{IH\ min}$= 0,7 V_{DD} (minimum input HIGH voltage)

For both groups of devices, the maximum output LOW value has been defined:

V_{OLmax} = 0,4 V (max. output voltage LOW) at 3 mA sink current.

The maximum LOW input current at V_{OLmax} of both the SDA and SCL pins of an I^2C device is -10 μA, including the leakage current of a possible output stage.

The maximum HIGH input current at 0,9 V_{DD} of both the SDA and SCL pins of an I^2C device is 10 μA, including the leakage current of a possible output stage.

The maximum capacitance of both the SDA and SCL pins of an I^2C device is 10 pF.

Devices with fixed input levels can each have their own power supply of 5 V$\pm$ 10 %. Pull-up resistors can be connected to any supply (Fig. 17).

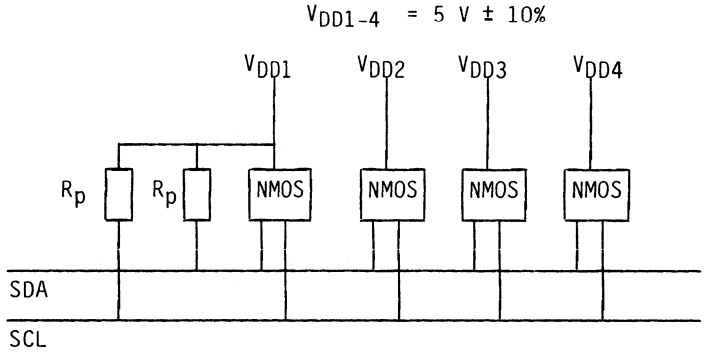


Fig. 17 Fixed input level devices connected to the I^2C bus.

However, the devices with input levels related to V_{DD} must also have one common supply line to which the pull-up resistor is also connected (Fig. 18).

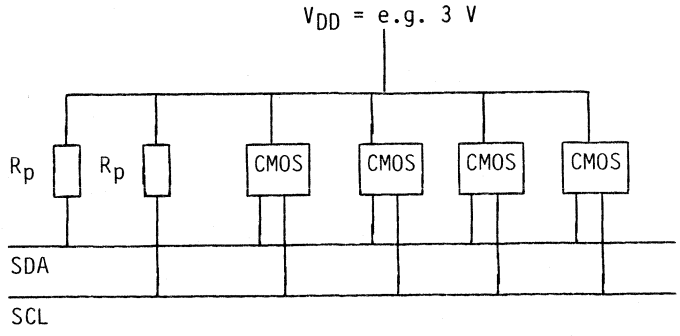


Fig. 18 Devices with a wide range of supply voltages connected to the I^2C-bus.

When devices with fixed input levels are mixed with devices with V_{DD} related levels, the latter devices have to be connected to one common supply line of 5 V ± 10 % along with the pull-up resistors (Fig. 19).

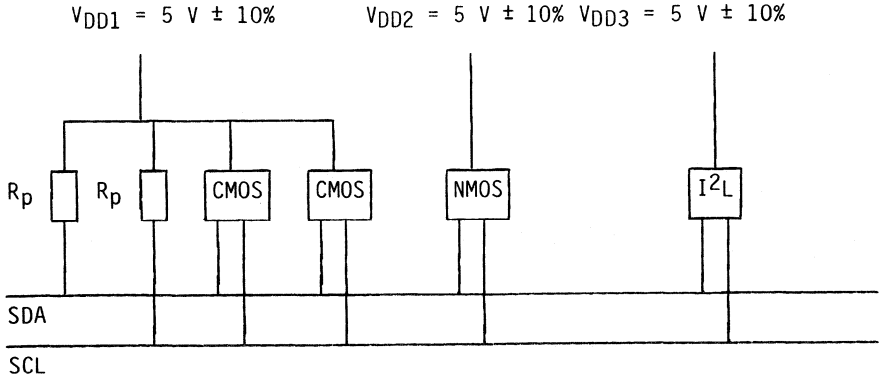


Fig. 19 Devices with V_{DD}-related levels mixed with fixed input level devices on the I^2C-bus.

Input levels are defined in such a way that:

1. The noise margin on the LOW level is 0,1 V_{DD}.
2. The noise margin on the HIGH level is 0,2 V_{DD}.
3. Series resistors (R_S) up to 300Ω can be used for flashover protection of a TV picture tube, for example (Fig. 20).

The maximum bus capacitance per wire is 400 pF. This includes the capacitance of the wire itself and the capacitance of the pins connected to it.

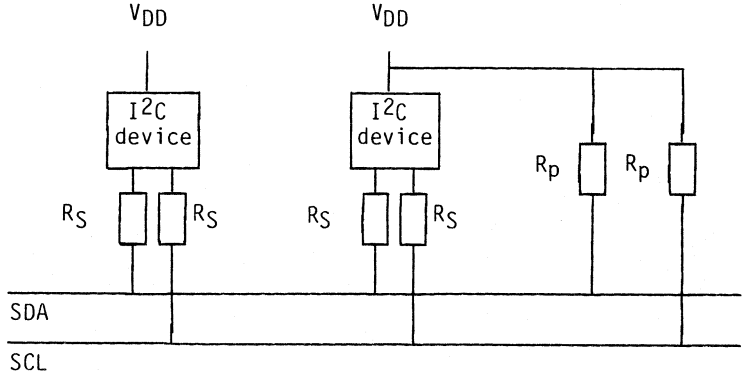


Fig. 20 Serial resistors (R_S) for protection against voltage spikes.

10.0 TIMING

The clock on the I^2C-bus has a minimum LOW period of 4,7 μs and a minimum HIGH period of 4 μs. Devices in the master mode can generate a clock with a frequency up to 100 kHz.

All devices connected to the bus must be able to follow the transfers with frequencies up to 100 kHz, either by being able to transmit or receive at that speed or by applying the clock synchronization procedure which will force the master into the wait state and stretch the LOW periods. Of course, in the latter case the frequency is then reduced.

Fig. 21 shows the timing requirements in detail, a description of the abbreviations used is given in Table 3. All timing references are at $V_{IL\ max}$ and $V_{IH\ min}$.

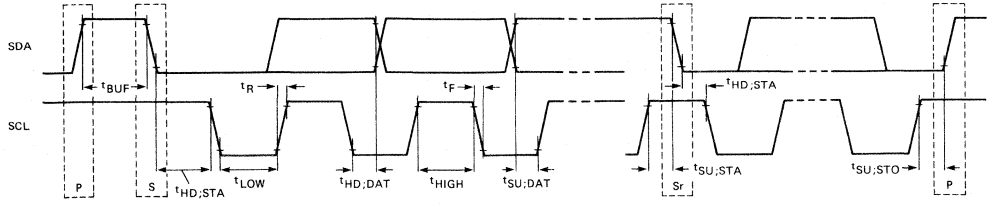


Fig. 21 Timing requirements for the I^2C-bus.

parameter	symbol	min	max	units
SCL clock frequency	f_{SCL}	0	100	kHz
Time the bus must be free before a new transmission can start	t_{BUF}	4,7	-	μs
Hold time START condition. After this period the first clock pulse is generated	$t_{HD;STA}$	4,0	-	μs
The LOW period of the clock	t_{LOW}	4,7	-	μs
The HIGH period of the clock	t_{HIGH}	4,0	-	μs
Set-up time for START condition (only relevant for a repeated start condition).	$t_{SU;STA}$	4,7	-	μs
Hold time DATA for CBUS compatible masters (also see the note in section 8.1.3 for I^2C devics.	$t_{HD;DAT}$	5,0	-	μs
		0*	-	μs
Set-up time DATA	$t_{SU;DAT}$	250	-	ns
Rise time of both SDA and SCL lines.	t_R	-	1	μs
Fall time of both SDA and SCL lines.	t_F		300	ns
Set-up time for STOP condition	$t_{SU;DAT}$	250	-	ns
All values referred to V_{IH} and V_{IL} levels (see section 9.0).				

* Note that internally the transmitter must provide at least a hold time to bridge the undefined region (max 300 ns) of the falling edge of SCL.

APPENDIX A

Maximum and minimum values of the pull-up resistors R_P and series resistors R_S (see Fig. 20)

In an I^2C-bus system these values depend on the following parameters:

- supply voltage
- bus capacitance
- number of devices (input current and leakage current)

1) The supply voltage limits the minimum value of the R_P resistor, due to the specified 3 mA as minimum sink current of the output stages, at 0,4 V as maximum LOW voltage (see Fig. 22).

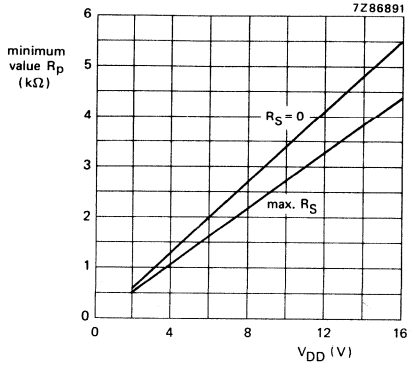


Fig. 22 shows the relationship between V_{DD} and R_P min.

The desired noise margin of 0,1 V_{DD} for the LOW level, limits the maximum value of R_S (see Fig. 23).

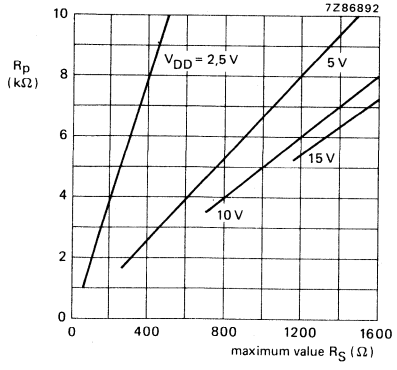


Fig. 23 shows the relationship between $R_{S\ max}$ and R_P.

2) The bus capacitance is the total capacitance of the wire, connections and pins. This capacitance limits the maximum value of R_P due to the specified rise time of 1 us (see Fig. 24).

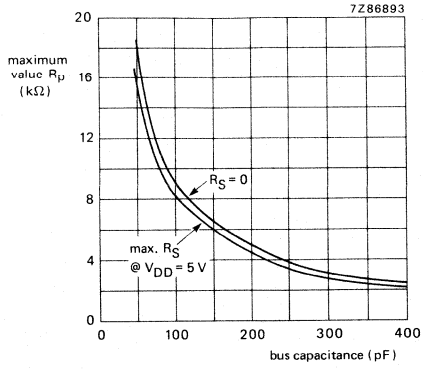


Fig. 24 shows the relationship betwee the bus capacitance and R_P max.

3) The maximum HIGH-level input current of each input/output connection has a specified value of 10 μA (max). Owing to the desired noise margin of 0,2 V_{DD} for the HIGH level, this input current limits the maximum value of R_P. This limit depends on V_{DD} (see Fig. 25).

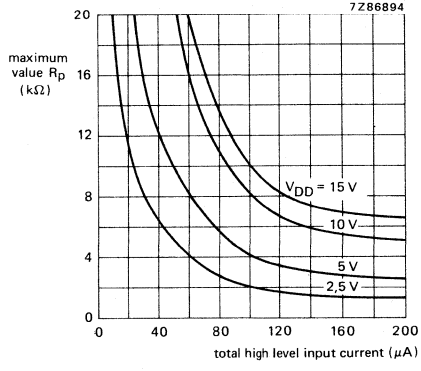


Fig. 25 shows the relationship between the total HIGH level input current and R_P max.

Appendix B.

Note to chapter 6.2

In an I^2C-bus system, the arbitration procedure between two or more masters may continue without decision, until either a repeated START or a STOP condition from either one of the masters is transmitted. It is essential therefore, that each of the masters must contain these conditions in its format at exactly the same location.

In other words, arbitration is not allowed between:

- a repeated START condition and a data bit,
- a STOP condition and a data bit,
- a repeated START condition and a STOP condition.

Explanation: I^2C protocol prevents a repeated START or STOP condition from being masked or disturbed if masters transmit using varied formats. Masking or disturbance could occur, if a master brings SCL low with a data byte, before or during, the generation of another master's repeated START or STOP condition.

9. The MAB8048/MAB84X1/PCF84CXX instruction set

THE GENERAL INSTRUCTION SET

This chapter, the General instruction set, illustrates the opcodes used by the our complete range of 8-bit microcontrollers. Differences exist between the various microcontrollers in both hardware and software, consequently a number of instructions have become particular to each device and these are highlighted appropriately.

The format of the chapter follows the instruction set of the MAB8048 arranged alphabetically. It is suplemented with additional instructions not implemented on the MAB8048 but which are employed in other devices.

Each instruction is introduced by its mnemonic followed by a descriptive title. The hexadecimal opcode is presented together with a byte pattern or binary equivalent. An active description of the each instruction follows and each instruction is furnished with a simple example.

Instructions concerning the PCB8051/C51 are not listed here, but in the separate chapter on the PCB8051/C51.

INSTRUCTION SET

Symbols and Abbreviations Used

A	Accumulator
AC	Auxillary Carry
addr	12-bit Program Memory Address
Bb	Bit Designator (b=0-7)
BUS	BUS Port
C	Carry (CY)
CLK	Clock
CNT	Event Counter
D	Mnemonic for 4-Bit Digit (Nibble)
Dx	Mnemonic derivative register (x = 0 to 255) used as a direct derivative register address
data	8-Bit Number or Expression
F0,F1	Flag 0, Flag 1
H	Data in Hexadecimal
I	Interrupt
MBFF	Memory Bank Flip-Flop(s) (MBFF1=MSB, MBFF0=LSB)
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p = 1, 2 or 4-7)
PSW	Program Status Word
RBS	Register Bank Switch
Rr	Register Designator (r = 0, 1 or 0-7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0,T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X

← Is Replaced by
←→ Is exchanged with

Note: When used in the section (e.g. MAB8048 only)
8048 refers to MAB8048/49/50, PCB80C48/C49 family of microcomputers
84X1 " " MAB84X1/MAB8422/42
84CXX " " PCF84CXX family

*) is used in this section to signify:
bits 0-6 in the 84X1 family incl. 8422/42, 8520, 8540, 8049, 84C20
bits 0-7 in the 84CXX
when referring to register contents.

ADD A,Rr — Add Register Contents to Accumulator

Opcodes 68H to 6FH

0 1 1 0	1 r r r

Working register 'r' contents are added to the accumulator. Carry and half carry are affected.

(A) ← (A)+(Rr) r=0-7

Example:

```
        ADD A,R6        ADD REG 6 CONTENTS
*                       TO ACC
```

ADD A,@Rr — Add Data Memory Contents to Accumulator

Opcodes 60H, 61H

0 1 1 0	0 0 0 r

Resident data memory contents of the location addressed by working register 'r' (bits 0-5*) are added to the accumulator. Carry and half carry are affected.

(A) ← (A)+((Rr)) r=0-1

Example:

```
        MOV R0,#H'1F'   MOVE '1F' HEX TO REG 0
        ADD A,@R0       ADD VALUE OF LOCATION
*                       31 TO ACC
```

ADD A,#data — Add Immediate Data to Accumulator

Opcode 03H

0 0 0 0	0 0 1 1	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. The data specified is added to the accumulator. Carry and half carry are affected.

(A) ← (A)+data

Example:

```
        ADD A,#ADDLE    ADD VALUE OF SYMBOL
*                       'ADDLE' TO ACC
```

ADDC A,Rr — Add Carry and Register Contents to Accumulator

Opcodes 78H to 7FH

0 1 1 1	1 r r r

The carry bit value is added to accumulator location 0 and the carry bit cleared. The contents of working register 'r' are then added to the accumulator. Carry and half carry are affected.

(A) ← (A)+(C)+(Rr) r=0-7

Example:

```
        ADDC A,R5           ADD CARRY AND REG 5
*                           CONTENTS TO ACC
```

ADDC A,@Rr — Add Carry and Data Memory Contents to Accumulator

Opcodes 70H, 71H

0 1 1 1	0 0 0 r

The carry bit value is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by working register 'r' (bits 0-5*) are added to the accumulator. Carry and half carry are affected.

(A) ← (A)+(C)+((Rr)) r=0-1

Example:

```
        MOV R1,#40          MOVE '40' DEC TO REG 1
        ADDC A,@R1          ADD CARRY AND LOCATION 40
*                           CONTENTS TO ACC
```

ADDC A,#data — Add Carry and Immediate Data to Accumulator

Opcode 13H

0 0 0 1	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

A 2-cycle instruction. The carry bit value is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

(A) ← (A)+data+(C)

Example:

```
        ADDC A,#225         ADD CARRY AND '255' DEC
*                           TO ACC
```

ANL A,Rr — Logical AND Accumulator With Register Mask

Opcodes 58H to 5FH

0 1 0 1	1 r r r

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

(A) ← (A) and (Rr) r=0-7

Example

```
        ANL A,R4            'AND' ACC CONTENTS WITH MASK
*                           IN REG 4
```

ANL A,@Rr — Logical AND Accumulator With Memory Mask

Opcodes 50H, 51H

0 1 0 1	0 0 0 r

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0-5*).

(A) ← (A) and ((Rr)) r=0-1

Example:

```
      MOV R0,#H'3F'        MOVE '3F' HEX TO REG 0
      ANL A,@R0            'AND ACC CONTENTS WITH
*                          MASK IN LOCATION 63
```

ANL A,#data — Logical AND Accumulator With Immediate Mask

Opcode 53H

0 1 0 1	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

A 2-cycle instruction. Data in the accumulator is logical ANDed with an immediately-specified mask.

(A) ← (A) AND data

Examples:

```
      ANL A,#H'AF'         'AND' ACC CONTENTS WITH
*                          MASK IN 10101111
      ANL A,#3+X/Y         'AND' ACC CONTENTS WITH
*                          VALUE OF EXPRESSION
*                          '3+X/Y'
```

ANL BUS,#data — Logical AND BUS With Immediate Mask (8048 only)

Opcode 98H

1 0 0 1	1 0 0 0	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

A 2-cycle instruction. BUS port data is logical ANDed with immediately-specified mask, assuming prior specification of an 'OUTL BUS, A' instruction.

(BUS) ← (BUS) AND data

Example:

```
      ANL BUS,#MASK        'AND' BUS CONTENTS
*                          WITH MASK EQUAL VALUE
*                          OF SYMBOL 'MASK'
```

ANL Dx,A — Logical AND derivative register with accumulator (84CXX family only)

Opcode 8EH

1 0 0 0	1 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

A 2-byte, 2-cycle instruction. The data in the derivative register addressed by the second byte, is logically ANDed with the accumulator contents.

(Dx) ← (Dx) AND (A)

Example:

```
ANL D7,A                   'AND' contents of derivative
                           register D7 with Acc.
```

ANL Pp,#data — Logical AND Port 1-2 With Immediate Mask

Opcodes 98H to 9AH

1 0 0 1	1 0 p p	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. Port 'p' data is logically ANDed with an immediately-specified mask.

(Pp)◄— (Pp) AND data p=1-2 (p=0-2 for 84X1, 84CXX)

Example:

```
        ANL P1,#H'F0'       'AND' PORT 1 CONTENTS WITH
*                           MASK 'F0' HEX
*                           (CLEAR P10-13)
```

ANLD Pp,A — Logical AND Port 4-7 With Accumulator Mask (only in 8048)

Opcodes 9CH to 9FH

1 0 0 1	1 1 p p

A 2-cycle instruction. Port 'p' data is logically ANDed with the digit mask of accumulator bits 0-3.

(Pp)◄— (Pp) AND (A0-3) p=4-7

Note: The mapping of Port 'p' to opcode bits 0-1 is as follows:

	Bit 1 0	Port
(9CH)	0 0	4
(9DH)	0 1	5
(9EH)	1 0	6
(9FH)	1 1	7

Example:

```
        ANLD P4,A           'AND' PORT 4 CONTENTS WITH
*                           ACC BITS 0-3
```

CALL address — Subroutine Call

Opcode 14H, 34H, 54H, 74H, 94H, B4H, D4H and F4H

a_{10} a_9 a_8	1 0 1 0 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack (PSW bits 4,6 and 7 are saved in 84X1, 84CXX). The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 (and bit 12 for 84X1, 84CXX) is determined by the most recent SEL MB instruction.

Execution continues at the instruction following the CALL upon return from the subroutine.

8048	84X1, 84CXX
$((SP)) \leftarrow (PC), (PSW_{4-7})$ $(SP) \leftarrow (SP) + 1$	$((SP)) \leftarrow (PC_{0-7})$ $((SP) + 1) \leftarrow (PC_{8-12})$, PSW 4, 6 and 7
$(PC_{8-10}) \leftarrow (addr_{8-10})$ $(PC_{0-7}) \leftarrow addr_{0-7}$ $(PC_{11}) \leftarrow MBFF_0$	$(SP \leftarrow (SP)+1$ $(PC_{8-10}) \leftarrow addr_{8-10}$ $(PC_{0-7}) \leftarrow addr_{0-7}$ $(PC_{11-12}) \leftarrow MBFF_{0-1}$

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
       MOV R0,#50          MOVE '50' DEC TO ADDRESS
*                          REG 0
       MOV A,R1            MOVE CONTENTS OF REG 1
*                          TO ACC
       ADD A,R2            ADD REG 2 TO ACC
       CALL SUBTOT         CALL SUBROUTINE 'SUBTOT'
       ADDC A,R3           ADD REG 3 TO ACC
       ADDC A,R4           ADD REG 4 TO ACC
       CALL SUBTOT         CALL SUBROUTINE 'SUBTOT'
       ADDC A,R5           ADD REG 5 TO ACC
       ADDC A,R6           ADD REG 6 TO ACC
       CALL SUBTOT         CALL SUBROUTINE 'SUBTOT'

SUBTOT MOV @R0,A           MOVE CONTENTS OF ACC TO
*                          LOCATION ADDRESSED BY
*                          REG 0
       INC R0              INCREMENT REG 0
       RET                 RETURN TO MAIN PROGRAM
```

CLR A — Clear Accumulator

Opcode 27H

0 0 1 0	0 1 1 1

The accumulator contents are cleared to zero.

$(A) \leftarrow 0$

CLR C — Clear Carry Bit

Opcode 97H

1 0 0 1	0 1 1 1

The carry bit can be set to one by the ADD, ADDC, RLC, CPL and DAA instructions during normal program execution. This instruction resets the carry bit to zero.

$(C) \leftarrow 0$

CLR F1 — Clear Flag 1 (8048 only)

Opcode A5H

1 0 1 0	0 1 0 1

Flag 1 cleared to zero.

(F1) ← 0

CLR F0 — Clear Flag 0 (8048 only)

Opcode 85H

1 0 0 0	0 1 0 1

Flag 0 cleared to zero.

(F0) ← 0

CPL A — Complement Accumulator

Opcode 37H

0 0 1 1	0 1 1 1

The accumulator contents are complemented, strictly a one's complement. Each zero is changed to a one and vice-versa.

(A) ← NOT (A)

Example: Assume accumulator contains 10010101.

```
        CPL A                   ACC CONTENTS ARE
*                               COMPLEMENTED to 01101010
```

CPL C — Complement Carry Bit

Opcode A7H

1 0 1 0	0 1 1 1

The carry bit setting is complemented, a zero is changed to one, and vice-versa.

(C) ← NOT (C)

Example: Set C to one - current setting is unknown.

```
        CLR C                   C IS CLEARED TO ZERO
        CPL C                   C IS SET TO ONE
```

CPL F0 — Complement Flag 0 (8048 only)

Opcode 95H

1 0 0 1	0 1 0 1

Flag 0 setting is complemented; zero is changed to one and vise-versa.

(F0) ← NOT (F0)

CPL F1 — Complement Flag 0 (8048 only)

Opcode B5H

1 0 1 1	0 1 0 1

Flag 1 setting is complemented; zero is changed to one and one is changed to zero.

(F1) ← NOT (F1)

DA A — Decimal Adjust Accumulator

Opcode 57H

0 1 0 1	0 1 1 1

The value of the 8-bit accumulator is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

Example: Assume accumulator contains 10011011.

```
        DA A                    ACC ADJUSTED TO 00000001
*                               WITH C SET

C AC    7 6 5 4  3 2 1 0
-------------------------
0 0     1 0 0 1  1 0 1 1
                 0 1 1 0        ADD SIX TO BITS 0-7
0 0     1 0 1 0  0 0 0 1
        0 1 1 0                 ADD SIX TO BITS 4-7
1 0     0 0 0 0  0 0 0 1        OVERFLOW TO C
```

DEC A — Decrement Accumulator

Opcode 07H

0 0 0 0	0 1 1 1

Accumulator contents are decremented by one.

(A) ← (A)-1

Example: Decrement contents of external data memory location 63.

```
        MOV RO,#H'3F'          MOVE '3F' HEX TO REG 0
        MOV A,@RO              MOVE CONTENTS OF LOCATION
*                              63 TO ACC
        DEC A                  DECREMENT ACC
        MOV @RO,A              MOVE CONTENTS OF ACC TO
*                              LOCATION 63 IN INTERNAL
*                              MEMORY
```

DEC @Rr — Decrement Data Memory Location (8422/42, 84X1, 84CXX)

Opcodes C0H to C1H

1 1 0 0	0 0 0 r

The contents of the resident data memory addressed by the working register 'r' (bits 0-5*) are decremented by one.

((R)) ← ((Rr))-1

Example:

```
        MOV R1,#H'4F'          MOVE '4F' HEX TO REG 1
        DEC @R1                DECREMENT LOCATION 4F
```

DEC Rr — Decrement Register

Opcodes C8H to CFH

1 1 0 0	1 r r r

The contents of working register 'r' are decremented by one.

(Rr) ← (Rr)-1 r=0-7

Example

```
        DEC R1                 DECREMENT CONTENTS OF
*                              REG 1
```

DIS I

Disable External Interrupt

Opcode 15H

0 0 0 1	0 1 0 1

External interrupts are disabled. A LOW signal on the interrupt input would have no effect.

DIS SI

Disable Serial Input/Output Interrupt (Not in 8048 or 84C12)

Opcode 95H

1 0 0 1	0 1 0 1

Serial input-output interrupts are disabled. An interrupt request from the SIO has no effect. If the processor enters the IDLE mode (84CXX), when the SIO interrupt is disabled, it cannot be restarted by this interrupt.

(SIFF) ← 0

DIS TCNTI

Disable Timer/Counter Interrupt

Opcode 35H

0 0 1 1	0 1 0 1

Timer/counter interrupts are disabled. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ Rr,address Decrement Register and Test

Opcodes E8H to EFH

1 1 1 0	1 r r r	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Register 'r' is decremented and tested for zero. If the register contains all zeros, program control passes to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

(Rr) ← (Rr)-1 r=0-7
If Rr not 0
(PC_{0-7}) ← addr
else (PC) ← (PC) + 2

Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

Example: Increment values in data memory locations 40-54.

```
       MOV R0,#50          MOVE '50' DEC TO ADDRESS
*                          REG 0
       MOV R3,#5           MOVE '5' DEC TO COUNTER
*                          REG 3
INCRT  INC @R0             INCREMENT CONTENTS OF
*                          LOCATION ADDRESSED BY
*                          REG 0
       INC R0              INCREMENT ADDRESS IN REG 0
       DJNZ R3,INCRT       INCREMENT REG 3 - JUMP TO
*                          'INCRT' IF REG 3 NONZERO
       NEXT  - - -         'NEXT' ROUTINE EXECUTED
*                          IF R3 IS ZERO
```

DJNZ @Rr,address Decrement Data Memory location and Test (not in 8048)

Opcode EOH, E1H

1 1 1 0	0 0 0 r	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. The memory location addressed by the working register 'r' bits 0-5*) is decremented and tested for zero. If the memory location contains all zeros, program control passes to the next instruction.

If the register contents are not zero, control jumps to the specified 'address'. The address, in this case, the jump must be to a location within the current 256-location page.

```
((Rr))←((Rr))-1              r=0-1
If ((Rr))≠0
(PC0-7)←addr
else
(PC)←(PC)+2
```

Example: Increment values in data memory locations 81-86.

```
        MOV RO,#32          MOVE '32' DEC TO ADDRESS
*                           REG 0
        MOV @RO,#6          MOVE '6' DEC TO COUNTER
*                           REG 32
        MOV R1,#81          MOVE '81' DEC TO ADDRESS
*                           REG 1
INCRT   INC @R1             INCREMENT CONTENTS OF MEMORY
*                           LOCATION ADDRESSED BY
*                           REG 1
        INC R1              INCREMENT ADDRESS IN REG 1
        DJNZ @RO,DECRT      DECREMENT LOC.32 - JUMP TO
*                           'DECRT' IF LOCATION 32
*                           NONZERO
NEXT    ----                'NEXT' ROUTINE EXECUTED
*                           IF LOCATION 32 IS ZERO
```

EN I — Enable External Interrupt

Opcode 05H

0 0 0 0	0 1 0 1

External interrupts are enabled. A LOW signal (8048) or a HIGH-to-LOW transition (84X1, 84CXX) on the interrupt input pin initiates the interrupt sequence. For the 84CXX, a HIGH-to-LOW transition on the external interrupt pin initiates the interrupt sequence in normal mode or in IDLE mode. In the STOP mode of the 84CXX a LOW level signal rather than a transition initiates the interrupt sequence.

EN SI — Enable Serial Input/Output Interrupt (84CXX, 84X1 only)

Opcode 85H

1 0 0 0	0 1 0 1

Serial input-output interrupts are enabled. An interrupt request from serial I/O initiates the interrupt sequence, in normal mode or in IDLE mode.

(SIFF) ← 1

EN TCNTI — Enable Timer/Counter Interrupt

Opcode 25H

0 0 1 0	0 1 0 1

Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

ENTO CLK — Enable Clock Output (8048 only)

Opcode 75H

0 1 1 1	0 1 0 1

The T0 pin is enabled to act as the clock output. This function is disabled by a system reset.

Example:

```
ENTO CLK                ENABLE TO AS CLOCK OUTPUT
```

IDLE — Select Idle operation (80C49 only)

Opcode 01H

0 0 0 0	0 0 0 1

A 2-cycle instruction. A low-power mode which keeps the oscillator, the internal timer, the external interrupt and counter pins functioning and maintains the register and RAM status. To terminate IDLE mode, a system reset must be performed or interrupts must be enabled and an interrupt signal generated.

IDLE

Select Idle Operation (84CXX only)

Opcode 01H

0 0 0 0	0 0 0 1

The processor enters the IDLE mode. The oscillator, timer/counter and serial I/O are kept running. The processor leaves the IDLE mode by one of the three possible interrupts, if they are enabled, or by activating a RESET. An active signal on the RESET pin always restarts the processor and a normal RESET sequence is executed.

An active signal coming from an interrupt source (if enabled) causes the execution of the normal interrupt routine since normal interrupt scanning is still being carried out. A HIGH-to-LOW transition on the external interrupt pin reactivates the processor, not just a LOW level as was the case in the STOP mode. Therefore if the INT/T0 pin was LOW before the processor entered the IDLE mode, it must go HIGH before it can reactivate the processor.

IN A,Pp

Input Port or Data to Accumulator

Opcodes 08H to 0AH (for 84X1, 84CXX family)
08H, 09H (for 84C12)
09H, 0AH (for 8048)

0 0 0 0	1 0 p p

A 2-cycle instruction. Data present on Port 'p' is transferred (read) to the accumulator.
In the 84CXX IN A,P2 inputs P20-P23 to A0-A3 while A4-A7 is set to zero. In the 84X1, IN A,P2 transfers data on P20-P23 to A0-A3 while A4-A7 are set to ones.

(A) ← (Pp) p=1-2 (p=0-2 for 84X1, 84CXX)

Example:

```
        IN A,P1             INPUT PORT 1 CONTENTS
*                           TO ACC
        MOV R6,A            MOVE ACC CONTENTS TO
*                           REG 6
        IN A,P2             INPUT PORT 2 CONTENTS
*                           TO ACC
        MOV R7,A            MOVE ACC CONTENTS TO REG 7
```

INC A — Increment Accumulator

Opcode 17H

0 0 0 1	0 1 1 1

Accumulator contents are incremented by one.

(A) ← (A)+1

Example: Increment contents of location 100 in external data memory.

```
        MOV R0,#100         MOVE '100' DEC TO ADDRESS
*                           REG 0
        MOVX A,@R0          MOVE CONTENTS OF LOCATION
*                           100 TO ACC
        INC A               INCREMENT ACC
        MOVX @R0,A          MOVE ACC CONTENTS TO
*                           LOCATION 101
```

INC Rr — Increment Register

Opcodes 18H to 1FH

0 0 0 1	1 r r r

The contents of working register 'r' are incremented by one.

(Rr) ← (Rr)+1 r=0-7

```
Example     INC R0          INCREMENT ADDRESS REG 0
```

INC @Rr — Increment Data Memory Location

Opcodes 10H, 11H

0 0 0 1	0 0 0 r

The contents of the resident data memory location addressed by register 'r' (bits 0-5*) are decremented by one.

((R)) ← ((Rr))+1 r=0-1

```
Example:    MOV R1,#H'3F'       MOVE 3F TO REG 1
            INC @R1             INCREMENT LOCATION 63
```

INS A,BUS — Strobed Input of BUS Data to Accumulator (8048 only)

Opcode 08H

0 0 0 0	1 0 0 0

A 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped.

(A) ← (BUS)

Example:

```
        INS A,BUS          INPUT BUS CONTENTS
*                          TO ACC
```

JBb address — Jump If Accumulator Bit is Set (Not in 8021)

Opcode 12H, 32H, 52H, 72H, 92H, B2H, D2H, F2H

b_2 b_1 b_0 1	0 0 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

(PC_{0-7}) ← addr — If Bb=1
(PC) ← (PC)+2 — If Bb=0

Example:

```
        JB4 NEXT           JUMP TO 'NEXT' ROUTINE
*                          IF ACC BIT 4 IS ONE
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JC address — Jump If Carry is Set

Opcode F6H

1 1 1 1	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

(PC_{0-7}) ← addr — If C=1
(PC) ← (PC)+2 — If C=0

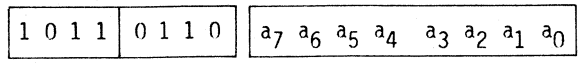
Example:

```
        JC OVFLOW          JUMP TO 'OVFLOW' ROUTINE
*                          IF CARRY IS ONE
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JF0 address — Jump If Flag 0 is Set (8048 only)

Opcode B6H



A 2-cycle instruction. Control passes to the specified address if the flag 0 is set to one.

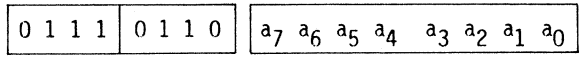
$(PC_{0-7}) \leftarrow$ addr	If F0=1
$(PC) \leftarrow (PC)+2$	If F0=0

Example:

```
        JF0 TOTAL           JUMP TO 'TOTAL' ROUTINE
*                           IF F0=1
```

JF1 address — Jump If Flag 1 is Set (8048 only)

Opcode 76H



A 2-cycle instruction. Control passes to the specified address if the flag 1 is set to one.

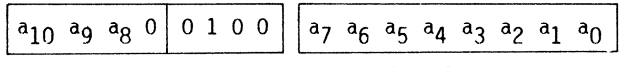
$(PC_{0-7}) \leftarrow$ addr	If F1=1
$(PC) \leftarrow (PC)+2$	If F1=0

Example:

```
        JF1 FILBLUF         JUMP TO 'FILBLUF'
*                           ROUTINE IF F1=1
```

JMP address — Direct Jump Within 2K Block

Opcodes 04H, 24H, 44H, 64H, 84H, A4H, C4H, E4H



A 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 (and bit 12 in 84X1, 84CXX) is determined by the most recent SEL MB instruction.

$(PC_{8-10}) \leftarrow$ addr 8-10	$(PC_{8-10}) \leftarrow$ addr 8-10	84XX, 84CXX
$(PC_{11}) \leftarrow$ MBFF	$(PC_{11,12}) \leftarrow$ MBFF0-1	

Example:

```
        JMP SUBTOT          JUMP TO SUBROUTINE
*                           'SUBTOT'
```

```
        JMP $-6             JUMP TO INSTRUCTION SIX
*                           LOCATIONS BEFORE CURRENT
*                           LOCATION
        JMP H'2F'           JUMP TO ADDRESS '2F' HEX
```

JMPP @A — Indirect Jump within Page

Opcode B3H

1 0 1 1	0 0 1 1

A 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for 'page' portion of the program counter (PC bits 0-7)

$(PC_{0-7}) \leftarrow ((A))$

Example:

```
Assume accumulator contains H'0E'
        JMPP @A             JUMP TO ADDRESS STORED IN
*                           LOCATION 14 IN CURRENT
*                           PAGE
```

JNC address — Jump If Carry is Not Set

Opcode E6H

1 1 1 0	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the carry bit is not set, and therefore, equals zero.

```
(PC0-7) ← addr          If C=0
(PC) ← (PC)+2           If C=1
```

Example:

```
        JNC NOVFLO          JUMP TO 'NOVFLO' ROUTINE
*                           IF CARRY IS ZERO
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JNI address — Jump If Interrupt Input is LOW (8048 only)

Opcode 86H

1 0 0 0	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the interrupt input is LOW (=0), that is an external interrupt has been signalled. (This signal initiates an interrupt service sequence if the external interrupt is enabled).

```
(PC0-7) ← addr          If I=0
(PC) ← (PC)+2           If I=1
```

Example:

```
        JNI EXTINT          JUMP TO 'EXTINT' ROUTINE
*                           IF I=0
```

JNTF address — Jump If Timer Flag is Not Set (Not in 8048)

Opcode 06H

0 0 0 0	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the timer flag is not set, that is, if the timer/counter has not overflowed. Otherwise, the program control falls through the next instruction. Testing the timer flag resets it to zero.

$(PC_{0-7}) \leftarrow$ addr	If TF=0
$(PC) \leftarrow (PC)+2$	If TF=1

```
Example:        JNTF NOTIM          JUMP TO 'NOTIM' ROUTINE IF
                *                   THE TIMER HAS NOT OVERFLOWED
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JNTO address — Jump If Interrupt input is LOW

Opcode 26H

0 0 1 0	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the interrupt input signal is LOW at the time this instruction is executed. Otherwise, the program control falls through the next instruction.

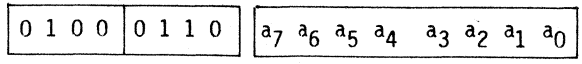
$(PC_{0-7}) \leftarrow$ addr	If TO=0
$(PC) \leftarrow (PC)+2$	If TO=1

```
Example:        JNTO 60             JUMP TO LOCATION 60 DEC
                *                   IF TO=0
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JNT1 address — Jump If T1 input is LOW

Opcode 46H



A 2-cycle instruction. Control passes to the specified address, if the test 1 signal is LOW.

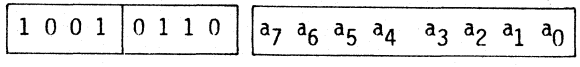
$(PC_{0-7}) \leftarrow$ addr	If T1=0
$(PC) \leftarrow (PC)+2$	If T1=1

```
Example:        JNT1 H'110'     JUMP TO LOCATION '110' HEX
                *               IF T1 IS LOW
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JNZ address — Jump if Accumulator is Not Zero

Opcode 96H



A 2-cycle instruction. Control passes to the specified address if the accumulator contents are non-zero at the time this instruction is executed. Otherwise, the program control passes to the next instruction.

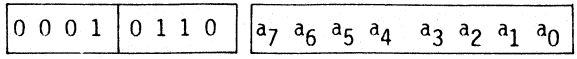
$(PC_{0-7}) \leftarrow$ addr	If A≠0
$(PC) \leftarrow (PC)+2$	If A=1

```
Example:        JNZ ACCNOT      JUMP TO ROUTINE 'ACCNOT'
                *               IF ACC VALU IS NON-ZERO
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JTF address — Jump If Timer Flag is Set

Opcode 16H



A 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter has overflowed. Otherwise, program control passes to the next instruction. Testing the timer flag resets it to zero.

$(PC_{0-7}) \leftarrow$ addr — If TF=1
$(PC) \leftarrow (PC)+2$ — If TF=0

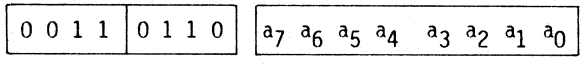
Example:

```
        JTF TIMER       JUMP TO TIMER ROUTINE
*                       IF TIMER HAS OVERFLOWED
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JTO address — Jump if Interrupt input is HIGH

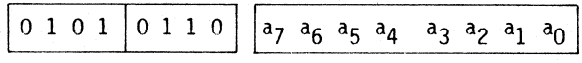
Opcode 36H



(See Instruction JT1)

JT1 address — Jump If Test 1 is HIGH

Opcode 56H



A 2-cycle instruction. Control passes to the specified address, if the T1 input pin is HIGH at the time this instruction is executed. Otherwise program control passes to the next instruction.

$(PC_{0-7}) \leftarrow$ addr — If T1=1
$(PC) \leftarrow (PC)+2$ — If T1=0

```
Example:        JT1 COUNT          JUMP TO 'COUNT' ROUTINE
                *                  IF T1 IS HIGH
```

Note: If this instruction begins in location 255 of a page, the target address is located in the following page.

JZ address — Jump if Accumulator is Zero

Opcode C6H

1 1 0 0	0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed. Otherwise, program control passes to the next instruction.

$(PC_{0-7}) \leftarrow$ addr If A=0
$(PC) \leftarrow (PC)+2$ If A≠0

```
Example:        JZ H'A3'           JUMP TO LOCATION 'A3' HEX
                *                  IF ACC VALU IS ZERO
```

MOV A,Dx — Move derivative register contents to accumulator

Opcode 8CH

1 0 0 0	1 0 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-byte, 2-cycle instruction. The contents of the derivative register (x = 0 to 255) addressed by the second byte are moved to the accumulator.

$(A) \leftarrow (Dx)$

```
Example:        MOV A,D5           MOVE THE CONTENTS OF
                                   DERIVATIVE REGISTER 5 TO
                                   THE ACCUMULATOR
```

MOV A,#data — Move Immediate Data to Accumulator

Opcode 23H

0 0 1 0	0 0 1 1	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

$(A) \leftarrow$ data

```
Example:        MOV A,#H'A3'       MOV 'A3' HEX TO ACC
```

MOV A,PSW — Move PSW Contents to Accumulator

Opcode C7H

1 1 0 0	0 1 1 1

The contents of the program status word are moved to the accumulator.

Example: Jump to the 'RBISET' routine if Register Bank Switch, bit 4, is set.

```
        MOV A,PSW           MOVE PSW CONTENTS TO ACC
        JB4 RBISET          JUMP TO 'RBISET' IF ACC
*                           BIT 4=1
```

MOV A,Rr — Move Register Contents to Accumulator

Opcode F8H to FFH

1 1 1 1	1 r r r

8-bits of data are moved from working register 'r' into the accumulator.

(A) ← (Rr) r=0-7

Example:

```
        MOV A,R3            MOVE CONTENTS OF REG
*                           TO ACC
```

MOV A,@Rr — Move Data Memory Contents to Accumulator

Opcode FOH, F1H

1 1 1 1	0 0 0 r

The contents of the resident data memory location, addressed by working register 'r' (bits 0-5*) are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((Rr)) r=0-1

Example: Assume R1 contains 00110110

```
        MOV A,@R1           MOVE CONTENTS OF DATA MEM
*                           LOCATION 54 TO ACC
```

MOV A,Sn — Move Serial I/O Register Contents to Accumulator

Opcode OCH, ODH (84X1, 84CXX family not 8048, 8422/42 or 84C12)

0 0 0 0	1 1 s s

A 2-cycle instruction. Contents of the serial I/O register 's' are moved to the accumulator.

(A) ← (Ss) s=0-1

Example:

```
        MOV A,SO            MOVE CONTENTS OF SERIAL I/O
*                           DATA REGISTER TO ACC
```

MOV A,T — Move Timer/Counter Contents to Accumulator

Opcode 42H

0 1 0 0	0 0 1 0

The contents of the timer/event-counter register are moved to the accumulator.

(A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 is set.

```
        MOV A,T              MOVE TIMER CONTENTS TO
*                            ACC
        JB6 EXIT             JUMP TO 'EXIT' ROUTINE IF
*                            ACC BIT 6 IS SET
```

MOV PSW,A — Move Accumulator Bit 3 to Prescaler Switch (84X1, 84CXX)

Opcode D7H

1 1 0 1	0 1 1 1

The content of accumulator bit 3 is moved into the prescaler switch.

(PSW) ← (A_3)

Example: Set the timer to 'module 1' mode

```
        MOV A,#H'08'         SET ACC BIT 3
        MOV PSW,A            SET PSW TO ONE
```

MOV PSW,A — Move Accumulator contents to PSW (8048 only)

Opcode D7H

1 1 0 1	0 1 1 1

The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

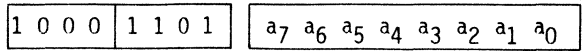
(PSW) ← (A)

Example: Move up the stack pointer by two memory locations, i.e. increment the pointer by one.

```
        MOV A, PSW           MOVE PSW CONTENTS TO ACC
        INC A                INCREMENT ACC BY ONE
        MOV PSW, A           MOVE ACC CONTENTS TO PSW
```

MOV Dx,A — Move accumulator contents to derivative register (84CXX only)

Opcode 8DH



A 2-byte, 2-cycle instruction. The contents of the accumulator are moved to the derivative register, addressed by the second byte.

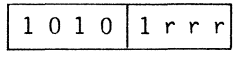
(Dx) ← (A) (x = 0 to 255)

Example:

```
MOV D2,A                MOVE CONTENTS OF ACC. TO
                        DERIVATIVE REGISTER 2
```

MOV Rr,A — Move Accumulator Contents to Register

Opcode A8H to AFH



The contents of the accumulator are moved to the working register 'r'.

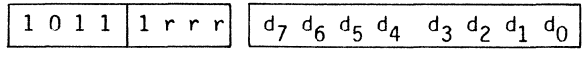
(Rr) ← (A) r=0-7

Example:

```
        MOV R0,A        MOVE CONTENTS OF ACC TO
*                       REG 0
```

MOV Rr,#data — Move Accumulator Contents to Register

Opcode B8H to BFH



A 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

(Rr) ← data r=0-7

Examples:

```
        MOV R4,#HEXTEN      THE VALUE OF THE SYMBOL
*                           'HEXTEN' IS MOVED INTO
*                           REG 4
        MOV R5,#P1(RR)      THE VALUE OF THE
*                           EXPRESSION 'P1(RR)'
*                           IS MOVED INTO REG 5
        MOV R3,#H'AD'       'AD' HEX IS MOVED INTO
*                           REG 3.
```

MOV @Rr,A — Move Accumulator Contents to Data Memory

Opcode A0H, A1H

1 0 1 0	0 0 0 r

A 2-cycle instruction. The contents of the accumulator are moved to the resident data memory location whose address is specified by bits (0-5*) of working register 'r'. Register 'r' contents are unaffected.

((Rr)) ← (A) r=0-1

Example: Assume R0 contains 00000111

```
        MOV @R0,A          MOVE CONTENTS OF ACC TO
*                          LOCATION 7 (REG 7)
```

MOV @Rr,#data — Move Immediate Data to Data Memory

Opcode B0H, B1H

1 0 1 1	0 0 0 r	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by the working register 'r', bits (0-5*).

((Rr)) ← data r=0-1

Example: Move the hexadecimal value AC3F to locations 61-62.

```
MOV R0,#61          MOVE '61' DEC TO ADDR REG 0
MOV @R0,#H'AC       MOVE 'AC' HEX TO LOCATION 61
INC R0              INCREMENT POINTER
MOV @R0,#H'3F'      MOVE '3F' HEX TO LOCATION 62
```

MOV Ss,A — Move Accumulator Contents to Serial I/O Register

Opcodes 3CH to 3EH (84X1, 84CXX only not 84C12 or 8422/42)

0 0 1 1	1 1 s s

A 2-cycle instruction. The contents of the accumulator are moved to the serial I/O register 's'

(Ss) ← (A) s=0-2

Example:

```
      MOV SO,A          MOVE CONTENTS OF ACC TO SIO
*                       DATA REGISTER
```

MOV Ss,#data — Move Immediate Data to Serial I/O Register

Opcode 9CH to 9EH (84X1, 84CXX only not 84C12 or 8422/42

1 0 0 1	1 1 s s	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. The byte specified by 'data' is moved to the serial I/O register 's'

(Ss) ← data s=0-2

Example:

```
      MOV S2,#H'28'     LOAD SIO CLOCK REGISTER
```

MOV Ss, #data — Move Immediate data to serial I/O Register (84CXX only)

Opcode 3CH

0 0 1 1	1 1 s s	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. The byte specified by 'data' is moved to the serial I/O registers.

(Ss) ← data s=0-2

Example:

```
      MOV S2, #28H      LOAD SIO CONTROL REGISTER S2
```

MOV T,A — Move Accumulator Contents to Timer/Counter

Opcode 62H

0 1 1 0	0 0 1 0

The contents of the accumulator are moved to the timer/event counter register.

(T) ← (A)

Example: Initialize and start event counter.

```
      CLR A             CLEAR ACC TO ZEROS
      MOV T,A           MOVE ZEROS TO EVENT COUNTER
      STRT CNT          START COUNTER
```

MOVD A,Pp — Move Port bits 4-7 Data to Accumulator (8048 only)

Opcode 0CH to 0FH

0 0 0 0	1 1 p p

A 2-cycle instruction. Data on 8243 Port 'p' is moved (read) to accumulator bits 0-3.

Accumulator bits 4-7 are zeroed.

(0-3) ← (Pp) p=4-7
(4-7) ← 0

Note: Bits 0-1 of the opcode are used to represent ports 4-7. If coding in binary rather than assembly language, the mapping of as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

Example:

```
      MOVD A,P5          MOVE PORT 5 DATA TO ACC
*                        BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A — Move Accumulator Data to Port 4-7 (8048 only)

Opcode 3CH to 3FH

0 0 1 1	1 1 p p

Data in accumulator bits 0-3 is moved (written) to 8243 Port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping).

$(Pp) \leftarrow (A_{0-3})$ p=4-7

Example: Move data in accumulator to ports 4 and 5.

```
      MOVD P4,A          MOVE ACC BITS 0-3 TO PORT 4
      SWAP A             EXCHANGE ACC BITS 0-3 AND
*                        4-7
      MOVD P5,A          MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A — Move Current Page Data to Accumulator

Opcode A3H

1 0 1 0	0 0 1 1

A 2-cycle instruction. The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation.

$(PC_{0-7}) \leftarrow (A)$
$(A) \leftarrow ((PC))$

NOTE: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory, @A addresses a location in the following page.

Example:

```
        MOV  A,#128          MOVE '128' DEC TO ACC
        MOVP A,@A            CONTENTS OF 129th LOCATION
*                            IN CURRENT PAGE ARE MOVED
*                            TO ACC.
```

MOVP3 A,@A — Move Page 3 Data to Accumulator (Only in 8048)

Opcode E3H

1 1 1 0	0 0 1 1

A 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

$(PC_{0-7}) \leftarrow (A)$
$(PC_{8-11}) \leftarrow 1100$
$(A) \leftarrow ((PC))$

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

```
        MOV A,#H'B8'         MOVE 'B8' HEX TO ACC
*                            (10111000)
        ANL A,#H'7F'         LOGICAL AND ACC TO MASK
*                            BIT 7 (00111000)
        MOVP3 A,@A           MOVE CONTENTS OF LOCATION
*                            '38' HEX IN PAGE 3 TO ACC
*                            (ASCII '8')
```

Access contents of location in page 3 labelled TAB1. Assume current program location is not in page 3.

```
        MOV A,#LOW TAB1         ISOLATE BITS 0-7 OF LABEL
*                               ADDRESS VALUE
        MOVP3 A,@A              MOVE CONTENTS OF PAGE 3
*                               LOCATION LABELLED 'TAB1'
*                               TO ACC.
```

MOVX A,@Rr — Move External-Data Memory Contents to Accumulator

Opcode 80H, 81H (Only in 8048)

1 0 0 0	0 0 0 r

A 2-cycle instruction. The contents of the external data memory location addressed by register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((Rr)) r=0-1

Example: Assume R1 contains 01110110

```
        MOV A,@R1               MOVE CONTENTS OF LOCATION
*                               76H TO ACC.
```

MOVX @Rr,A — Move Accumulator Contents to External Data Memory

Opcode 90H, 91H (Only in 8048)

1 0 0 1	0 0 0 r

A 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'r'. Register 'r' contents are unaffected.

((Rr)) ← (A)

Example: Assume R0 contains 11000111

```
        MOV @R0,A               MOVE CONTENTS OF ACC TO
*                               LOCATION C7H IN EXPANDED
*                               DATA MEMORY
```

NOP — The NOP Instruction

Opcode 00H

0 0 0 0	0 0 0 0

No operation is performed. Execution continues with the next instruction.

ORL A,Rr

Logical OR Accumulator With Register Mask

Opcode 48H to 4FH

0 1 0 0	1 r r r

Data in the accumulator is logically ORed with the mask contained in working register 'r'.

(A) ← (A) OR (Rr) r=0-7

Example:

```
        ORL A,R4          'OR' ACC CONTENTS WITH
*                         MASK IN REG 4
```

ORL A,@Rr

Logical OR Accumulator With Memory Mask

Opcode 40H, 41H

0 1 0 0	0 0 0 r

Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register 'r', (bits 0-5*).

(A) ← (A) OR ((Rr)) r=0-1

```
        MOV R0,#H'3F'     MOVE '3F' HEX TO REG 0
        ORL A,@R0         'OR' ACC CONTENTS WITH MASK
*                         IN LOCATION 63.
```

ORL A,#data

Logical OR Accumulator With Immediate Mask

Opcode 43H

0 1 0 0	0 0 1 1	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

(A) ← (A) OR data

Example:

```
        ORL A,#'X'        'OR' ACC CONTENTS WITH MASK
*                         01011000 (ASCII VALUE OF
*                         'X').
```

ORL BUS,#data — Logical OR BUS With Immediate Mask (Only in 8048)

Opcode 88H

1 0 0 0	1 0 0 0	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior implementation an 'OUTL BUS,A' instruction.

(BUS) ←(BUS) OR data

Example:

```
        ORL BUS,#HEXMSK         'OR' BUS CONTENTS WITH
*                               MASK EQUAL VALUE OF SYMBOL
*                               'HEXMSK'
```

ORL Dx,A — Logical OR derivative register contents with accumulator (84CXX family not 84C12)

Opcode 8FH

1 0 0 0	1 1 1 1	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

A 2-byte, 2-cycle instruction. The contents of the accumulator are logically ORed with the contents of the derivative register addressed by the second byte, and the result placed in the same derivative register.

(Dx) ←(Dx) OR (A) (x = 0 to 255)

ORL Pp,#data — Logical OR Port With Immediate Mask

Opcodes 89H to 8AH for 8048
88H to 8AH for 84X1, 84CXX
88H to 89H for 84C12

1 0 0 0	1 0 p p	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. Port 'p' data is logically ORed with an immediately-specified mask.

(Pp) ←(Pp) OR data p=1-2 (p=0-2 for 84X1, 84CXX)

Example:

```
        ORL P1,#H'FF'           'OR' PORT 1 CONTENTS WITH
*                               MASK 'FF' HEX (SET PORT 1
*                               TO ALL ONES)
```

ORLD Pp,A — Logical OR Port 4-7 With Immediate Mask (Not 84X1 or 84CXX)

Opcode 8CH to 8FH

1 0 0 0	1 1 p p

A 2-cycle instruction. Data on Port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

(Pp) ←(Pp) AND (A_{0-3}) p=4-7

Example:

```
        ORLD P6,A               'OR' PORT 6 CONTENTS
*                               WITH ACC BITS 0-3
```

OUTL BUS,A

Output Accumulator Data to BUS (8048 only)

Opcode 02H (8048), 90H (84X1)

A 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction.

Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Does not apply for OUTL P0,A

(BUS) ← (A)

Example:

```
        OUTL BUS,A              QUTPUT ACC CONTENTS TO BUS
```

OUTL Pp,A

Output Accumulator Data to Port

Opcodes 39H, 3AH for 8048
38H, 39H, 3AH for 84X1, 84CXX
38H, 39H for 84C12

0 0 1 1	1 0 p p

A 2-cycle instruction. Data residing in the accumulator is transferred (written) to Port'p' and latched.

(Pp) ← (A) p=1-2 (p=0-2 in 84XX, 84CXX)

Example:

```
        MOV A,R7                MOVE REG 7 CONTENTS TO ACC
        OUTL P1,A               OUTPUT ACC CONTENTS TO
*                               PORT 1
        MOV A,R6                MOVE REG 6 CONTENTS TO ACC
        OUTL P2,A               OUTPUT ACC CONTENTS TO
*                               PORT 2
```

RET — Return Without PSW Restore

Opcode 83H

1 0 0 0	0 0 1 1

A 2-cycle instruction. The stack pointer is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored (PSW bits 4,6,7 in 84XX, 84CXX)

(SP) ← (SP)-1
(PC) ← ((SP))

RETR — Return With PSW Restored

Opcode 93H

1 0 0 1	0 0 1 1

A 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 (bits 4, 6 and 7 in 84X1,84CXX) of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt service routine, but should not be used within it, as RETR signals the end of an interrupt routine.

(SP) ← (SP)-1	(SP) ← (SP)-1	
(PC) ← ((SP))	(PC) ← ((SP))	84X1
(PSW 4-7) ← ((SP))	($PSW_{4,6,7}$) ← ((SP))	84CXX

RL A — Rotate Left Without Carry

Opcode E7H

1 1 1 0	0 1 1 1

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

(An+1) ← (An) n=0-6
(A0) ← (A7)

Example: Assume accumulator contains 10110001

```
      RL A                    NEW ACC CONTENTS ARE
*                             01100011.
```

RLC A

Rotate Left Through Carry

Opcode F7H

1 1 1 1	0 1 1 1

The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

(An+1) ← (An) n=0-6

(A0) ← (C)
(C) ← (A7)

Example:

Assume accumulator contains a 'signed' number; isolate sign without changing value.

```
        CLR C           CLEAR CARRY TO ZERO
        RLC A           ROTATE ACC LEFT, SIGN
*                       BIT (7) IS PLACED IN CARRY
        RR A            ROTATE ACC RIGHT - VALUE
*                       (BITS 0-6) IS RESTORED
*                       CARRY UNCHANGED, BIT 7
*                       IS ZERO
```

RR A

Rotate Right Without Carry

Opcode 77H

0 1 1 1	0 1 1 1

The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

(An+1) ← (An) n=0-6
(A7) ← (A0)

Example:

Assume accumulator contains 10110001

```
        RR A            NEW ACC CONTENTS ARE
*                       11000110
```

RRC A — Rotate Right Through Carry

Opcode 67H

0 1 1 0	0 1 1 1

The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

(An)◄— (An+1) n=0-6
(A7)◄— (C)
(C)◄— (A0)

Example: Assume carry is not set and accumulator contains 10110001.

```
        RRC A                   CARRY IS SET AND ACC
*                               CONTAINS 01011000
```

SEL MB0 — Select Memory Bank 0 (not 84C12)

Opcode E5H

1 1 1 0	0 1 0 1

PC bit 11 (and bit 12 for 84X1, 84CXX) is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

(MBFF)◄— 0 [(MBFF1,0)◄— 0,0]- 84X1, 84CXX

SEL MB1 — Select Memory Bank 1 (not 84C12)

Opcode F5

1 1 1 1	0 1 0 1

PC bit 11 (bit 12 also set to zero for 84X1, 84CXX) is set to '1' on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

(MBFF) ◄—1 [(MBFF1,0) ◄— 0,1]- 84X1, 84CXX

SEL MB2

Select Memory Bank 2 (84X1, 84CXX, not 84C12)

Opcode A5H

1 0 1 0	0 1 0 1

PC bit 11 is set to zero and PC bit 12 is set to 1 on the next JMP or CALL instruction. All references to program memory addresses fall within the range 4092-6143.

(MBFF1,0) ← 1,0

SEL MB3

Select Memory Bank 3 (84X1, 84CXX not 84C12)

Opcode B5H

1 0 1 1	0 1 0 1

PC bits 11 and 12 is set to 1 on the next JMP or CALL instruction. All references to program memory addresses fall within the range 6144-8191.

(MBFF1,1) ← 1,1

SEL RB0 — Select Register Bank 0

Opcode C5H

1 1 0 0	0 1 0 1

PSW bit is set to 0. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

(RBS) ← 0

SEL RB1 — Select Register Bank 1

Opcode D5H

1 1 0 1	0 1 0 1

PSW bit 4 is set to '1'. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

(RBS) ← 1

Example: Assume an external interrupt has accurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

```
LOC3   JNI INIT              JUMP TO ROUTINE 'INIT' IF
*                            INTERRUPT INPUT IS ZERO
INIT   MOV R7,A              MOVE ACC CONTENTS TO
*                            LOCATION 7
       SEL RB1               SELECT REG BANK 1
       MOV R7,#H'FA'         MOVE 'FA' HEX TO LOCATION 31
       .
       .
       .
       SEL RB0               SELECT BANK 0
       MOV A,R7              RESTORE ACC FROM LOCATION 7
       RETR                  RETURN - RESTORE PC AND PSW
```

STOP

Stop Processor (84CXX only)

Opcode 22H

0 0 1 0	0 0 1 0

The processor enters the STOP mode. The oscillator is switched off. The internal status of the CPU, RAM contents and the state of I/O ports are not affected. The processor can be brought out of the STOP mode either by an active signal at the external interrupt input or by an external RESET signal. When one of these two signals is applied at the input of the processor, an internal delay is provided to ensure that before restarting, all internal clocks are working correctly.

If the processor leaves the STOP mode via RESET, a normal RESET sequence is executed.

If the processor leaves the STOP mode by pulling the external interrupt pin LOW, an interrupt sequence is only executed if the external interrupt has been enabled. In this event, the processor resumes the normal program sequence after returning from the interrupt routine, as in the normal mode. If the external interrupt is not enabled, the processor continues the normal program sequence, executing the instruction following the STOP instruction in main program.

Note: The processor is restarted by a LOW level applied at the INT/T0 pin, and not by a HIGH-to-LOW transition as in a normal interrupt mechanism.
If the INT/T0 level is LOW during the STOP instruction then the STOP mode instruction is ignored.

STOP TCNT — Stop Timer/Event Counter

Opcode 65H

0 1 1 0	0 1 0 1

This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
        DIS TCNTI           DISABLE TIMER INTERRUPT
        CLR A               CLEAR ACC TO ZEROS
        MOV T,A             MOVE ZEROS TO TIMER
        MOV R7,A            MOVE ZEROS TO REG 7
        STRT T              START TIMER
MAIN    JTF COUNT           JUMP TO ROUTINE 'COUNT'
*                           IF TF=1 AND CLEAR TIMER FLAG
        JMP MAIN            CLOSE LOOP
COUNT   INC R7              INCREMENT COUNTER
        MOV A,R7            MOVE REG 7 CONTENTS TO ACC
        JB3 INT             JUMP TO 'INT' ROUTINE IF
*                           ACC BIT 3 IS SET (REG 7=8)
        JMP MAIN            OTHERWISE RETURN TO ROUTINE
*                           'MAIN'
        .
        .
        .
INT     STOP TCNT           STOP TIMER
        JMP H'7'            JUMP TO LOCATION 7 (TIMER
*                           INTERRUPT ROUTINE)
```

STRT CNT — Start Event Counter

Opcode 45H

0 1 0 0	0 1 0 1

The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each HIGH-to-LOW (LOW-to-HIGH for 84X1, 84CXX) transition on the T1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T1 transition.

```
EN   TCNTI          ENABLE COUNTER INTERRUPT
MOV  A,#H'FF'       MOVE ALL ONES TO ACC
MOV  T,A            LOAD COUNTER
STRT CNT            INPUT AND START
```

STRT T — Start Timer

Opcode 55H

0 1 0 1	0 1 0 1

Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles for the 8048 and 8021. It can be incremented on each cycle or every 32 cycles for the 84X1-84CXX, depending on the state of the prescaler switch. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
        CLR A                   CLEAR ACC TO ZEROS
        MOV T,A                 MOVE ZEROS TO TIMER
        EN  TCNTI               ENABLE TIMER INTERRUPT
        STRT T                  START TIMER
```

SWAP A — Swap Nibbles Within Accumulator

Opcode 47H

0 1 0 0	0 1 1 1

Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

$(A_{4-7}) \longleftrightarrow (A_{0-3})$

Example: Pack bits 0-3 of locations 50-51 into location 50.

```
        MOV  R0,#50             MOVE '50' DEC TO REG 0
        MOV  R1,#51             MOVE '51' DEC TO REG 1
        XCHD A,@R0              EXCHANGE BITS 0-3 OF ACC
*                               AND LOCATION 50
        SWAP A                  SWAP BITS 0-3 AND 4-7 OF
*                               ACC
        XCHD A,@R1              EXCHANGE BITS 0-3 OF ACC
*                               AND LOCATION 51
        MOV  @R0,A              MOVE CONTENTS OF ACC TO
*                               LOCATION 50
```

XCH A,Rr — Exchange Accumulator Register contents

Opcode 28H to 2FH

0 0 1 0	1 r r r

The contents of the accumulator and the contents of working register 'r' are exchanged.

$(A) \longleftrightarrow (Rr)$ r=0-7

Example: Move PSW contents to Reg 6 without losing accumulator contents.

```
        XCH A,R6                EXCHANGE CONTENTS OF REG 6
*                               AND ACC
```

```
        MOV A,PSW           MOVE PSW CONTENTS TO ACC
        XCH A,R6            EXCHANGE CONTENTS OF REG 6
*                           AND ACC AGAIN
```

XCH A,@Rr — Exchange Accumulator and Data Memory Contents

Opcodes 20H, 21H

0 0 1 0	0 0 0 r

The contents of the accumulator and the contents of the resident data memory location addressed (bits 0-5*) of register 'r' are exchanged. Register 'r' contents are unaffected.

(A) ⟷ ((Rr)) r=0-1

Example: Decrement contents of location 52.

```
        MOV R0,#52          MOVE 52 DEC TO ADDRESS
*                           REG 0
        XCH A,@R0           EXCHANGE CONTENTS OF ACC
*                           AND LOCATION 52
        DEC A               DECREMENT ACC CONTENTS
        XCH A,@R0           EXCHANGE CONTENTS OF ACC
*                           AND LOCATION 52 AGAIN
```

XCHD A,@Rr — Exchange Accumulator and Data Memory 4-bit Data

Opcode 30H, 31H

0 0 1 1	0 0 0 r

This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5*) of register 'r'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'r' are unaffected.

(A_{0-3}) ⟷ ((Rr0-3) r=0-1

Example: Assume program counter contents have been stacked in locations 22-23.

```
        MOV R0,#22          MOVE '22' DEC TO REG 0
        CLR A               CLEAR ACC TO ZEROS
        XCHD A@R0           EXCHANGE BITS 0-3 OF ACC
*                           AND LOCATION 22 (BIT 8-11
*                           AF PC ARE ZEROED)
```

XRL A,Rr — Logical XOR Accumulator With Register Mask

Opcodes D8H to DFH

1 1 0 1	1 r r r

Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

(A) ← (A) XOR (Rr) r=0-7

Example:

```
        XRL A,R4            'XOR' ACC CONTENTS WITH
*                           MASK IN REG 4
```

XRL A,@Rr — Logical XOR Accumulator With Memory Mask

Opcode D0H, D1H

1 1 0 1	0 0 0 r

Data in the accumulator is EXCLUSIVE ORed with the mask contained in the resident data memory location addressed by register 'r', bits 0-5*).

(A) ← (A) XOR ((Rr)) r=0-1

```
        MOV R1,#H'20'       MOVE '20' HEX TO REG 1
        ORL A,@R1           'XOR' ACC CONTENTS WITH
*                           MASK IN LOCATION 32.
```

XRL A,#data — Logical XOR Accumulator With Immediate Mask

Opcode D3H

1 1 0 1	0 0 1 1	d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

A 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

(A) ← (A) OR data

Example:

```
        XOR A,#HEXTWO       XOR CONTENTS OF ACC WITH
*                           MASK EQUAL VALUE OF SYMBOL
*                           'HEXTWO'
```

10. Software examples for the MAB8048/MAB84X1/PCF84CXX family

CONTENTS – SOFTWARE EXAMPLES

		page
1.0	INTRODUCTION	10-4
2.0	GENERAL SOFTWARE EXAMPLES: APPLICABLE TO THE 8048, 84X1/84CXX FAMILY	10-4
2.1	Double precision unsigned addition subroutine	10-4
2.2	Double precision unsigned subtraction subroutine	10-5
2.3	Double precision product unsigned multiplication	10-5
2.4	Branch on input value	10-6
2.5	Arithmetic shift left and right subroutines	10-6
2.6	BCD to binary conversion subroutine	10-7
3.0	SERIAL I/O SOFTWARE EXAMPLES: APPLICABLE TO THE 84X1/84CXX FAMILY	10-8
3.1	Single-master routines	10-8
3.1.1	Master transmitter routine (MAB84X1 - SAA1300)	10-8
3.1.2	Master receiver routine (MAB84X1 - SAB3028)	10-16
3.1.3	Master transmitter/receiver routine including general call reset (MAB84X1 - SAB3035/36/37)	10-19
3.1.4	Master transmitter/receiver routine with repeated start condition (MAB84X1 - PCD8571)	10-23
3.1.5	Master transmitter routine - CBUS (MAB84X1 - SAA1061)	10-27
3.1.6	Master transmitter routine - CBUS (MAB84X1-2 x PCE2111)	10-32
3.1.7	Master transmitter/receiver routine - special format (IBUS)	10-37
3.2	Slave routines	10-41
3.2.1	Slave receiver routine (MAB84X1 - master)	10-41
3.2.2	Slave transmitter routine (MAB84X1 - master)	10-44
3.2.3	Slave transmitter/receiver routine including general call reception (MAB84X1 - master)	10-46
3.3	Multi-master routines	10-50
3.3.1	Master transmitter/receiver routine with repeated start (MAB84X1 - PCD8571)	10-50
3.3.2	Master transmitter/receiver routine; slave transmitter/receiver routine (MAB84X1 - MAB84X1)	10-55
3.3.3	Further Multi-master routines	10-62
3.3.3.1	I^2C-bus communications multi-master/slave routine	10-63
3.3.3.2	I^2C-bus system level utility routines	10-72
4.0	SERIAL I/O SOFTWARE EXAMPLES: APPLICABLE TO MAB8048	10-84
4.1	Single-master routines	10-84
4.1.1	Master transmitter routine (MAB8048 - SAA1300)	10-84
4.1.2	Master receiver routine (MAB8048 - SAB3028)	10-87
4.1.3	Master transmitter/receiver routine with repeated start (MAB8048 - PCD8571)	10-91

1.0 INTRODUCTION

Software Examples links the other sections within this manual. However, it is not the only section in which software examples can be found. Flow charts and examples for the MAB84X1 family can also be found in the Serial I/O section; an introduction to the various data formats can be found in the I^2C-bus specification.

All routines in this section are given as examples only- the construction of these routines in section 3.3 gives an indication of how the 84X1 and 8048 operate using the I^2C bus at system level. Each example is taken from a Philips Microcomputer Development System (PMDS II) using 84X1 cross assembler (Rel: 3.0). (This does not apply to section 4 where the 8048 cross assembler was used).

2.0 GENERAL SOFTWARE EXAMPLES (APPLICABLE FOR 8048 FAMILY AND 84X1 FAMILY)

2.1 Double precision unsigned addition subroutine

Add two unsigned 16 bit quantities from data memory locations (OPRNDA, OPRNDA + 1) and store result in data memory locations RSLT, RSLT + 1.

R0 and R1 are used as pointers, R2 for temporary storage.

```
                                  9 OPRNDB EQU     H'20'          DEFINE OPERANDS: OPERAND B HIGH: LOC 20
                                 10 *                                              OPERAND B LOW:  LOC 21
                                 11 OPRNDA EQU     OPRNDB+2                       OPERAND A HIGH: LOC 22
                                 12 *                                             OPERAND A LOW:  LOC 23
                                 13 RSLT   EQU     OPRNDB+4       DEFINE RESULT:  RESULT HIGH:    LOC 24
                                 14 *                                             RESULT LOW:     LOC 25
                                 15 *
000000 B823             1       16 DADD   MOV     R0,#OPRNDA+1   R0 POINTS TO OPERAND A LOW
000002 B921             2       17        MOV     R1,#OPRNDB+1   R1 POINTS TO OPERAND B LOW
000004 F0               3       18        MOV     A,@R0          GET OPERAND A LOW
000005 61               4       19        ADD     A,@R1          ADD OPERAND B LOW
000006 AA               5       20        MOV     R2,A           SAVE RESULT TEMPORARILY
000007 C8               6       21        DEC     R0             R0 POINTS TO OPERAND A HIGH
000008 C9               7       22        DEC     R1             R1 POINTS TO OPERAND B HIGH
000009 F0               8       23        MOV     A,@R0          GET OPERAND A HIGH
00000A 71               9       24        ADDC    A,@R1          ADD OPERAND B HIGH AND CARRY
00000B B824            10       25        MOV     R0,#RSLT       R0 POINTS TO RESULT HIGH
00000D A0              11       26        MOV     @R0,A          STORE RESULT HIGH
00000E 18              12       27        INC     R0             R0 POINTS TO RESULT LOW
00000F FA              13       28        MOV     A,R2           GET RESULT LOW
000010 A0              14       29        MOV     @R0,A          STORE RESULT LOW
000011 83              15       30        RET
000012                          31        END
```

2.2 Double precision unsigned subtraction subroutine

Subtract an unsigned 16 bit subtrahend in data memory locations OPRNDB, OPRNDB + 1 from an unsigned 16 bit minuend in data memory locations OPRNDA, OPRNDA + 1 and store result in data memory locations RSLT, RSL + 1.

R0 and R1 are used as pointers and R2 is used for temporary storage.

```
                                    9 *
                                   10 OPRNDB EQU     H'20'          DEFINE OPERANDS: SUBTRAHEND HIGH: LOC 20
                                   11 *                                              SUBTRAHEND LOW:  LOC 21
                                   12 OPRNDA EQU     OPRNDB+2                        MINUEND HIGH:    LOC 22
                                   13 *                                              MINUEND LOW:     LOC 23
                                   14 RSLT   EQU     OPRNDB+4       DEFINE RESULT:   RESULT HIGH:     LOC 24
                                   15 *                                              RESULT LOW:      LOC 25
                                   16 *
000000 B823             1      17 DSUB   MOV     R0,#OPRNDA+1   R0 POINTS TO MINUEND LOW
000002 B921             2      18        MOV     R1,#OPRNDB+1   R1 POINTS TO SUBTRAHEND LOW
000004 F1               3      19        MOV     A,@R1          GET SUBTRAHEND LOW
000005 37               4      20        CPL     A              MAKE 2'S COMPLEMENT
000006 0301             5      21        ADD     A,#1
000008 AA               6      22        MOV     R2,A           SAVE 2'S COMPLEMENTED SUBTRAHEND LOW
000009 C9               7      23        DEC     R1             R1 POINTS TO SUBTRAHEND HIGH
00000A F1               8      24        MOV     A,@R1          GET SUBTRAHEND HIGH
00000B 37               9      25        CPL     A              COMPLEMENT BUT INCREMENT ONLY IF CARRY
00000C 1300            10      26        ADDC    A,#0           FROM LOW ORDER
00000E 2A              11      27        XCH     A,R2           SAVE 2'S COMPLEMENTED SUBTRAHEND HIGH
00000F 60              12      28        ADD     A,@R0          GET LOW AND ADD MINUEND LOW
000010 B925            13      29        MOV     R1,#RSLT+1     R1 POINTS TO RESULT LOW
000012 A1              14      30        MOV     @R1,A          STORE DIFFERENCE LOW
000013 C9              15      31        DEC     R1             R1 POINTS TO RESULT HIGH
000014 C8              16      32        DEC     R0             R0 POINTS TO MINUEND HIGH
000015 FA              17      33        MOV     A,R2           GET 2'S COMPLEMENTED SUBTRAHEND HIGH
000016 70              18      34        ADDC    A,@R0          ADD MINUEND HIGH WITH CARRY
000017 A1              19      35        MOV     @R1,A          STORE DIFFERENCE HIGH
000018 83              20      36        RET
000019                         37        END
```

2.3 Double precision product unsigned multiplication

Multiply two unsigned 8 bit quantities from data memory locations OPRNDA, OPRNDB and store the 16 bit result in data memory locations RSLT, RSL + 1.

R0, R1 and R2 are used as pointers and temporary storage registers.

R3 is used as a counter.

```
                                    9 *
                                   10 OPRNDA EQU     H'20'          DEFINE OPERAND LOCATIONS
                                   11 OPRNDB EQU     OPRNDA+1
                                   12 RSLT   EQU     OPRNDA+2       DEFINE RESULT LOCATIONS
                                   13 *
000000 B820             1      14 DMUL   MOV     R0,#OPRNDA     R0 POINTS TO MULTIPLIER
000002 B921             2      15        MOV     R1,#OPRNDB     R1 POINTS TO MULTIPLICAND
000004 BB09             3      16        MOV     R3,#9          INITIALIZE LOOP COUNTER
000006 F0               4      17        MOV     A,@R0          GET MULTIPLIER
000007 A8               5      18        MOV     R0,A           SAVE TEMPORAILY IN R0
000008 F1               6      19        MOV     A,@R1          GET MULTIPLICAND
000009 A9               7      20        MOV     R1,A           SAVE TEMPORARILY IN R1
00000A 27               8      21        CLR     A              CLEAR PRODUCT
00000B 97               9      22        CLR     C              CLEAR CARRY
00000C 67              10      23 DMULLP RRC     A              RIGHT SHIFT PRODUCT
00000D 28              11      24        XCH     A,R0           R0 <-- PRODUCT; A <-- MULTIPLIER
00000E 67              12      25        RRC     A              SHIFT MULTIPLIER LSB INTO CARRY
00000F 28              13      26        XCH     A,R0           A <-- PRODUCT; R0 <-- MULTIPLIER
000010 E613            14      27        JNC     ZROBIT         JUMP IF MULTIPLIER LSB WAS 0
000012 69              15      28        ADD     A,R1           ELSE ADD MULTIPLICAND TO PRODUCT
000013 EB0C            16      29 ZROBIT DJNZ    R3,DMULLP      LOOP UNTIL ALL MULTIPLIER BITS TESTED
000015 B922            17      30        MOV     R1,#RSLT       R1 POINTS TO RESULT HIGH
000017 A1              18      31        MOV     @R1,A          STORE RESULT HIGH
000018 19              19      32        INC     R1             R1 POINTS TO RESULT LOW
000019 F8              20      33        MOV     A,R0           GET RESULT LOW
00001A A1              21      34        MOV     @R1,A          STORE
00001B 83              22      35        RET                    RETURN
00001C                         36        END
```

2.4 Branch on input value

This routine inputs a value from an input device on PORT 1 (e.g. keyboard)looks it up in a table and branches to an address specified in another table.

R6 and R7 are used for temporary storage.

```
000000 09              1      9 LOOKUP IN       A,P1                INPUT VALUE FROM PORT 1
000001 AF              2     10        MOV      R7,A                SAVE TEMPORARILY IN R7
000002 BE0F            3     11        MOV      R6,#VALTBL-1        INITIALIZE TABLE LOOK UP POINTER
000004 1E              4     12 LOOKLP INC      R6                  R6 POINTS TO NEXT VALUE IN TABLE
000005 FE              5     13        MOV      A,R6                MOVE POINTER TO ACCU
000006 A3              6     14        MOVP     A,@A                GET A VALUE FROM TABLE
000007 C617            7     15        JZ       NOTFND              JUMP TO ERROR ROUTINE IF END OF LIST
000009 DF              8     16        XRL      A,R7                COMPARE TABLE VALUE WITH INPUT VALUE
00000A 9604            9     17        JNZ      LOOKLP              LOOP IF NO MATCH
00000C FE             10     18        MOV      A,R6                MOVE LOOKUP TABLE POINTER TO ACCU
00000D 0304           11     19        ADD      A,#BRCHTB-VALTBL    ADD OFFSET ONTO BRANCH TABLE
00000F B3             12     20        JMPP     @A                  BRANCH TO ON PAGE ROUTINE
                             21 *
000010 41                    22 VALTBL  DATA     'A'                 TABLE OF LOOKUP VALUES
000011 31                    23        DATA     '1'
000012 0F                    24        DATA     H'0F'
000013 00                    25        DATA     0
                             26 *
000014 18                    27 BRCHTB  DATA     .LOW.ALETTR         TABLE OF ROUTINE ADDRESSES CORRESPONDING
000015 19                    28        DATA     .LOW.ONENUM         TO VALUES ABOVE
000016 1A                    29        DATA     .LOW.CRCHAR
                             30 *
000017 00             13     31 NOTFND  NOP                          ERROR HANDLING ROUTINE STARTS HERE
                             32 *
000018 00             14     33 ALETTR  NOP                          CHARACTER 'A' HANDLING ROUTINE STARTS HERE
                             34 *
000019 00             15     35 ONENUM  NOP                          CHARACTER '1' HANDLING ROUTINE STARTS HERE
                             36 *
00001A 00             16     37 CRCHAR  NOP                          CARRIAGE RETURN HANDLING ROUTINE STARTS HERE
00001B                       38        END
```

2.5 Arithmetic shift left and right subroutines

These subroutines arithmetically shift the 16 bit number formed by the contents of R7 (high-order byte) and the contents of the accumulator (low-order byte).

R6 specifies the number of places to shift.

```
                             8 *
000000 97              1     9 ASLSUB   CLR      C                   CLEAR CARRY SO SHIFT 0'S FROM THE RIGHT
000001 F7              2    10        RLC      A                   ROTATE LOW BYTE LEFT
000002 2F              3    11        XCH      A,R7                GET HIGH BYTE
000003 F7              4    12        RLC      A                   ROTATE LEFT WITH MSB OF LOW BYTE
000004 2F              5    13        XCH      A,R7                EXCHANGE BACK
000005 EE00            6    14        DJNZ     R6,ASLSUB           LOOP UNTILL DONE
000007 83              7    15        RET
                            16 *
000008 97              8    17 ASRSUB  CLR      C                   CLEAR CARRY SO SHIFT 0'S FROM THE LEFT
000009 2F              9    18        XCH      A,R7                GET HIGH ORDER IN ACCU
00000A 67             10    19        RRC      A                   SHIFT RIGHT
00000B 2F             11    20        XCH      A,R7                GET LOW ORDER, SAVE HIGH ORDER
00000C 67             12    21        RRC      A                   SHIFT LOW ORDER RIGHT WITH MSB OF HIGH
00000D EE08           13    22        DJNZ     R6,ASRSUB             ORDER, LOOP UNTIL DONE
00000F 83             14    23        RET
                            24 *
000010                      25        END
```

2.6 BCD to binary conversion subroutine

This subroutine is entered with a two digit BCD number in the accumulator.

The number is converted to binary and the result is returned in the accumulator.

R6 and R7 are used as temporary storage registers.

```
                          8 *
000000 AF            1    9 BCDBIN MOV     R7,A            SAVE BCD NUMBER TEMPORARILY
000001 53F0          2   10        ANL     A,#H'F0'        MASK OUT LOWER DIGIT
000003 47            3   11        SWAP    A               MOVE HIGH DIGIT DOWN
000004 AE            4   12        MOV     R6,A            SAVE TEMPORARILY
000005 E7            5   13        RL      A               MULTIPLY HIGH DIGIT BY 10 DECIMAL
000006 E7            6   14        RL      A               10X=2(4X+X)
000007 6E            7   15        ADD     A,R6
000008 E7            8   16        RL      A
000009 2F            9   17        XCH     A,R7            SWAP RESULT WITH ORIGINAL NUMBER
00000A 530F         10   18        ANL     A,#H'0F'        GET LOW DIGIT
00000C 6F           11   19        ADD     A,R7
00000D 83           12   20        RET
                         21 *
00000E                   22        END
```

3.0 SERIAL I/O SOFTWARE EXAMPLES: APPLICABLE TO MAB84X1 FAMILY

The serial I/O equate list given below, is used throughout the programs in section 3.0 to 3.2 inclusive.

```
3  * SERIAL I/O EQUATE LIST
4  ************************
5  *
6  ***** REGISTER S0' *****
7  ALS     EQU     H'01'           ALWAYS SELECTED BIT
8  SLAB    EQU     H'FE'           SLAVE ADDRESS BITS
9  *
10 ***** REGISTER S0 *****
11 RW      EQU     H'01'           READ/WRITE NOT CONTROL BIT
12 *
13 ***** REGISTER S1 *****
14 * READ AND WRITE BITS
15 MST     EQU     H'80'           MASTER BIT
16 TRX     EQU     H'40'           TRANSMITTER BIT
17 BB      EQU     H'20'           BUS BUSY BIT
18 PIN     EQU     H'10'           PENDING INTERRUPT NOT BIT
19 *
20 * WRITE BITS
21 ESO     EQU     H'08'           ENABLE SERIAL I/O BIT
22 BC2     EQU     H'04'   %)      SIO BITCOUNTER BIT 2
23 BC1     EQU     H'02'   %)      SIO BITCOUNTER BIT 1
24 BC0     EQU     H'01'   %)      SIO BITCOUNTER BIT 0
25 *
26 STRTC   EQU     MST+TRX+BB+PIN+ESO  %)  START CONDITION BITS
27 STOPC   EQU     MST+TRX+PIN+ESO     %)  STOP CONDITION BITS
28 *
29 * READ BITS
30 AL      EQU     H'08'           ARBITRATION LOST BIT
31 AAS     EQU     H'04'           ADDRESSED AS SLAVE BIT
32 ADO     EQU     H'02'           ADDRESS ZERO BIT
33 LRB     EQU     H'01'           LAST RECEIVED BIT
34 *
35 MTTST   EQU     MST+TRX+BB  %)  MASTER TRANSMITTER TEST BITS
36 MRTST   EQU     MST+BB      %)  MASTER RECEIVER TEST BITS
37 *
38 ***** REGISTER S2 *****
39 ASC     EQU     H'20'           ASYNCHRONE CLOCK BIT
40 ACK     EQU     H'40'           WITH ACKNOWLEDGE BIT
41 SCLFB   EQU     H'1F'           SCL CLOCK FREQUENCY BITS
42 *
```

3.1 Single-master routines

These routines can only be used if no other masters are connected to the I^2C-bus. All these routines are based on polling of the PIN bit in register S1.

3.1.1 Master transmitter routine (MAB84X1 - SAA1300)

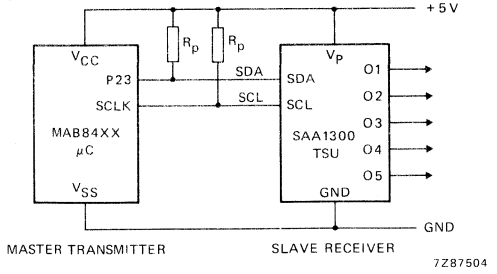


Fig. 3.1 Block diagram of MAB84X1 - SAA1300 configuration.

Initialization:

Normally the serial I/O (SIO) of the MAB84X1 family has to be initialized as follows:

- program the SCL frequency in register S2
- select the "with acknowledge" mode in register S2
- load its own slave address in register S0'
- enable the SIO logic with the ESO bit in register S1
- select the slave receiver mode in register S1
- enable the SIO interrupt

The slave address is not programmed here because there are no other masters connected to the I^2C-bus. So this MAB84X1 microcomputer cannot be addressed as a slave.

Also, the SIO interrupt is not enabled because the routines are based on polling of the PIN bit in status register S1.

```
                          52 IICFR  EQU     H'04'          Fscl = 95 kHz @fxtal = 6,0 MHz
                          53 *
                          54 * POWER-ON RESET:
                          55 *
                          56        PAGE    256
                          57 ROM0   ASECT   ROM
000000                    58        ORG     H'000'
                          59 *
000000 2400          1    60 RESET  JMP     INIT           JUMP TO INITIALIZE ROUTINE


000002                    77        ORG     H'100
                          78 *
000100 9E44          2    79 INIT   MOV     S2,#IICFR+ACK  LOAD SCL FREQUENCY WORD
                          80 *                             SET WITH ACKNOWLEDGE MODE
000102 9D18          3    81        MOV     S1,#PIN+ESO    ENABLE SIO
                          82 *                             SET SLAVE RECEIVER MODE
000104 B8FF          4    83        MOV     R0,#H'FF'      LOAD DATA TO BE WRITTEN TO TSU
                          84 *
```

Main program:

The main program transmits data to TSU (SAA1300) by calling the subroutine WTSU0 or WTSU1.

If the transmission is not successful, the data transfer is repeated, otherwise the data is incremented and the sequence starts again.

The format of the data transfer to TSU is given in Fig. 3.2:

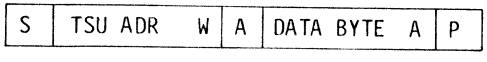


Fig. 3.2 Data format of transfer to TSU.

S	is the START condition
TSU ADR	is the slave address of the TSU
W	is the read/write bit in write state (= 0)
A	is the acknowledge bit; this has to be 0
P	is the STOP condition.

In all these software examples each slave address is defined as an eight bit word with a R/W bit in the WRITE state (= 0).

```
                                102 * The slave address of TSU is 0100 0XX
                                103 TSUAD  EQU     H'40'
                                104 *
                                105 *
000106 18              5        106 MAIN   INC     R0              INCR. DATA TO BE TRANSMITTED
                                107 *
000107 3422            6        108 MAIN1  CALL    WTSU1           TRANSMIT DATA BYTE IN R0 TO TSU
                                109 *                          CALL SUBROUTINE WTSU0 OR WTSU1
000109 9607            7        110        JNZ     MAIN1           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00010B 2406            8        111        JMP     MAIN            CONTINUE
                                112 *
```

Master writes one data byte to TSU (SAA1300) subroutine 1:

To start a transmission, the slave address together with the read/write bit (R/W) - in write state - has to be loaded into the SIO data shift register S0. The transmission really starts when the start condition word (STRTC) is loaded in the SIO status register S1.

The start condition word STRTC is programmed in the SIO equate list as MST + TRX + BB + PIN + ESO. The combination of status bits is always defined with the "+" statement. This can be done while in the SIO equate list; each SIO status bit is defined individually.

The MAB84X1 family microcomputer generates the START condition, slave address, R/W bit and an acknowledge related clock pulse. During this last clock pulse it inputs the acknowledge bit which will be represented by the LRB bit in the status register S1.

If the SIO logic has finished this part of the transmission, this is indicated by the SIO status bit PIN, which becomes 0. Now the SIO status can be tested by comparing (XRL instr.) it with the mask MTTST (= MST + TRX + BB).

If equal (Accumulator = 0), the data transfer continues by loading the data byte - to be transmitted and stored in R0 - into the SIO data register S0. If PIN becomes zero again, the SIO status is tested and the data transfer to TSU is terminated by a STOP condition.

The subroutine returns with the accumulator contents equal to zero if the data transfer to TSU has been successfull.

If the slave address is not acknowledged, the transmission is terminated immediately with a STOP condition and the Accumulator contents equal to H'01'.

If the data byte is not acknowledged, the transmission is also terminated by a STOP condition with the accumulator contents equal to H'01'.

entry: R0 contains data to be transmitted
exit : A = 0 if transmission is successful

```
00010D 9C40          9    143 WTSU0   MOV     S0,#TSUAD       LOAD TSU SLAVE ADDRESS AND WRITE BIT
00010F 9DF8         10    144         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
                          145 *
000111 0D           11    146 WTSU01  MOV     A,S1            FETCH SIO STATUS
000112 9211         12    147         JB4     WTSU01          WAIT FOR PIN = 0
000114 D3E0         13    148         XRL     A,#MTTST        TEST MST/TRX SIO BUS STATUS
000116 961F         14    149         JNZ     WTSU03          JUMP IF NO ACKN. RECEIVED OR IF ERROR
000118 F8           15    150         MOV     A,R0            FETCH DATA TO BE TRANSMITTED
000119 3C           16    151         MOV     S0,A            TRANSMIT DATA BYTE'
                          152 *
00011A 0D           17    153 WTSU02  MOV     A,S1            FETCH SIO STATUS
00011B 921A         18    154         JB4     WTSU02          WAIT FOR PIN = 0
00011D D3E0         19    155         XRL     A,#MTTST        TEST MST/TRX SIO BUS STATUS
                          156 *
00011F 9DD8         20    157 WTSU03  MOV     S1,#STOPC       OUTPUT STOP CONDITION
000121 83           21    158         RET
                          159 *
```

Disturbances of the bus signals:

When using the previous subroutine, the bus can become blocked by disturbances from an external source (e.g. flash-overs in a TV system).

As a safeguard against this, first of all, a digital filter is designed in at the SDA and SCL inputs. Spikes have to be longer than 2 xtal clock cycles Txtal before they will be seen by the internal SIO logic. However to prevent a "bus blocked" situation some software precautions also have to be taken.

The influences of disturbance on the data line are described below. Similar results occur with disturbance on the clock line and are solved in a similar manner to that described below.

Disturbance of a data bit in master transmitter mode can create:

- An arbitration lost situation if the data bit which is output high is completely pulled-down.

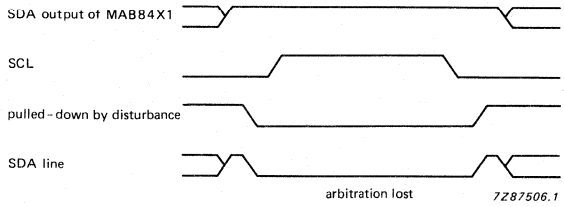


Fig. 3.3. Arbitration lost

If the master loses arbitration, it switches over to the slave receiver mode and inputs the data word by generating clock pulses until the bit counter becomes zero. PIN is then set to zero and the bus status indicated in S1 is 0010 1XXX.

The bit AAS is set if the master's own address is received. When a general call address is received, the bits AAS and ADO are set.

Since there are no other masters in the system, an arbitration lost situation can only occur due to a disturbance in the data bit. In the above subroutine, the microcomputer tries to generate a STOP condition and exits with A ≠0.

- A new start condition if the last part of a data byte which was output high is pulled low.

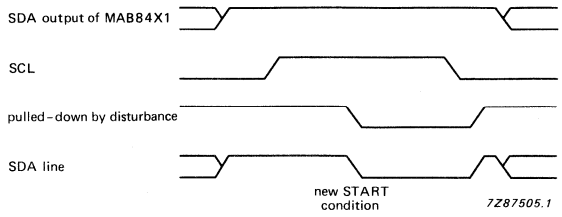


Fig. 3.4. New start condition

Such a new START condition resets the bit counter and starts again to transmit the current contents of S0 as a slave address. Because the contents of S0 is already shifted some bits to the left, this output slave address is not correct.

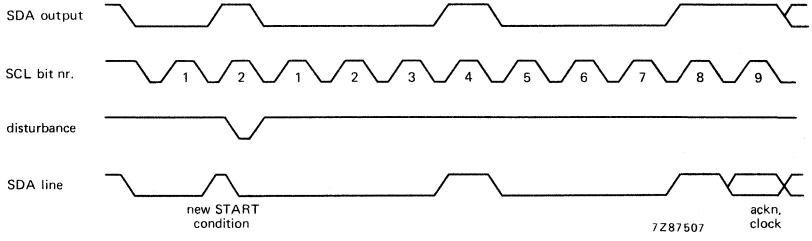


Fig. 3.5.

In the above timing diagram the master wants to transmit the slave address H'44'. However, a new START condition is generated due to a disturbance of the second data bit. At that moment the contents of S0 has become H'11' because S0 has already been shifted to the left twice. Now this H'11' code is transmitted as a new slave address. If PIN becomes zero the bus status will be 1010 000X. After comparison with the MTTST mask, a STOP condition is generated and the subroutine returns with A $\neq$ 0.

- Generation of a STOP condition if the first part of a data bit which is output high is pulled-down.

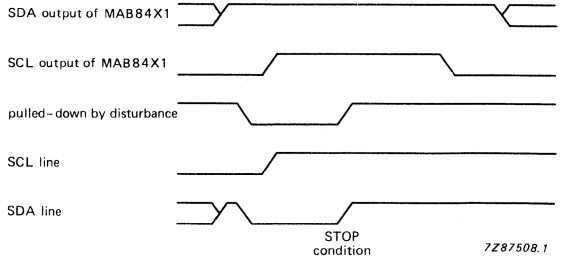


Fig. 3.6.

A STOP condition stops the data transmission and the bus status becomes 0001 100X. The bit counter also stops and stays at its current value. Now the status bit PIN will never become zero.

In the above subroutine the bus is blocked, which is unacceptable.

- Generation of a START/STOP condition, if a small part in the middle of a data bit which is output high is pulled-down.

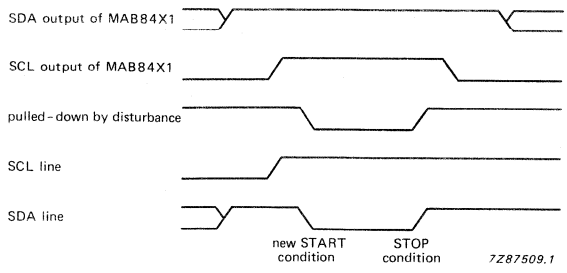


Fig. 3.7.

The new START condition resets the bit counter and the STOP condition stops the data transfer.

To solve a "bus blocked" problem a PIN = 0 time out counter can be added (see subroutine WTSU1 and MTTB1).

Master writes one data byte to TSU (SAA1300) subroutine 2:

This subroutine also returns with A ≠ 0 if the transmission is not successful due to a not received acknowledge or a disturbance of a data bit.

In the first case, the subroutine generates a STOP condition to release the bus. In the second case, the bus is released by returning to the SLAVE mode.

By testing the contents of the accumulator, the main program can repeat the transmission until it succeeds, if necessary.

Because the status of the bus is not always known at the beginning of the subroutine, it is reset with the MOV S1,#PIN+ESO instruction.

*In the second case the bus is released by returning to the slave mode.

If a certain transmission is followed by another transmission, the time between the MOV S1,#STOPC instruction and the MOV S1,#PIN+ESO has to be longer than the time the SIO logic needs to execute the STOP condition. This time is about one SCL clock pulse.

To prevent errors in the case of a slow SCL clock, a test on BB = 0 can be done for a certain maximum time which is longer than one SCL clock cycle before the execution of the MOV S1,#PIN+ESO instruction.

Since the test of PIN = 0 has to be performed twice, a subroutine is used.

entry: R0 contains data to be transmitted,
exit : A = 0 if transmission is successful.

```
000122 9D18          22    295 WTSU1   MOV     S1,#PIN+ESO     RESET BUS STATUS
000124 9C40          23    296         MOV     S0,#TSUAD       LOAD TSU SLAVE ADDRESS AND WRITE BIT
000126 9DF8          24    297         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000128 345A          25    298         CALL    MTTBS           TEST BUS STATUS
                           299 *                       CALL SUBROUTINE MTTB1, MTTB2 OR MTTBS
00012A 9632          26    300         JNZ     ERROR           JUMP IF NO ACK. RECEIVED OR ERROR
00012C F8            27    301         MOV     A,R0            FETCH DATA TO BE TRANSMITTED
00012D 3C            28    302         MOV     S0,A            TRANSMIT DATA BYTE
00012E 345A          29    303         CALL    MTTBS           TEST BUS STATUS
                           304 *                       CALL SUBROUTINE MTTB1, MTTB2 OR MTTBS
000130 C644          30    305         JZ      STOP            JUMP IF ACK. RECEIVED & NO ERRORA
                           306 *
                           307 * NO ACKNOWLEDGE RECEIVED OR ERROR EXIT ROUTINE
                           308 *
000132 D301          31    309 ERROR   XRL     A,#LRB          INVERT  ACK
000134 C642          32    310         JZ      ERROR1          IF NO ACK. OUTPUT STOP COND.
000136 9EC3          33    311         MOV     S2,#IICFR
000138 9E44          34    312         MOV     S2,#IICFR+ACK
00013A 9D18          35    313         MOV     S1,#PIN+ESO     RESET BUS STATUS TWICE IF ERROR FEARED
00013C 9D18          36    314         MOV     S1,#PIN+ESO     RETURN TO SLAVE MODE
00013E 0C            37    315         MOV     A,S0            DUMMY READ
00013F 2302          38    316         MOV     A,#2            TO RETURN WITH A ≠ 0
000141 83            39    317         RET
000142 2301          40    318 ERROR1  MOV     A,#1            TO RETURN WITH A = 1
000144 9DD8          41    319 STOP    MOV     S1,#STOPC       OUTPUT STOP CONDITION
000146 83            42    320         RET
```

Master transmitter test bus status subroutine 1:

In this subroutine a time-out counter is inserted to prevent a bus block in the case where PIN doesn't become 0 because of a disturbance of a data bit.

The contents of register R2 are modified.

```
                                324 *  in case PIN will never become 0 because of a disturbance of a data bit.
                                325 * The contents of register R2 are modified.
                                326 *
000147 BA00             43      327 MTTB1   MOV     R2,#00          LOAD PIN=0 TIME-OUT COUNTER
                                328 *
000149 0D               44      329 MTTB11  MOV     A,S1            FETCH BUS STATUS
00014A 924F             45      330         JB4     MTTB12          JUMP IF PIN = 1
00014C D3E0             46      331         XRL     A,#MTTST        TEST MST/TRX SIO STATUS
00014E 83               47      332         RET
                                333 *
00014F EA49             48      334 MTTB12  DJNZ    R2,MTTB11       DECR. AND TEST TIME-OUT COUNTER
000151 83               49      335         RET
                                336 *
```

Master transmitter test bus status subroutine 2:

If the SCL clock frequency is fixed, the test on PIN = 0 can be replaced by a fixed delay. Therefore subroutine MTTB2 can be used instead of subroutine MTTB1.

NOTE: this can only be done if none of the slaves are able to stretch the SCL clock.

The delay has to be longer than 9 SCL clock pulses. This subroutine saves some bytes in comparison with subroutine MTTB1.

The contents of register R2 are modified.

```
                                345 *
                                346 * The contents of register R2 are modified.
                                347 *
000152 BA0A             50      348 MTTB2   MOV     R2,#10          LOAD DELAY COUNTER
000154 EA54             51      349         DJNZ    R2,$            DECR. DELAY COUNTER
000156 0D               52      350         MOV     A,S1            FETCH BUS STATUS
000157 D3E0             53      351         XRL     A,#MTTST        TEST MST/TRX BUS STATUS
000159 83               54      352         RET
                                353 *
```

Master transmitter test bus status subroutine 3:

If slaves are able to stretch the SCL clock, the PIN = 0 time out counter can be replaced by a test on the MST bit. In this case the subroutine MTTBS can be used instead of subroutines MTTB1 or MTTB2.

As already mentioned, a STOP condition due to a disturbance of a data bit resets the MST bit. In this case, the subroutine MTTBS will be terminated with A ≠ 0 and the routine ERROR will try to generate a STOP condition if a NOT ACK. situation occurs. The routine ERROR performs an escape sequence and frees the bus if another bus error occurs.

```
                                367 * register contents are not modified.
                                368 * MASTER TRANSMITTER TEST BUS STATUS SUBROUTINE:
                                369 *
00015A 0D               55      370 MTTBS   MOV     A,S1            FETCH BUS STATUS
00015B F25E             56      371         JB7     MTTBS1          JUMP IF MST = 1
00015D 83               57      372         RET
                                373 *
00015E 925A             58      374 MTTBS1  JB4     MTTBS           JUMP IF PIN = 1
000160 D3E0             59      375         XRL     A,#MTTST        TEST MST/TRX BUS STATUS
000162 83               60      376         RET
                                377 *
000163                          378         END
```

3.1.2 Master receiver routine (MAB84X1 - SAB3028)

These routines can only be used if no other masters are connected to the I^2C-bus.

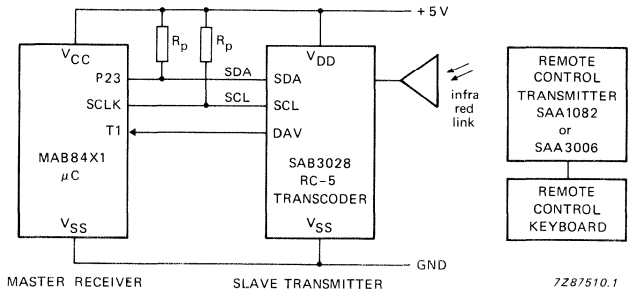


Fig. 3.8 Block diagram of MAB84X1 - SAB3028 configuration.

Initialization:

```
                                11 * SYMBOL DEFINITION:
                                12 *
                                13 IICFR  EQU     H'04'          Fscl =  95 kHz @Fxtal = 6,00 MHz
                                14 *
                                15 MRDTR1 EQU     H'20'          MASTER RECEIVER DATA BYTE 1 REGISTER
                                16 MRDTR2 EQU     MRDTR1+1                  ''              2     ''
                                17 MRDTR3 EQU     MRDTR2+1                  ''              3     ''
                                18 MRDTR4 EQU     MRDTR3+1                  ''              4     ''
                                19 *
                                20 * POWER-ON RESET:
                                21 *
                                22        PAGE    256
                                23 ROM0   ASECT   ROM
000000                          24        ORG     H'000'
                                25 *
000000 2400          1          26 RESET  JMP     INIT           JUMP TO INITIALIZE ROUTINE
                                27 *
                                28 * INITIALISATION:
                                29 *
000002                          30        ORG     H'100'
                                31 *
000100 9E44          2          32 INIT   MOV     S2,#IICFR+ACK  LOAD SCL FREQUENCY WORD
                                33 *                             SET WITH ACKNOWLEDGE MODE
000102 9D18          3          34        MOV     S1,#PIN+ESO    ENABLE SIO
                                35 *                             SET SLAVE RECEIVER MODE
```

Main program:

The main program polls the input T1, which is connected to the data valid output (DAV) of the RC-5 transcoder SAB3028. This output goes low if the RC-5 transcoder has received a valid remote control message. If it is low, the main program calls subroutine RRCT, which reads the data out of the RC-5 transcoder via the I^2C-bus.
The subroutine RRCT returns with Accu = 0 if the data transfer is successful.

The received data is stored in the registers MRDTR1-4. If the transfer is successful DAV is reset to 1.

If the data transfer is not successful ($A \neq 0$), the transmission is repeated.

The format of the data transfer from the RC-5 transcoder SAB3028 is shown in Fig. 3.9:

S	RCT ADR. R A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 NA	P

Fig. 3.9 RC-5 transcoder data transfer

S is the START condition
RCT ADR is the slave address of RC-5 transcoder
R is the read/write bit in read state (= 1)
A is the acknowledge bit; this has to be 0
NA is the not acknowledge bit, this has to be 1
P is the STOP condition

```
                          59 * The slave address of RC-5 transcoder is 0100 110.
                          60 RCTAD  EQU     H'4C'
                          61 *
000104 5604           4   62 MAIN   JT1     MAIN            WAIT FOR T1 = DAV = 0
                          63 *
000106 340C           5   64 MAIN1  CALL    RRCT            READ DATA OUT OF RC-5 TRANSCODER
000108 9606           6   65        JNZ     MAIN1           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00010A 2404           7   66        JMP     MAIN            CONTINUE
```

Master reads data out of RC-5 transcoder (SAB3028(subroutine:

This subroutine starts to output the slave address and R/W bit (in read state = 1) to the RC-5 transcoder. If this address is not acknowledged, the subroutine is terminated with a STOP condition and with the accumulator contents equal to H'01'. Otherwise the direction of the data transfer changes and the data is clocked out of the RC-5 transcoder. After each of the first three data bytes the microcomputer gives an acknowledge (a LOW level).

To tell the slave that no more data bytes are requested, the master has to give a not-acknowledge (a HIGH level) after the last received data byte.

At that moment the slave knows that the transfer will be terminated and has to allow the SDA line to go HIGH. Now the master is able to generate a STOP condition.

To generate a not-acknowledge the transmission of the last data word is split up in the reception of 8 bits of data and the generation of a "not-acknowledge" bit. After the reception of the third data byte the acknowledge mode bit in S2 is set in the "without acknowledge" state. Then the computer receives 8 bits of data. If PIN becomes '0' again, the bit-counter is set to 1 and the computer inputs the last bit. In input mode, the SDA output is HIGH, therefore the slave will receive an acknowledge bit which is a 1.

Before generating a STOP condition, the acknowledge mode bit in S2 has to be reset to the "acknowledge" state.

If during the transmission of the slave address a disturbance on the data line causes an error (for example an arbitration lost, start or stop condition or a NO ACK is received), the subroutine jumps to label ERROR whereupon the program tests the ACK bit. Upon the result of this test, the SIO will either output a STOP condition (no acknowledge) or invokes a re-initialize routine to free the SIO interface.

The same happens if, during the reception of data, a disturbance occurs.

exit: A = 0 if the transmission is successful, with the received data stored in registers MRDTR1-4.
A ≠ 0 if the transmission is not successful.

The contents of register R1 is modified.

```
00010C 9D18          8  103 RRCT   MOV     S1,#PIN+ESO      RESET SIO STATUS
00010E 9C4D          9  104        MOV     S0,#RCTAD+RW     LOAD SLAVE ADDRESS AND READ BIT
000110 9DF8         10  105        MOV     S1,#STRTC        OUTPUT START COND, SLAVE ADDRESS AND READ BIT
000112 343D         11  106        CALL    MRTBS            TEST BUS STATUS
000114 9646         12  107        JNZ     ERROR            JUMP IF NO ACKN. RECEIVED OR ERROR
000116 0C           13  108        MOV     A,S0             START RECEPTION OF DATA
000117 B920         14  109        MOV     R1,#MRDTR1       SET INDEX TO DATA REGISTER IN WHICH FIRST
                        110 *                                RECEIVED DATA BYTE HAS TO BE STORED
000119 343D         15  111        CALL    MRTBS            TEST BUS STATUS
00011B 9646         16  112        JNZ     ERROR            JUMP IF ERROR
00011D 343A         17  113        CALL    STORDT           STORE FIRST DATA BYTE AND TEST BUS STATUS
00011F 9646         18  114        JNZ     ERROR            JUMP IF ERROR
000121 343A         19  115        CALL    STORDT           STORE SECOND DATA BYTE AND TEST BUS STATUS
000123 9646         20  116        JNZ     ERROR            JUMP IF ERROR
000125 9E04         21  117        MOV     S2,#IICFR        SET WITHOUT ACKNOWLEDGE MODE
000127 343A         22  118        CALL    STORDT           STORE THIRD DATA BYTE AND TEST BUS STATUS
000129 53FE         23  119        ANL     A,#.NOT.LRB      CLEAR LRB
00012B 9633         24  120        JNZ     RRCT1            JUMP IF ERROR
00012D 9DA9         25  121        MOV     S1,#MST+BB+ESO+BCO SET BIT COUNTER TO ONE
00012F 343A         26  122        CALL    STORDT           STORE FOURTH DATA BYTE AND TEST BUS STATUS
                        123 *                                GENERATE NOT ACKNOWLEDGE
000131 D301         27  124        XRL     A,#LRB           INVERT NOT ACKNOWLEDGE BIT
                        125 *
000133 9E44         28  126 RRCT1  MOV     S2,#IICFR+ACK    SET WITH ACKNOWLEDGE MODE
000135 9646         29  127        JNZ     ERROR            ESCAPE & RESET BUS H/W
                        128 *
000137 9DD8         30  129        MOV     S1,#STOPC        OUTPUT STOP CONDITION
000139 83           31  130        RET
                        131 *
                        132 * STORE DATA AND TEST MASTER RECEIVER BUS STATUS SUBROUTINE
                        133 *
00013A 0C           32  134 STORDT MOV     A,S0             FETCH DATA OUT OF SIO DATA REGISTER
00013B A1           33  135        MOV     @R1,A            STORE DATA IN MST/REC DATA REGISTER
00013C 19           34  136        INC     R1               INCR. INDEX OF DATA REGISTER
                        137 *
                        138 * MASTER RECEIVER TEST BUS STATUS SUBROUTINE
                        139 *
00013D 0D           35  140 MRTBS  MOV     A,S1             FETCH BUS STATUS
00013E F241         36  141        JB7     MRTBS1           JUMP IF MST = 1
000140 83           37  142        RET
                        143 *
000141 923D         38  144 MRTBS1 JB4     MRTBS            JUMP IF PIN = 1
000143 D3A0         39  145        XRL     A,#MRTST         TEST MST/REC BUS STATUS
000145 83           40  146        RET
                        147 *
                        148 * NO ACKNOWLEDGE RECEIVED OR ERROR EXIT ROUTINE
                        149 *
000146 D301         41  150 ERROR  XRL     A,#LRB           INVERT  ACK
000148 C656         42  151        JZ      ERROR1           IF NO ACK. OUTPUT STOP COND.
00014A 9EC3         43  152        MOV     S2,#IICFR
00014C 9E44         44  153        MOV     S2,#IICFR+ACK
00014E 9D18         45  154        MOV     S1,#PIN+ESO      RESET BUS STATUS TWICE IF ERROR FEARED
000150 9D18         46  155        MOV     S1,#PIN+ESO      RETURN TO SLAVE MODE
000152 0C           47  156        MOV     A,S0             DUMMY READ
000153 2302         48  157        MOV     A,#2             TO RETURN WITH A ≠ 0
000155 83           49  158        RET
000156 2301         50  159 ERROR1 MOV     A,#1             TO RETURN WITH A = 1
000158 9DD8         51  160        MOV     S1,#STOPC
00015A 83           52  161        RET
                        162 *
00015B                  163        END
```

3.1.3 Master transmitter/receiver routine including general call reset (MAB84X1 - SAB3035/36/37)

These routines can only be used if no other masters are connected to the I^2C-bus.

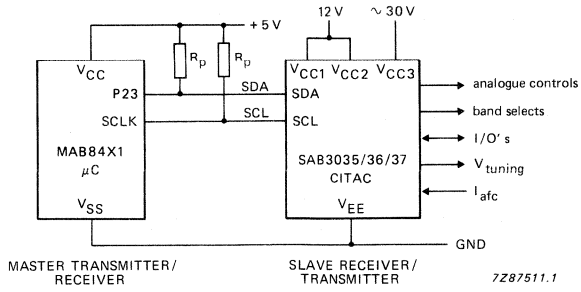


Fig. 3.10 Block diagram MAB84X1 - SAB3035/36/37 CITAC configuration.

Initialization:

```
                                13 * SYMBOL DEFINITION:
                                14 *
                                15 IICFR  EQU     H'04'           Fscl =  95 kHz @Fxtal = 6,00 MHz
                                16 *
                                17 * POWER-ON RESET:
                                18 *
                                19        PAGE    256
                                20 ROM0   ASECT   ROM
000000                          21        ORG     H'000'
                                22 *
000000 2400          1          23 RESET  JMP     INIT            JUMP TO INITIALIZE ROUTINE
                                24 *
                                25 * INITIALISATION:
                                26 *
000002                          27        ORG     H'100'
                                28 *
000100 9E44          2          29 INIT   MOV     S2,#IICFR+ACK   LOAD SCL FREQUENCY WORD
                                30 *                              SET WITH ACKNOWLEDGE MODE
000102 9D18          3          31        MOV     S1,#PIN+ESO     ENABLE SIO
                                32 *                              SET SLAVE RECEIVER MODE
000104 BB00          4          33        MOV     R3,#00          SET DATA BYTE 2; DATA OF ANALOGUE REGISTER 4
```

Main program:

The main program calls the three following I^2C-bus transmission subroutines:

- subroutine GCCIT which resets the CITAC (SAB3035/36/37).
  The format of this transmission shown in Fig. 3.11:

S	GC ADR	A	RESET CODE	A	P

Fig. 3.11 Data format of transfer to reset CITAC.

- subroutine WCIT which writes two bytes of data to the CITAC.
  The format is shown in Fig. 3.12:

S	CIT. ADR.	W	A	DATA 1	A	DATA 2	A	P

Fig. 3.12 Data format of data write to CITAC.

- subroutine RCIT which reads two data bytes out of the CITAC.
  The format is shown in Fig. 3.13:

S	CIT. ADR. R A	DATA 1 A	DATA 2	NA	P

Fig. 3.13 Data format of data read from CITAC.

S	is the START condition
CIT ADR	is the slave address of CITAC
GC ADR	is the general call address (H'00')
RESET CODE	is defined as H'06'
W	is the read/write bit in write state (= 0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

If one of the transmissions does not succeed (A ≠ 0), it is repeated.

```
                          64 CITAD  EQU     H'C0'
                          65 *
                          66 * The general call address is:
                          67 GCAD   EQU     H'00'
                          68 *
                          69 *
000106 3417          5    70 MAIN   CALL    GCCIT           GENERAL CALL RESET TO CITAC
000108 9606          6    71        JNZ     MAIN            REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00010A BA24          7    72        MOV     R2,#H'24'       SET DATA BYTE 1; SUBADDRESS OF ANALOGUE 4
00010C 1B            8    73        INC     R3              INCR. DATA BYTE 2 TO BE WRITTEN TO CITAC
                          74 *
00010D 342A          9    75 MAIN1  CALL    WCIT            WRITE DATA OF R2 AND R3 TO CITAC
00010F 960D         10    76        JNZ     MAIN1           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
                          77 *
000111 3443         11    78 MAIN2  CALL    RCIT            READ DATA OUT OF CITAC AND STORE IN R4 AND R5
000113 9611         12    79        JNZ     MAIN2           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000115 2406         13    80        JMP     MAIN            CONTINUE
                          81 *
```

Masters writes general call reset code to CITAC (SAB3035/36/37) subroutine:

exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful;
register contents are not modified.

```
000117 9D18         14    89 GCCIT  MOV     S1,#PIN+ESO     RESET BUS STATUS
000119 9C00         15    90        MOV     S0,#GCAD        LOAD GENERAL CALL ADDRESS
00011B 9DF8         16    91        MOV     S1,#STRTC       OUTPUT START COND. AND GENERAL CALL ADDRESS
00011D 346B         17    92        CALL    MTTBS           TEST BUS STATUS
00011F 967D         18    93        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
000121 9C06         19    94        MOV     S0,#H'06'       TRANSMIT RESET CODE
000123 346B         20    95        CALL    MTTBS           TEST BUS STATUS
000125 967D         21    96        JNZ     ERROR           JUMP IF NO ACK. RECEIVED OR ERROR
000127 9DD8         22    97        MOV     S1,#STOPC       OUTPUT STOP CONDITION
000129 83           23    98        RET
```

Master writes two data bytes to CITAC (SAB3035/36/37) subroutine:

```
entry: R2 contains first data byte to be transmitted
       R3 contains second data byte
exit: A = 0 if transmission is successful
      A ≠ 0 if transmission is not successful;
      register contents are not modified.
```

```
00012A 9D18          24    108 WCIT   MOV     S1,#PIN+ESO     RESET BUS STATUS
00012C 9CC0          25    109        MOV     S0,#CITAD       LOAD CITAC SLAVE ADDRESS AND WRITE BIT

00012E 9DF8          26    110        MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000130 346B          27    111        CALL    MTTBS           TEST BUS STATUS
000132 967D          28    112        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
000134 FA            29    113        MOV     A,R2            FETCH FIRST DATA BYTE
000135 3C            30    114        MOV     S0,A            TRANSMIT DATA BYTE 1
000136 346B          31    115        CALL    MTTBS           TEST BUS STATUS
000138 967D          32    116        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
00013A FB            33    117        MOV     A,R3            FETCH SECOND DATA BYTE
00013B 3C            34    118        MOV     S0,A            TRANSMIT DATA BYTE 2
00013C 346B          35    119        CALL    MTTBS           TEST BUS STATUS
00013E 967D          36    120        JNZ     ERROR           JUMP IF NO ACK.RECEIVED OR ERROR
000140 9DD8          37    121        MOV     S1,#STOPC       OUTPUT STOP CONDITION
000142 83            38    122        RET
```

Master reads two data bytes out of CITAC (SAB3035/36/37) subroutine:

```
exit: A = 0 if transmission is successful and received data is stored in
        registers R4 and R5.
      A ≠ 0 if transmission is not successful.
Register R4 and R5 contents are modified.
```

```
000143 9D18          39    132 RCIT   MOV     S1,#PIN+ESO     RESET SIO STATUS
000145 9CC1          40    133        MOV     S0,#CITAD+RW    LOAD SLAVE ADDRESS AND READ BIT
000147 9DF8          41    134        MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND READ BIT
000149 3474          42    135        CALL    MRTBS           TEST BUS STATUS
00014B 967D          43    136        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
00014D 0C            44    137        MOV     A,S0            START RECEPTION OF DATA
00014E 3474          45    138        CALL    MRTBS           TEST BUS STATUS
000150 967D          46    139        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
000152 9E04          47    140        MOV     S2,#IICFR       SET WITHOUT ACKNOWLEDGE MODE
000154 0C            48    141        MOV     A,S0            FETCH FIRST RECEIVED DATA BYTE
000155 AC            49    142        MOV     R4,A            SAVE
000156 3474          50    143        CALL    MRTBS           TEST BUS STATUS
000158 53FE          51    144        ANL     A,#.NOT.LRB     CLEAR LRB
00015A 9664          52    145        JNZ     RCIT1           JUMP IF ERROR
00015C 9DA9          53    146        MOV     S1,#MST+BB+ESO+BCO SET BIT COUNTER TO ONE
00015E 0C            54    147        MOV     A,S0            FETCH SECOND DATA BYTE
00015F AD            55    148        MOV     R5,A            SAVE DATA BYTE
000160 3474          56    149        CALL    MRTBS           TEST BUS STATUS
                           150 *                              GENERATE NOT ACKNOWLEDGE
000162 D301          57    151        XRL     A,#LRB          INVERT NOT ACKNOWLEDGE BIT
                           152 *
000164 9E44          58    153 RCIT1  MOV     S2,#IICFR+ACK   SET WITH ACKNOWLEDGE MODE
000166 967D          59    154        JNZ     ERROR           JUMP IF ERROR
                           155 *
000168 9DD8          60    156        MOV     S1,#STOPC       OUTPUT STOP CONDITION
00016A 83            61    157        RET
```

Master transmitter test bus status subroutine:

```
                           159 *
                           160 *
00016B 0D            62    161 MTTBS  MOV     A,S1            FETCH BUS STATUS
00016C F26F          63    162        JB7     MTTBS1          JUMP IF MST = 1
00016E 83            64    163        RET
                           164 *
00016F 926B          65    165 MTTBS1 JB4     MTTBS           JUMP IF PIN = 1
000171 D3E0          66    166        XRL     A,#MTTST        TEST MST/TRX BUS STATUS
000173 83            67    167        RET
                           168 *
```

Master receiver test bus status subroutine:

```
                                169 *
                                170 *
000174 0D               68      171 MRTBS   MOV     A,S1            FETCH BUS STATUS
000175 F278             69      172         JB7     MRTBS1          JUMP IF MST = 1
000177 83               70      173         RET
                                174 *
000178 9274             71      175 MRTBS1  JB4     MRTBS           JUMP IF PIN = 1
00017A D3A0             72      176         XRL     A,#MRTST        TEST MST/REC BUS STATUS
00017C 83               73      177         RET
                                178 *
                                179 * NO ACKNOWLEDGE RECEIVED OR ERROR EXIT ROUTINE
                                180 *
00017D D301             74      181 ERROR   XRL     A,#LRB          INVERT  ACK
00017F C68D             75      182         JZ      ERROR1          IF NO ACK. OUTPUT STOP COND.
000181 9E04             76      183         MOV     S2,#IICFR
000183 9E44             77      184         MOV     S2,#IICFR+ACK
000185 9D18             78      185         MOV     S1,#PIN+ESO     RESET BUS STATUS TWICE IF ERROR FEARED
000187 9D18             79      186         MOV     S1,#PIN+ESO     RETURN TO SLAVE MODE
000189 0C               80      187         MOV     A,S0            DUMMY READ
00018A 2302             81      188         MOV     A,#2            TO RETURN WITH A ≠ 0
00018C 83               82      189         RET
00018D 2301             83      190 ERROR1  MOV     A,#1            TO RETURN WITH A = 1
00018F 9DD8             84      191         MOV     S1,#STOPC       OUTPUT STOP CONDITION
000191 83               85      192         RET
                                193 *
000192                          194         END
```

3.1.4 Master transmitter/receiver routine with repeated start (MAB84X1 - PCD8571)

These routines can only be used if no other masters are connected to the I^2C-bus.

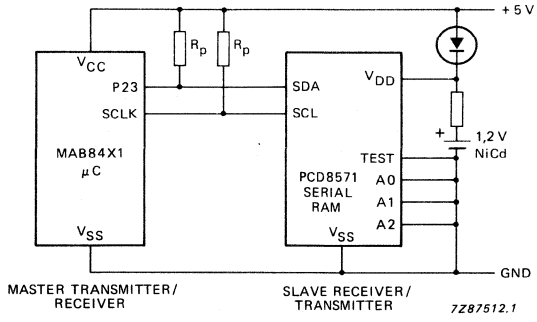


Fig. 3.14 Block diagram MAB84X1 - PCD8571 serial RAM configuration.

Initialization:

```
                                13 * SYMBOL DEFINITION:
                                14 *
                                15 IICFR  EQU     H'04'          Fscl =  95 kHz @Fxtal = 6,00 MHz
                                16 *
                                17 * POWER-ON RESET:
                                18 *
                                19        PAGE    256
                                20 ROM0   ASECT   ROM
000000                          21        ORG     H'000'
                                22 *
000000 2400           1         23 RESET  JMP     INIT           JUMP TO INITIALIZE ROUTINE
                                24 *
                                25 * INITIALISATION:
                                26 *
000002                          27        ORG     H'100'
                                28 *
000100 9E44           2         29 INIT   MOV     S2,#IICFR+ACK  LOAD SCL FREQUENCY WORD
                                30 *                             SET WITH ACKNOWLEDGE MODE
000102 9D18           3         31        MOV     S1,#PIN+ESO    ENABLE SIO
                                32 *                             SET SLAVE RECEIVER MODE
000104 23FF           4         33        MOV     A,#H'FF'
000106 AD             5         34        MOV     R5,A           SET POINTER VALUE
000107 A9             6         35        MOV     R1,A           SET DATA BYTE 1 TO BE WRITTEN
000108 AA             7         36        MOV     R2,A                    ,,    2       ,,
                                37 *
```

Main program:

The main program calls the two following I^2C-bus transmission subroutines:

- subroutine WMEM which writes the pointer value and two data bytes to the PCD8571 CMOS memory.
  The format of this transmission is shown in Fig. 3.15:

S	MEM. ADR. W A	POINT A	DATA 1 A	DATA 2 A	P

Fig. 3.15 Format to write pointer value and two data bytes to the PCD8571.

- subroutine RMEM which writes the pointer value to the PCD8571 memory and reads two data bytes out of it.
  The format of this transmission is shown in Fig. 3.16:

S	MEM. ADR. W A	POINT A	S MEM. ADR. R A	DATA 1 A	DATA 2 NA	P

Fig. 3.16 Format to write pointer value and read two data bytes.

S	is the START condition
MEM ADR	is the slave address of the CMOS memory PCD8571
POINT	is the pointer address (internal RAM location address)
W	is the read/write bit in write state (= 0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit, this has to be 1
P	is the STOP condition

If one of the transmissions does not succeed (A ≠ 0), it is repeated.

```
                                65 * The slave address of the CMOS memory is 1010 XXX.
                                66 MEMAD  EQU     H'A0'
                                67 *
000109 1D                8      68 MAIN   INC     R5              INCR. POINTER VALUE
00010A 2301              9      69        MOV     A,#1            INCR. DATA TO BE WRITTEN
00010C 6A               10      70        ADD     A,R2             R1,R2 + 1 --> R1,R2
00010D AA               11      71        MOV     R2,A
00010E 27               12      72        CLR     A
00010F 79               13      73        ADDC    A,R1
000110 A9               14      74        MOV     R1,A
                                75 *
000111 341B             15      76 MAIN1  CALL    WMEM            WRITE DATA TO CMOS MEMORY
000113 9611             16      77        JNZ     MAIN1           REPEAT IF TRANSMISSION NOT SUCCESSFUL
                                78 *
000115 3443             17      79 MAIN2  CALL    RMEM            READ DATA OUT OF CMOS MEMORY
000117 9615             18      80        JNZ     MAIN2           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000119 2409             19      81        JMP     MAIN            CONTINUE
```

Master writes two data bytes to CMOS memory (PCD8571) subroutine:

To write data to the memory, the memory needs to know at which address (location) this data has to be stored. The CMOS memory PCD8571 is designed in such a way that the first data word after the slave address and write bit is always the internal address or pointer.

If more data bytes are written into the memory, during one transmission, this pointer is automatically incremented.

entry: R5 contains the pointer value to be transmitted
R1 contains the first data byte which has to be written in the memory
R2 contains the second data byte which has to be written in the memory

exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful;
register contents are not modified.

```
                        97 *
00011B 9D18      20     98 WMEM   MOV     S1,#PIN+ESO     RESET BUS STATUS
00011D 9CA0      21     99        MOV     S0,#MEMAD       LOAD MEMORY SLAVE ADDRESS AND WRITE BIT
00011F 9DF8      22    100        MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000121 343A      23    101        CALL    MTTBS           TEST BUS STATUS
000123 968F      24    102        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
000125 FD        25    103        MOV     A,R5            FETCH POINTER VALUE
000126 3C        26    104        MOV     S0,A            TRANSMIT POINTER VALUE
000127 343A      27    105        CALL    MTTBS           TEST BUS STATUS
000129 968F      28    106        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
00012B F9        29    107        MOV     A,R1            FETCH FIRST DATA BYTE
00012C 3C        30    108        MOV     S0,A            TRANSMIT DATA BYTE 1
00012D 343A      31    109        CALL    MTTBS           TEST BUS STATUS

00012F 968F      32    110        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
000131 FA        33    111        MOV     A,R2            FETCH SECOND DATA BYTE
000132 3C        34    112        MOV     S0,A            TRANSMIT DATA BYTE 2
000133 343A      35    113        CALL    MTTBS           TEST BUS STATUS
000135 968F      36    114        JNZ     ERROR           JUMP IF NO ACKN. RECEIVED OR ERROR
                       115 *
000137 9DD8      37    116 ERROR  MOV     S1,#STOPC       OUTPUT STOP CONDITION
000139 83        38    117        RET
                       118 *
```

Master transmitter test bus status subroutine:

```
00013A 0D        39    121 MTTBS  MOV     A,S1            FETCH BUS STATUS
00013B F23E      40    122        JB7     MTTBS1          JUMP IF MST = 1
00013D 83        41    123        RET
                       124 *
00013E 923A      42    125 MTTBS1 JB4     MTTBS           JUMP IF PIN = 1
000140 D3E0      43    126        XRL     A,#MTTST        TEST MST/TRX BUS STATUS
000142 83        44    127        RET
```

Master reads two data bytes out of CMOS memory (PCD8571) subroutine:

When the master wants to read data out of the memory, it first has to write the pointer into the memory. Since changing of the data direction is only possible after the R/W bit of the slave address, the master has to repeat the START condition, the slave address and the R/W bit (in read state) to change the direction within one transmission.

In the subroutine given below, the MAB84X1 family releases the bus with the MOV S1,#PIN + ESO instruction. Now SDA goes high followed by SCL, if all the slaves release the SCL line and no STOP condition is generated.

If SCL goes high the information on the SDA line is latched into the LRB flag of the serial I/O status register S1. It is possible that slaves can keep the SCL line low, therefore the MAB84X1 tests if the SCL line is released by all slaves connected to the bus. This is done by testing the status bits BB and LRB (see Fig. 3.17).

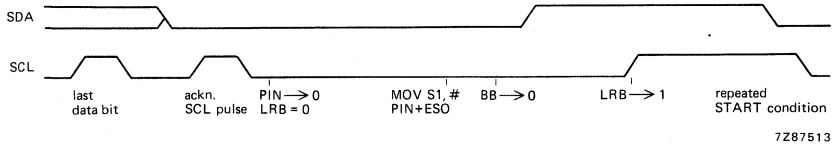


Fig. 3.17 Testing status bits BB and LRB to see whether the SCL line has been released.

As Fig. 3.17 shows, if BB has become 0 and LRB has become 1, a new start condition can be generated.

entry: R5 contains pointer value to be written to the memory.
exit: A = 0 if the transmission is successful and the received data is stored in registers R3 and R4.
A ≠ 0 if the transmission is not successful.
Register R0, R3 and R4 contents are modified.

```
000143 9D18          45    162 RMEM   MOV     S1,#PIN+ESO      RESET SIO STATUS
000145 9CA0          46    163        MOV     S0,#MEMAD        LOAD SLAVE ADDRESS AND WRITE BIT
000147 9DF8          47    164        MOV     S1,#STRTC        OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000149 343A          48    165        CALL    MTTBS            TEST BUS STATUS

00014B 968F          49    166        JNZ     ERROR            JUMP IF NO ACKN. RECEIVED OR ERROR
00014D FD            50    167        MOV     A,R5             FETCH POINTER VALUE
00014E 3C            51    168        MOV     S0,A             TRANSMIT POINTER VALUE
00014F 343A          52    169        CALL    MTTBS            TEST BUS STATUS
000151 968F          53    170        JNZ     ERROR            JUMP IF NO ACKN. RECEIVED OR ERROR
000153 9D18          54    171        MOV     S1,#PIN+ESO      RELEASE SDA AND SCL LINE
                           172 *                               NO STOP CONDITION
000155 B80A          55    173        MOV     R0,#10           LOAD REPEATED START COUNTER
                           174 *
000157 0D            56    175 RPSTR1 MOV     A,S1             FETCH BUS STATUS
000158 B25C          57    176        JB5     RPSTR2           JUMP IF BB = 1
00015A 1260          58    177        JB0     RPSTR3           JUMP IF BB = 0 AND LRB = 1
                           178 *                                SCL HAS BECOME HIGH
00015C E857          59    179 RPSTR2 DJNZ    R0,RPSTR1        DECR. AND TEST REP. START COUNTER
00015E 248F          60    180        JMP     ERROR            JUMP IF ERROR
                           181 *
000160 9CA1          61    182 RPSTR3 MOV     S0,#MEMAD+RW     LOAD SLAVE ADDRESS AND READ BIT
000162 9DF8          62    183        MOV     S1,#STRTC        OUTPUT START COND, SLAVE ADDRESS AND READ BIT
000164 3486          63    184        CALL    MRTBS            TEST BUS STATUS
000166 968F          64    185        JNZ     ERROR            JUMP IF NO ACKN. RECEIVED OR ERROR
000168 0C            65    186        MOV     A,S0             START RECEPTION OF DATA
000169 3486          66    187        CALL    MRTBS            TEST BUS STATUS
00016B 968F          67    188        JNZ     ERROR            JUMP IF ERROR
00016D 9E04          68    189        MOV     S2,#IICFR        SET WITHOUT ACKNOWLEDGE MODE
00016F 0C            69    190        MOV     A,S0             FETCH FIRST RECEIVED DATA BYTE
000170 AB            70    191        MOV     R3,A             SAVE
000171 3486          71    192        CALL    MRTBS            TEST BUS STATUS
000173 53FE          72    193        ANL     A,#.NOT.LRB      CLEAR LRB
000175 967F          73    194        JNZ     RMEM1            JUMP IF ERROR
000177 9DA9          74    195        MOV     S1,#MST+BB+ESO+BCO SET BIT COUNTER TO ONE
000179 0C            75    196        MOV     A,S0             FETCH SECOND DATA BYTE
00017A AC            76    197        MOV     R4,A             SAVE DATA BYTE
00017B 3486          77    198        CALL    MRTBS            TEST BUS STATUS
                           199 *                               GENERATE NOT ACKNOWLEDGE
00017D D301          78    200        XRL     A,#LRB           INVERT NOT ACKNOWLEDGE BIT
                           201 *
00017F 9E44          79    202 RMEM1  MOV     S2,#IICFR+ACK    SET WITH ACKNOWLEDGE MODE
000181 968F          80    203        JNZ     ERROR            RESET BUS H/W & ESCAPE
000183 9DD8          81    204        MOV     S1,#STOPC        OUTPUT STOP CONDITION
000185 83            82    205        RET
                           206 *
```

Master receiver test bus status subroutine:

```
                           208 *
000186 0D            83    209 MRTBS  MOV     A,S1             FETCH BUS STATUS
000187 F28A          84    210        JB7     MRTBS1           JUMP IF MST = 1
000189 83            85    211        RET
                           212 *
00018A 9286          86    213 MRTBS1 JB4     MRTBS            JUMP IF PIN = 1
00018C D3A0          87    214        XRL     A,#MRTST         TEST MST/REC BUS STATUS
00018E 83            88    215        RET
                           216 *
                           217 * NO ACKNOWLEDGE RECEIVED OR ERROR EXIT ROUTINE
                           218 *
00018F D301          89    219 ERROR  XRL     A,#LRB           INVERT  ACK
000191 C69F          90    220        JZ      ERROR1           IF NOT.ACK OUTPUT STOP COND.
000193 9EC3          91    221        MOV     S2,#IICFR+.NOT.ACK

000195 9E44          92    222        MOV     S2,#IICFR+ACK
000197 9D18          93    223        MOV     S1,#PIN+ESO      RESET BUS STATUS TWICE IF ERROR FEARED
000199 9D18          94    224        MOV     S1,#PIN+ESO      RETURN TO SLAVE MODE
00019B 0C            95    225        MOV     A,S0             DUMMY READ
00019C 2302          96    226        MOV     A,#2             TO RETURN WITH A ≠ 0
00019E 83            97    227        RET
00019F 2301          98    228 ERROR1 MOV     A,#1             TO RETURN WITH A = 1
0001A1 9DD8          99    229        MOV     S1,#STOPC        OUTPUT STOP CONDITION
0001A3 83           100    230        RET
                           231 *
0001A4                     232        END
```

3.1.5 Master transmitter routine - CBUS (MAB84X1 - SAA1061)

These routines can only be used if no other masters are connected to the I^2C-bus.

CBUS devices are compatible with the I^2C-bus. The CBUS was formerly used in microcomputer-based systems such as Video Tuning Systems (VTS) or Radio Tuning Systems (RTS). The master transmitter for the CBUS is always the microcomputer. The CBUS devices are all slave receivers.

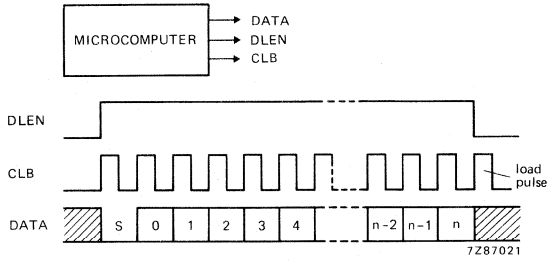


Fig. 3.18 The CBUS structure.

Characteristics of the CBUS:

- one clock pulse per bit
- clock frequency variable over a wide range
- data bit length fixed for each peripheral
- clock burst of (n + 1) pulses or continuously-running clock
- one start bit (S)
- one load pulse
- chip addressing according to the duration of the DLEN signal.

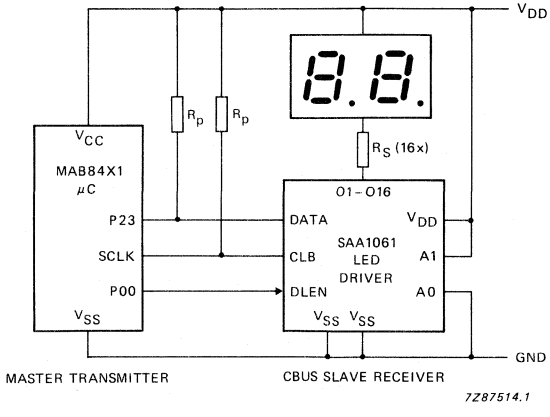


Fig. 3.19 Block diagram MAB84X1 - SAA1061 LED driver configuration.

Initialization:

```
                                33 *
                                34 * SYMBOL DEFINITION:
                                35 *
                                36 IICFR   EQU     H'04'           Fscl =  95 kHz @Fxtal = 6,00 MHz
                                37 DLEN    EQU     H'01'           P00 IS DLEN OUTPUT
                                38 *
                                39 * POWER-ON RESET:
                                40 *
                                41         PAGE    256
                                42 ROM0    ASECT   ROM
000000                          43         ORG     H'000'
                                44 *
000000 2400           1         45 RESET   JMP     INIT            JUMP TO INITIALIZE ROUTINE
                                46 *
                                47 * INITIALISATION:
                                48 *
000002                          49         ORG     H'100'
                                50 *
000100 9E44           2         51 INIT    MOV     S2,#IICFR+ACK   LOAD SCL FREQUENCY WORD
                                52 *                               SET WITH ACKNOWLEDGE MODE
000102 9D18           3         53         MOV     S1,#PIN+ESO     ENABLE SIO
```

Main program:

The main program increments the data word to be displayed and calls the subroutine CBUS, which converts the display word to 7 segment code and writes this code via the I^2C-bus to the SAA1061 LED display driver.

The format of the data transfer is shown in Fig. 3.20:

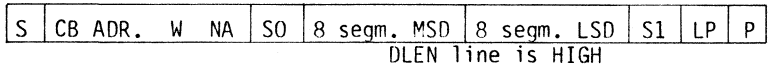


Fig. 3.20 CBUS format for writing to an LED display driver.

S	is the START condition
CB ADR	is the CBUS slave address (H'02')
W	is the read/write bit in write state (= 0)
NA	is the not-acknowledge bit; this has to be 1
SO	is the device select bit
S1	is the device select bit
	a device is selected if SO = A0 and if S1 = A1
8 segm.	are the segments a, b, c, d, e, f, g, dp.
MSD	is the most significant digit
LSD	is the least significant digit
LP	is the CBUS load pulse
P	is the STOP condition

```
                                82 CBAD    EQU     H'02'
                                83 *
000108 BB40           6         84 MAIN    MOV     R3,#H'40'       SELECT BITS SO=0, S1=1
00010A 1A             7         85         INC     R2              INCR. DISPLAY WORD
                                86 *
00010B 3411           8         87 MAIN1   CALL    CBUS1           CONVERT DATA AND WRITE 7 SEGM. CODE TO
                                88 *                                DISPLAY DRIVER
                                89 *
00010D 960B           9         90         JNZ     MAIN1           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00010F 2408          10         91         JMP     MAIN            CONTINUE
```

Master transmitter CBUS format to display driver (SAA1060/61/63) subroutine 1:

To transmit data to a CBUS receiver, the CBUS slave address H'02' first has to be transmitted. At the reception of this address, none of the connected I^2C-bus devices are allowed to generate an acknowledge.

After this, the real CBUS data transfer can start by setting the DLEN line to high and generating the number of data bits requested by the CBUS device. SAA1061 needs two device select bits and two eight bit data bytes within the DLEN high pulse. The transmission of this data has to be done in the "no acknowledge" mode.

After resetting the DLEN line the master generates a load pulse to enable the slave receiver to latch the received data into its output register.

entry: R2 contains hex code which has to be converted to 7 segment code
and to be transmitted to the display driver.
R3 contains the device select bits; format is S0 S1 X X X X X X
exit: A = 0 if the transmission is successful.
A ≠ 0 if the transmission is not successful.
Register contents are not modified.

```
                          113 *
000111 9D18          11   114 CBUS1  MOV     S1,#PIN+ESO      RESET SIO BUS STATUS
000113 9C02          12   115         MOV     S0,#CBAD         LOAD CBUS SLAVE ADDRESS AND WRITE BIT
000115 9DF8          13   116         MOV     S1,#STRTC        OUTPUT START COND, CBUS SLAVE ADDRESS AND
                          117 *                                 WRITE BIT
000117 344F          14   118         CALL    CBTBS1           TEST BUS STATUS
000119 965A          15   119         JNZ     ERROR            JUMP IF ERROR
00011B 9E04          16   120         MOV     S2,#IICFR        SET WITHOUT ACKNOWLEDGE MODE
00011D 8801          17   121         ORL     P0,#DLEN         SET DLEN OUTPUT
00011F 9DE9          18   122         MOV     S1,#MST+TRX+BB+ESO+BC0   SET BITCOUNTER TO 1
000121 FB            19   123         MOV     A,R3
000122 344E          20   124         CALL    CBOUT1           TRANSMIT SELECT BIT S0
000124 965A          21   125         JNZ     ERROR            JUMP IF ERROR
000126 FA            22   126         MOV     A,R2
000127 47            23   127         SWAP    A
000128 3449          24   128         CALL    CBCON1           CONVERT AND TRANSMIT MSD
00012A 965A          25   129         JNZ     ERROR            JUMP IF ERROR
00012C FA            26   130         MOV     A,R2
00012D 3449          27   131         CALL    CBCON1           CONVERT AND TRANSMIT LSD
00012F 965A          28   132         JNZ     ERROR            JUMP IF ERROR
000131 9DE9          29   133         MOV     S1,#MST+TRX+BB+ESO+BC0   SET BITCOUNTER TO 1
000133 FB            30   134         MOV     A,R3
000134 E7            31   135         RL      A
000135 344E          32   136         CALL    CBOUT1           TRANSMIT SELECT BIT S1
000137 965A          33   137         JNZ     ERROR            JUMP IF ERROR
000139 98FE          34   138         ANL     P0,#.NOT.DLEN    RESET DLEN OUTPUT
00013B 9DE9          35   139         MOV     S1,#MST+TRX+BB+ESO+BC0   SET BITCOUNTER TO 1
00013D 27            36   140         CLR     A
00013E 344E          37   141         CALL    CBOUT1           TRANSMIT LP
000140 965A          38   142         JNZ     ERROR
                          143 *
000142 9E44          39   144         MOV     S2,#IICFR+ACK    SET WITH ACKNOWLEDGE MODE
000144 98FE          40   145         ANL     P0,#.NOT.DLEN
000146 9DD8          41   146         MOV     S1,#STOPC        OUTPUT STOP CONDITION IF NO ERROR DETECTED
000148 83            42   147         RET
                          148 *
                          149 * HEX TO 7 SEGMENT CONVERSION SUBROUTINE:
                          150 *
000149 530F          43   151 CBCON1 ANL     A,#H'0F'         MASK LSD
00014B 0368          44   152         ADD     A,#.LOW.CONVT1
00014D A3            45   153         MOVP    A,@A             FETCH 7 SEGMENT CODE OUT OF LOOK-UP TABLE
                          154 *
                          155 * CBUS OUTPUT SUBROUTINE:
                          156 *
00014E 3C            46   157 CBOUT1 MOV     S0,A
                          158 *
                          159 * MASTER CBUS TRANSMITTER TEST BUS STATUS SUBROUTINE:
                          160 *
00014F 0D            47   161 CBTBS1 MOV     A,S1             FETCH BUS STATUS
000150 F253          48   162         JB7     CBTB10           JUMP IF MST = 1
000152 83            49   163         RET
                          164 *
000153 924F          50   165 CBTB10 JB4     CBTBS1           JUMP IF PIN = 1
```

```
000155 53FE          51    166          ANL     A,#.NOT.LRB      RESET LRB
000157 D3E0          52    167          XRL     A,#MTTST         TEST MST/TRM BUS STATUS
000159 83            53    168          RET
00015A 9E04          54    169 ERROR    MOV     S2,#IICFR
00015C 9E44          55    170          MOV     S2,#IICFR+ACK
00015E 9D18          56    171          MOV     S1,#PIN+ESO
000160 9D18          57    172          MOV     S1,#PIN+ESO
000162 0C            58    173          MOV     A,S0
000163 2302          59    174          MOV     A,#2
000165 98FE          60    175          ANL     P0,#.NOT.DLEN
000167 83            61    176          RET
                           177 *
                           178 * SEVEN SEGMENT SYMBOL DEFINITION FOR DISPLAY DRIVER SAA1060/61/63
                           179 *
                           180 SA1      EQU     H'80'
                           181 SB1      EQU     H'40'
                           182 SC1      EQU     H'20'
                           183 SD1      EQU     H'10'
                           184 SE1      EQU     H'08'
                           185 SF1      EQU     H'04'
                           186 SG1      EQU     H'02'
                           187 SP1      EQU     H'01'
                           188 *
                           189 * HEX CODE TO 7 SEGM. CODE CONV. TABLE FOR DISPLAY DRIVER SAA1060/61/63
                           190 *
000168 03                  191 CONVT1 DATA     .NOT.(SA1+SB1+SC1+SD1+SE1+SF1)          DIGIT   0
000169 9F                  192        DATA     .NOT.(SB1+SC1)                          ''      1
00016A 25                  193        DATA     .NOT.(SA1+SB1+SD1+SE1+SG1)              ''      2
00016B 0D                  194        DATA     .NOT.(SA1+SB1+SC1+SD1+SG1)              ''      3
00016C 99                  195        DATA     .NOT.(SB1+SC1+SF1+SG1)                  ''      4
00016D 49                  196        DATA     .NOT.(SA1+SC1+SD1+SF1+SG1)              ''      5
00016E 41                  197        DATA     .NOT.(SA1+SC1+SD1+SE1+SF1+SG1)          ''      6
```

Master transmitter CBUS format to display driver (SAA1060/61/63) subroutine 2:

To save some program bytes, the error jumps after each transmission can be deleted. The disadvantage of this method though, is that, if an error occurs, the program continues trying to output data but does not succeed. Whereas in subroutine CBUS1, it jumps out directly which does save some time.

entry: R2 contains hex code which has to be converted to 7 segment code
and to be transmitted to the display driver.
R3 contains the device select bits; format is S0 S1 X X X X X X
exit: A = 0 if the transmission is successful.
A ≠ 0 if the transmission is not successful.
Register contents are not modified.

```
000178 9D18          62    221 CBUS2   MOV     S1,#PIN+ESO     RESET SIO BUS STATUS

00017A 9C02          63    222         MOV     S0,#CBAD        LOAD CBUS SLAVE ADDRESS AND WRITE BIT
00017C 9DF8          64    223         MOV     S1,#STRTC       OUTPUT START COND, CBUS SLAVE ADDRESS AND
                           224 *                                  WRITE BIT
00017E 34AA          65    225         CALL    CBTBS2          TEST BUS STATUS
000180 9E04          66    226         MOV     S2,#IICFR       SET WITHOUT ACKNOWLEDGE MODE
000182 8801          67    227         ORL     P0,#DLEN        SET DLEN OUTPUT
000184 9DE9          68    228         MOV     S1,#MST+TRX+BB+ESO+BC0  SET BITCOUNTER TO 1
000186 FB            69    229         MOV     A,R3
000187 34A9          70    230         CALL    CBOUT2          TRANSMIT SELECT BIT S0
000189 FA            71    231         MOV     A,R2
00018A 47            72    232         SWAP    A
00018B 34A4          73    233         CALL    CBCON2          CONVERT AND TRANSMIT MSD
00018D FA            74    234         MOV     A,R2
00018E 34A4          75    235         CALL    CBCON2          CONVERT AND TRANSMIT LSD
000190 9DE9          76    236         MOV     S1,#MST+TRX+BB+ESO+BC0  SET BITCOUNTER TO 1
000192 FB            77    237         MOV     A,R3
000193 E7            78    238         RL      A
000194 34A9          79    239         CALL    CBOUT2          TRANSMIT SELECT BIT S1
000196 98FE          80    240         ANL     P0,#.NOT.DLEN   RESET DLEN OUTPUT
000198 9DE9          81    241         MOV     S1,#MST+TRX+BB+ESO+BC0  SET BITCOUNTER TO 1
00019A 27            82    242         CLR     A
00019B 34A9          83    243         CALL    CBOUT2          TRANSMIT LP
00019D 9E44          84    244         MOV     S2,#IICFR+ACK   SET WITH ACKNOWLEDGE MODE
00019F D3E0          85    245         XRL     A,#MTTST        TEST BUS STATUS
0001A1 9DD8          86    246         MOV     S1,#STOPC       OUTPUT STOP CONDITION
0001A3 83            87    247         RET
                           248 *
                           249 * HEX TO 7 SEGMENT CONVERSION SUBROUTINE:
                           250 *
0001A4 530F          88    251 CBCON2  ANL     A,#H'0F'        MASK LSD
0001A6 0368          89    252         ADD     A,#.LOW.CONVT1
0001A8 A3            90    253         MOVP    A,@A            FETCH 7 SEGMENT CODE OUT OF LOOK-UP TABLE
                           254 *
                           255 * CBUS OUTPUT SUBROUTINE:
                           256 *
0001A9 3C            91    257 CBOUT2  MOV     S0,A
                           258 *
                           259 * MASTER CBUS TRANSMITTER TEST BUS STATUS SUBROUTINE:
                           260 *
0001AA 0D            92    261 CBTBS2  MOV     A,S1            FETCH BUS STATUS
0001AB F2AE          93    262         JB7     CBTB20          JUMP IF MST = 1
0001AD 83            94    263         RET
                           264 *
0001AE 92AA          95    265 CBTB20  JB4     CBTBS2          JUMP IF PIN = 1
0001B0 83            96    266         RET
                           267 *
0001B1                     268         END
```

3.1.6 Master transmitter routine - CBUS(MAB84X1 - 2 x PCE2111)

These routines can only be used if no other masters are connected to the I^2C-bus.

The principle of this subroutine is as in Section 3.1.5, but it is for a CBUS device which has a different CBUS format.

This routine shows how a number of CBUS transfers can be carried out within one I^2C-bus transmission.

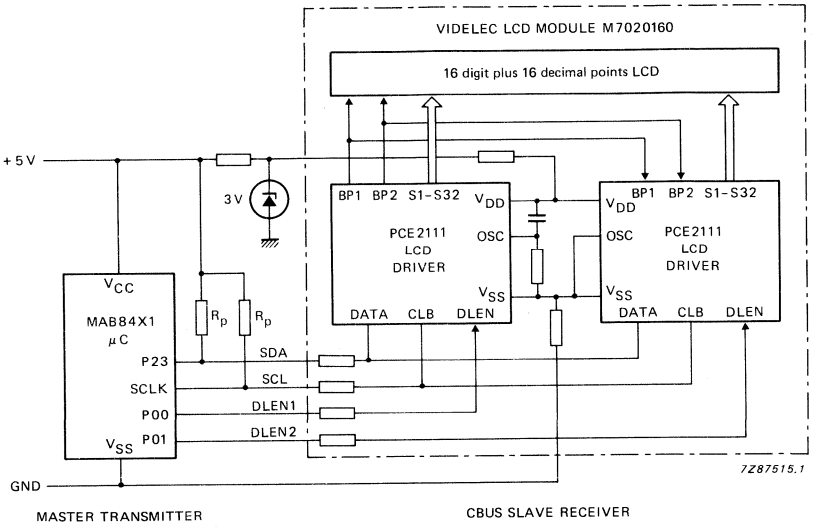


Fig. 3.21 Block diagram MAB84X1 - With two PCE2111 LCD drivers configuration.

Symbol definition:

```
16 IICFR   EQU     H'04'          Fscl = 95 kHz @Fxtal = 6,00 MHz
17 DLEN1   EQU     H'01'          P00 is DLEN1 output
18 DLEN2   EQU     H'02'          P01 is DLEN2 output
19 *
```

Data register definition:

```
20 * DATA REGISTER DEFINITION
21 CBDTR0  EQU     H'20'          DATA REGISTER, TWO MSD'S
22 CBDTR1  EQU     CBDTR0+1       ''
23 CBDTR2  EQU     CBDTR1+1       ''
24 CBDTR3  EQU     CBDTR2+1       ''
25 CBDTR4  EQU     CBDTR3+1       ''
26 CBDTR5  EQU     CBDTR4+1       ''
27 CBDTR6  EQU     CBDTR5+1       ''
28 CBDTR7  EQU     CBDTR6+1       DATA REGISTER, TWO LSD'S
```

Initialization:

```
000000                          34          ORG     H'000'
                                35  *
000000 2400              1      36  RESET   JMP     INIT            JUMP TO INITIALIZE ROUTINE
                                37  *
                                38  * INITIALISATION:
                                39  *
000002                          40          ORG     H'100'
                                41  *
000100 9E44              2      42  INIT    MOV     S2,#IICFR+ACK   LOAD SCL FREQUENCY WORD
                                43  *                               SET WITH ACKNOWLEDGE MODE
000102 9D18              3      44          MOV     S1,#PIN+ESO     ENABLE SIO
                                45  *                               SET SLAVE RECEIVER MODE
000104 98FC              4      46          ANL     P0,#.NOT.(DLEN1+DLEN2) RESET DLEN OUTPUTS
000106 B920              5      47          MOV     R1,#CBDTR0
000108 B1FE              6      48          MOV     @R1,#H'FE'      PRESET DATA REGISTERS
00010A 19                7      49          INC     R1
00010B B1DC              8      50          MOV     @R1,#H'DC'
00010D 19                9      51          INC     R1
00010E B1BA             10      52          MOV     @R1,#H'BA'
000110 19               11      53          INC     R1

000111 B198             12      54          MOV     @R1,#H'98'
000113 19               13      55          INC     R1
000114 B176             14      56          MOV     @R1,#H'76'
000116 19               15      57          INC     R1
000117 B154             16      58          MOV     @R1,#H'54'
000119 19               17      59          INC     R1
00011A B132             18      60          MOV     @R1,#H'32'
00011C 19               19      61          INC     R1
00011D B110             20      62          MOV     @R1,#H'10'
```

Main program:

The main program increments the data word to be displayed and calls the subroutine CBUS3, which converts the 16 hex digits to 7 segment code and transmits this 7 segment code to the 16 digit LCD display module MB7020160.

The format of the data transfer is shown in Fig. 3.22:

S	CB ADR. W NA	REG. 1B	REG. 1A	REG. 2B	REG. 2A	P

Fig. 3.22 CBUS data format for 16 digit LCD display module

S is the START condition
CB ADR is the CBUS slave address
W is the read/write bit in write state (= 0)
NA is the not acknowledge bit; this has to be 1
P is the STOP condition

Register 1B is data to register B of PCE2111-1.
Format of register 1B is shown in Fig. 3.23:

LZ	15a	15g	15b	15c	14a	14g	---	9b	9c	8a	8g	8b	8c	RSB	LPS

DLEN 1 is high

Fig. 3.23 Format of register 1B.

Register 1A is data to register A of PCE2111-1.
Format of register 1A is shown in Fig. 3.24:

LZ	15f	15e	15p	15d	14f	14e	---	9p	9d	8f	8e	8p	8d	RSA	LPS

DLEN 1 is high

Fig. 3.24 Format of register 1A.

Register 2B is data to register B of PCE2111-2.
Format of register 2B is shown in Fig. 3.25:

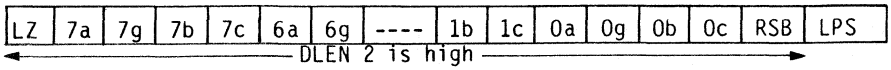


Fig. 3.25 Format of register 2B.

Register 2A is data to register A of PCE2111-2.
Format of register 2A is shown in Fig. 3.26:

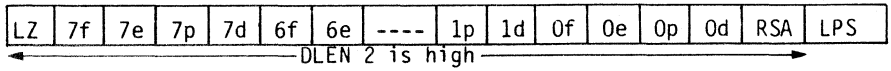


Fig. 3.26 Format of register 2A.

LZ	is the leading zero bit
RSB	is the register B select bit = 0
RSA	is the register A select bit = 1
15a	is the segment a of the MSD; digit 15
0d	is the segment d of the LSD; digit 0
LPS	is the CBUS load pulse

```
                                106 * The CBUS slave address is:
                                107 CBAD    EQU     H'02'
                                108 *
00011F 97              21       109 MAIN    CLR     C                 INCR. DISPLAY WORD

000120 A7              22       110         CPL     C
000121 BA08            23       111         MOV     R2,#8
000123 B927            24       112         MOV     R1,#CBDTR7
                                113 *
000125 27              25       114 MAIN0   CLR     A
000126 71              26       115         ADDC    A,@R1
000127 A1              27       116         MOV     @R1,A
000128 C9              28       117         DEC     R1
000129 EA25            29       118         DJNZ    R2,MAIN0
                                119 *
00012B 3431            30       120 MAIN1   CALL    CBUS3             CONVERT DATA AND WRITE 7 SEGM. CODE TO
                                121 *                                  DISPLAY DRIVER
                                122 *
00012D 962B            31       123         JNZ     MAIN1             REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00012F 241F            32       124         JMP     MAIN              CONTINUE
```

Subroutine in which master converts 16 hex digits to 7 segment code and transmits this code to the two PCE2111 LCD display drivers of the MB7020160 16 digit LCD module:

entry: registers CBDTR0 - CBDTR7 contain the hex digits
register CBDTR0 contains the two MSDs
exit: A = 0 if the transmission is successful
A ≠ 0 if the transmission is not successful

The contents of registers R1 and R2 are modified.

```
                        136 *
000131 9D18       33    137 CBUS3   MOV     S1,#PIN+ESO       RESET BUS STATUS
000133 9C02       34    138         MOV     S0,#CBAD          LOAD CBUS START ADDRESS AND WRITE BIT
000135 9DF8       35    139         MOV     S1,#STRTC         OUTPUT START COND, CBUS SLAVE ADDRESS AND
                        140 *                                 WRITE BIT
000137 544A       36    141         CALL    CBTBS3            TEST BUS STATUS
000139 9E04       37    142         MOV     S2,#IICFR         SET WITHOUT ACKNOWLEDGE MODE
00013B 97         38    143         CLR     C
00013C 5410       39    144         CALL    CBUS30            CONVERT AND OUTPUT SEGMENTS A, B, C AND G OF
                        145 *                                 THE 8 MSD'S, REGISTER 1B
00013E 9654       40    146         JNZ     ERROR             JUMP IF ERROR
000140 A7         41    147         CPL     C
000141 5410       42    148         CALL    CBUS30            CONVERT AND OUTPUT SEGMENTS D, E, F AND P OF
                        149 *                                 THE 8 MSD'S, REGISTER 1A
000143 9654       43    150         JNZ     ERROR             JUMP IF ERROR
000145 97         44    151         CLR     C
000146 5416       45    152         CALL    CBUS31            CONVERT AND OUTPUT SEGMENTS A, B, C AND G OF
                        153 *                                 THE 8 LSD'S, REGISTER 2B
000148 9654       46    154         JNZ     ERROR             JUMP IF ERROR
00014A A7         47    155         CPL     C
00014B 5416       48    156         CALL    CBUS31            CONVERT AND OUTPUT SEGMENTS D, E, F AND P OF
                        157 *                                  THE 8 LSD'S, REGISTER 2A
00014D 9654       49    158         JNZ     ERROR
                        159 *
00014F 9E44       50    160         MOV     S2,#IICFR+ACK     SET WITH ACKNOWLEDGE MODE BIT
000151 9DD8       51    161         MOV     S1,#STOPC
000153 83         52    162         RET
                        163 *
000154 9E04       53    164 ERROR   MOV     S2,#IICFR
000156 9E44       54    165         MOV     S2,#IICFR+ACK
000158 9D18       55    166         MOV     S1,#PIN+ESO
00015A 9D18       56    167         MOV     S1,#PIN+ESO
00015C 0C         57    168         MOV     A,S0
00015D 2302       58    169         MOV     A,#2
00015F 98FC       59    170         ANL     P0,#.NOT.(DLEN1+DLEN2)
000161 83         60    171         RET
                        172 *
                        173 *
000162                  174         ORG     H'200'
                        175 *
                        176 *
```

7 - Segment symbol definition for display driver PCE2111:

```
                        178 *
                        179 SA0     EQU     H'80'
                        180 SB0     EQU     H'20'
                        181 SC0     EQU     H'10'
                        182 SD0     EQU     H'01'
                        183 SE0     EQU     H'04'
                        184 SF0     EQU     H'08'
                        185 SG0     EQU     H'40'
                        186 SP0     EQU     H'02'
                        187 *
```

Hex to 7 - segment code conversion table for LCD display driver PCE2111:

```
                        189 *
                        190 * START ADDRESS OF THIS TABLE HAS TO BE AT H'X00'  !!!!!
                        191 *
000200 BD               192 CONVTO  DATA    SA0+SB0+SC0+SD0+SE0+SF0                    DIGIT   0
000201 30               193         DATA    SB0+SC0                                    ''      1
000202 E5               194         DATA    SA0+SB0+SD0+SE0+SG0                        ''      2
000203 F1               195         DATA    SA0+SB0+SC0+SD0+SG0                        ''      3
000204 78               196         DATA    SB0+SC0+SF0+SG0                            ''      4
000205 D9               197         DATA    SA0+SC0+SD0+SF0+SG0                        ''      5
000206 DD               198         DATA    SA0+SC0+SD0+SE0+SF0+SG0                    ''      6
000207 B0               199         DATA    SA0+SB0+SC0                                ''      7
000208 FD               200         DATA    SA0+SB0+SC0+SD0+SE0+SF0+SG0                ''      8
000209 F9               201         DATA    SA0+SB0+SC0+SD0+SF0+SG0                    ''      9
00020A FE               202         DATA    SA0+SB0+SC0+SE0+SF0+SG0+SP0                ''      A
00020B 5F               203         DATA    SC0+SD0+SE0+SF0+SG0+SP0                    ''      B
00020C 8F               204         DATA    SA0+SD0+SE0+SF0+SP0                        ''      C
00020D 77               205         DATA    SB0+SC0+SD0+SE0+SG0+SP0                    ''      D
00020E CF               206         DATA    SA0+SD0+SE0+SF0+SG0+SP0                    ''      E
00020F CE               207         DATA    SA0+SE0+SF0+SG0+SP0                        ''      F
                        208 *
```

Master transmits data to PCE2111 display driver 1 of the MB7020160 module subroutine:

entry: C = 0; segments a,b,c and g of the 8 MSDs are converted and output
C = 1; segments d,e,f and p of the 8 MSDs are converted and output
registers CBDTR0 - CBDTR3 contain the 8 MSDs.

```
000210 8801       61    216 CBUS30  ORL     P0,#DLEN1         SET DLEN1 OUTPUT
000212 B920       62    217         MOV     R1,#CBDTR0        SET DATA INDEX REGISTER
000214 441A       63    218         JMP     CBUS32
                        219 *
```

Master transmits data to PCE2111 display driver 2 of the MB7020160 module subroutine:

entry: C = 0; segments a,b,c and g of the 8 LSDs are converted and output
C = 1; segments d,e,f and p of the 8 LSDs are converted and output
registers CBDTR4 - CBDTR7 contain the 8 LSDs.

```
000216 8802          64   227 CBUS31 ORL     P0,#DLEN2        SET DLEN2 OUTPUT
000218 B924          65   228        MOV     R1,#CBDTR4       SET DATA INDEX REGISTER
                          229 *
00021A 5446          66   230 CBUS32 CALL    CBLZ0            TEST BUS STATUS
                          231 *
00021C F1            67   232 CONV00 MOV     A,@R1            FETCH DIGITS
00021D 47            68   233        SWAP    A
00021E 530F          69   234        ANL     A,#H'0F'
000220 A3            70   235        MOVP    A,@A             CONVERT MSD OF DATA BYTE
000221 E624          71   236        JNC     CONV01
000223 47            72   237        SWAP    A
                          238 *
000224 53F0          73   239 CONV01 ANL     A,#H'F0'
000226 AA            74   240        MOV     R2,A
000227 F1            75   241        MOV     A,@R1
000228 530F          76   242        ANL     A,#H'0F'
00022A A3            77   243        MOVP    A,@A             CONVERT LSD OF DATA BYTE
00022B F62E          78   244        JC      CONV02
00022D 47            79   245        SWAP    A
                          246 *
00022E 530F          80   247 CONV02 ANL     A,#H'0F'
000230 4A            81   248        ORL     A,R2             COMBINE BOTH SEGM. CODES
000231 5449          82   249        CALL    CBOUT3           OUTPUT SEGMENT CODE
000233 19            83   250        INC     R1               INCR. DATA INDEX REGISTER
000234 F9            84   251        MOV     A,R1
000235 121C          85   252        JB0     CONV00           JUMP IF NOT READY
000237 321C          86   253        JB1     CONV00                    ,,
000239 27            87   254        CLR     A
00023A E63D          88   255        JNC     CBUS33
00023C 37            89   256        CPL     A
                          257 *
00023D 5447          90   258 CBUS33 CALL    CBPLS            OUTPUT REGISTER SELECT BIT
00023F 98FC          91   259        ANL     P0,#.NOT.(DLEN1+DLEN2) RESET DLEN OUTPUTS
000241 5446          92   260        CALL    CBLZ0            OUTPUT LOAD PULSE
000243 D3E0          93   261        XRL     A,#MST+TRX+BB    TEST BUS STATUS
000245 83            94   262        RET
                          263 *
                          264 * OUTPUT ONE DATA BIT 0 SUBROUTINE:
                          265 *
000246 27            95   266 CBLZ0  CLR     A
                          267 *
                          268 * OUTPUT ONE DATA BIT SUBROUTINE:
                          269 *
000247 9DE9          96   270 CBPLS  MOV     S1,#MST+TRX+BB+ESO+BC0 SET BITCOUNTER TO 1
                          271 *
                          272 * DATA OUTPUT SUBROUTINE:
                          273 *
000249 3C            97   274 CBOUT3 MOV     S0,A             LOAD SIO DATA REGISTER
                          275 *
00024A 0D            98   276 CBTBS3 MOV     A,S1             FETCH BUS STATUS
00024B F24E          99   277        JB7     CBTB30           JUMP IF MST = 1

00024D 83           100   278        RET
                          279 *
00024E 924A         101   280 CBTB30 JB4     CBTBS3           JUMP IF PIN = 1
000250 83           102   281        RET
                          282 *
000251                    283        END
```

3.1.7 Master transmitter/receiver routine - special format (2-line IBUS)

These routines can only be used if no other masters are connected to the I^2C bus.

The 2-line IBUS controls directly text-handling circuits, e.g. the teletext decoder.

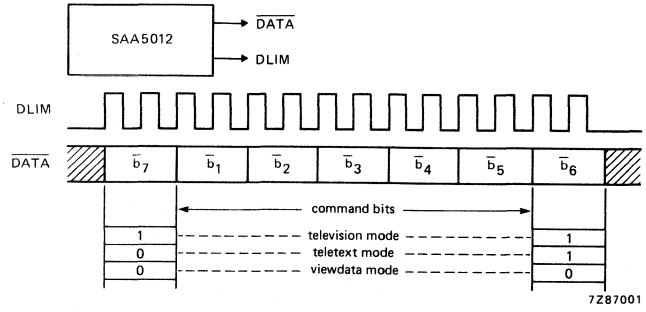


Fig. 3.27 The 2-line IBUS structure

Characteristics:

- 2 clock pulses per bit
- clock burst of 14 clock pulses
- 5 command bits
- 2 mode selection bits
- typical clock frequency of 62,5 kHz

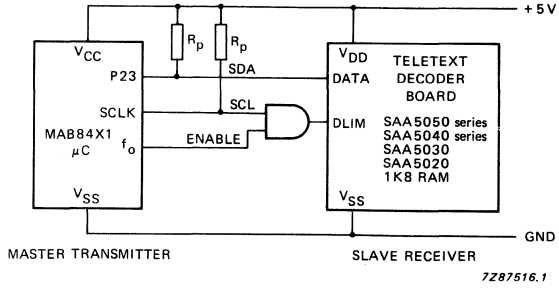


Fig. 3.28 Block diagram MAB84X1 - Teletex decoder configuration.

```
                                    11 * SYMBOL DEFINITION:
                                    12 *
                                    13 IICFR  EQU     H'02'          Fscl =  98 kHz @Fxtal = 4.43 MHz
                                    14 IBFR   EQU     H'05'          Fscl =  59 kHz @Fxtal = 4.43 MHz
                                    15 ENABLE EQU     H'01'          P00 IS IBUS ENABLE OUTPUT
                                    16 *
                                    17 * POWER-ON RESET:
                                    18 *
                                    19        PAGE    256
                                    20 ROM0   ASECT   ROM
000000                              21        ORG     H'000'
                                    22 *
000000 2400           1             23 RESET  JMP     INIT           JUMP TO INITIALIZE ROUTINE
                                    24 *
                                    25 * EXTERNAL INTERRUPT SERVICE SUBROUTINE:
                                    26 *
000002                              27        ORG     H'003'
                                    28 *
000003 00             2             29 EXINT  NOP                    INSERT EXTERNAL INTERRUPT SUBROUTINE
000004 93             3             30        RETR
                                    31 *
                                    32 * TIMER INTERRUPT SERVICE SUBROUTINE:
                                    33 *
000005                              34        ORG     H'007'
                                    35 *
000007 00             4             36 TIMINT NOP                    INSERT TIMER INTERRUPT SUBROUTINE
000008 93             5             37        RETR
                                    38 *
                                    39 * INITIALISATION:
                                    40 *
000009                              41        ORG     H'100'
                                    42 *
000100 9E42           6             43 INIT   MOV     S2,#IICFR+ACK  LOAD SCL FREQUENCY WORD
                                    44 *                             SET WITH ACKNOWLEDGE MODE
000102 9D18           7             45        MOV     S1,#PIN+ESO    ENABLE SIO
                                    46 *                             SET SLAVE RECEIVER MODE
000104 98FE           8             47        ANL     P0,#.NOT.ENABLE RESET IBUS ENABLE OUTPUT
000106 8D00           9             48        MOV     R5,#H'00'      SET IBUS CONTROL WORD
000108 55            10             49        STRT    T              START TIMER
000109 25            11             50        EN      TCNTI          ENABLE TIMER INTERRUPT
00010A 05            12             51        EN      I              ENABLE EXTERNAL INTERRUPT
                                    52 *
```

Main program:

The main program increments the 2-line IBUS control command stored in register R5. The format of this control word is: X B7 B1 B2 B3 B4 B5 B6.

This 2-line IBUS control word is output in the subroutine MIBUS according to the format in Fig. 23:

S	SP ADR	W	NA	$\overline{B7}$	B7	$\overline{B1}$	B1	$\overline{B2}$	B2	$\overline{B3}$	B3	$\overline{B4}$	B4	$\overline{B5}$	B5	$\overline{B6}$	B6	P

ENABLE is high

Fig. 3.29 I^2C bus format of a 2-line IBUS control word.

S	is the START condition
SP ADR	is the special format slave address
W	is the read/write bit in write state (= 0)
NA	is the not acknowledge bit; has to be 1
B7	is the command bit 7 inverted
P	is the STOP condition

```
                                    71 SPAD   EQU     H'04'
                                    72 *
00010B 1D             13            73 MAIN   INC     R5             INCREMENT IBUS COMMAND
                                    74 *
00010C 3412           14            75 MAIN1  CALL    MIBUS          CONVERT IBUS COMAND AND OUTPUT IT
00010E 960C           15            76        JNZ     MAIN1          REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000110 240B           16            77        JMP     MAIN           CONTINUE
                                    78 *
```

Master transmitter IBus to teletext decoder subroutine:

The subroutine first converts the 7 bits command, stored in R5, into a 14 bit word which is stored in registers R2 and R3.

After this conversion these registers contain:

R2 = B7 B7 B1 B1 B2 B2 B3 B3
R3 = B4 B4 B5 B5 B6 B6 X X

To transmit data of a special format, first the special format slave address has to be transmitted. At the reception of this address, none of the connected I^2C-bus devices is allowed to generate an acknowledge.

To disable all other data on the bus, a third line called ENABLE is used to enable only the IBUS commands. After the special format slave address, this ENABLE line is set HIGH and the IBUS command can be output.

Since the clock-low time within a command is limited to 60 usec, the timer/counter interrupts and the external interrupts are disabled. Both interrupts are enabled again when the subroutine is terminated.

entry: R5 = X B7 B1 B2 B3 B4 B5 B6
exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful.

The contents of registers R2, R3 and R4 are modified.

```
                          102 *
000112 BC07         17    103 MIBUS   MOV     R4,#7           SET LOOP COUNTER
000114 FD           18    104         MOV     A,R5            FETCH IBUS COMMAND
                          105 *
000115 E7           19    106 MIBUS1  RL      A               SHIFT LEFT
000116 F21D         20    107         JB7     MIBUS2          JUMP IF BIT IS 1
000118 2B           21    108         XCH     A,R3
000119 4303         22    109         ORL     A,#H'03'        SET TWO BITS

00011B 2420         23    110         JMP     MIBUS3
                          111 *
00011D 2B           24    112 MIBUS2  XCH     A,R3
00011E 53FC         25    113         ANL     A,#H'FC'        RESET TWO BITS
                          114 *
000120 F7           26    115 MIBUS3  RLC     A               SHIFT LEFT DOUBLE BYTES DATA WORD
000121 2A           27    116         XCH     A,R2
000122 F7           28    117         RLC     A
000123 2A           29    118         XCH     A,R2
000124 F7           30    119         RLC     A
000125 2A           31    120         XCH     A,R2
000126 F7           32    121         RLC     A
000127 2A           33    122         XCH     A,R2
000128 2B           34    123         XCH     A,R3
000129 EC15         35    124         DJNZ    R4,MIBUS1       DECR AND TEST LOOP COUNTER
```

entry: R2 = B7 B7 B1 B1 B2 B2 B3 B3

R3 = $\overline{B4}$ $\overline{B4}$ $\overline{B5}$ $\overline{B5}$ $\overline{B6}$ $\overline{B6}$ X X
exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful.

The contents of register R4 are modified.

```
                        132 *
00012B 9D18       36    133 MIBUS4 MOV     S1,#PIN+ESO      RESET SIO BUS STATUS
00012D 9C04       37    134        MOV     S0,#SPAD         LOAD SPECIAL FORMAT SLAVE ADDRESS AND
                        135 *                                WRITE BIT
00012F 9DF8       38    136        MOV     S1,#STRTC        OUTPUT START COND, SPECIAL FORMAT SLAVE
                        137 *                                ADDRESS AND WRITE BIT
                        138 *
000131 0D         39    139 MIBUS5 MOV     A,S1             FETCH BUS STATUS
000132 F236       40    140        JB7     MIBUS6           JUMP IF MST IS 1
000134 245A       41    141        JMP     ERROR            JUMP IF ERROR
                        142 *
000136 9231       42    143 MIBUS6 JB4     MIBUS5           JUMP IF PIN IS 1
000138 35         43    144        DIS     TCNTI            DISABLE INTERRUPTS
000139 15         44    145        DIS     I
00013A 9E05       45    146        MOV     S2,#IBFR         SET IBUS FREQUENCY
                        147 *                                RESET ACKNOWLDGE
00013C 8801       46    148        ORL     P0,#ENABLE       SET IBUS ENABLE OUTPUT
00013E FA         47    149        MOV     A,R2
00013F 3C         48    150        MOV     S0,A             OUTPUT B7 B7 B1 B1 B2 B2 B3 B3
000140 BC07       49    151        MOV     R4,#7
000142 EC42       50    152        DJNZ    R4,$             DELAY
000144 FB         51    153        MOV     A,R3
000145 9DEE       52    154        MOV     S1,#MST+TRX+BB+ESO+BC2+BC1 SET BITCOUNTER TO 6
                        155 *
000147 3C         53    156        MOV     S0,A             OUTPUT B4 B4 B5 B5 B6 B6 X  X
000148 BC06       54    157        MOV     R4,#6
00014A EC4A       55    158        DJNZ    R4,$             DELAY
00014C 98FE       56    159        ANL     P0,#.NOT.ENABLE  RESET IBUS ENABLE OUTPUT
00014E 9E42       57    160        MOV     S2,#IICFR+ACK    RESET IIC FREQ AND ACKN. MODE BIT
000150 0D         58    161        MOV     A,S1             FETCH BUS STATUS
000151 4301       59    162        ORL     A,#LRB           SET LRB
                        163 *
000153 9DD8       60    164        MOV     S1,#STOPC        OUTPUT STOP CONDITION
000155 D3E1       61    165        XRL     A,#MST+TRX+BB+LRB TEST BUS STATUS


000157 25         62    166        EN      TCNTI            ENABLE INTERRUPTS
000158 05         63    167        EN      I
000159 83         64    168        RET
                        169 *
00015A 9E02       65    170 ERROR  MOV     S2,#IICFR
00015C 9E42       66    171        MOV     S2,#IICFR+ACK
00015E 9D18       67    172        MOV     S1,#PIN+ESO
000160 9D18       68    173        MOV     S1,#PIN+ESO
000162 0C         69    174        MOV     A,S0
000163 2302       70    175        MOV     A,#2
000165 98FE       71    176        ANL     P0,#.NOT.ENABLE
000167 83         72    177        RET
000168                  178        END
```

3.2 Slave routines

All these slave routines are I/O interrupt driven

3.2.1 Slave receiver routine (MAB84X1 - master)

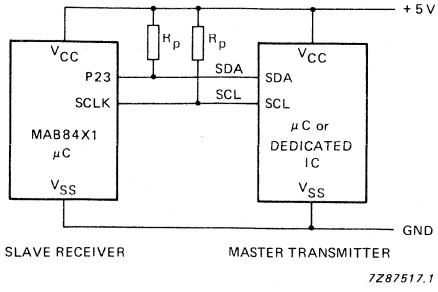


Fig. 3.30 Block diagram MAB84XX - master configuration.

Symbol definition:

```
 8 *
 9 * SYMBOL DEFINITION:
10 IICFR   EQU     H'02'   Fscl = 98,4 kHz @Fcrystal = 4.43 MHz
11 NRSR    EQU     4       number of data bytes which have to be received in
12 *                        SLV/REC mode
13 OWNAD   EQU     H'50'   Own slave address
14 *
```

Data register definition:

```
15 * DATA REGISTER DEFINITION:
16 *
17 * Register Bank 1
18 R11     EQU     H'19'   SIO interrupt data index register
19 R13     EQU     H'1B'   SIO interrupt data byte counter register
20 R14     EQU     H'1C'   SIO interrupt IIC-BUS status register
21 R17     EQU     H'1F'   accu save register during interrupt service subroutines
22 *
23 SRDTR1  EQU     H'3C'           slave receiver data byte 1 register
24 SRDTR2  EQU     SRDTR1+1                  ''              2     ''
25 SRDTR3  EQU     SRDTR2+1                  ''              3     ''
26 SRDTR4  EQU     SRDTR3+1                  ''              4     ''
```

Status flag definition of status register R14:
SRDAVF EQU H'10' SLV/REC data valid flag.

Flag SRDAVF is set in the serial I/O interrupt service subroutine, if the microcomputer has received its own slave address and "NRSR" data bytes.

The flag is cleared in the main program when the received data has been handled.

Initialization:

```
                                37 ROM0   ASECT   ROM
000000                          38        ORG     H'000'
                                39 *
000000 2400           1         40 RESET  JMP     INIT
                                41 *
                                42 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE
                                43 *
000002                          44        ORG     H'005'
                                45 *
000005 4400           2         46 SIOINT JMP     SIOIN1
                                47 *
                                48 * POWER-ON RESET INITIALISATION ROUTINE
                                49 *
000007                          50        ORG     H'100'
                                51 *
000100 B83F           3         52 INIT   MOV     R0,#H'3F'
000102 27             4         53        CLR     A
                                54 *
000103 A0             5         55 INIT1  MOV     @R0,A           CLEAR ALL DATA REGISTERS
000104 E803           6         56        DJNZ    R0,INIT1
                                57 *
                                58 * INITIALIZE SIO
                                59 *
000106 9E42           7         60        MOV     S2,#IICFR+ACK   INITIALISE IIC-BUS FREQUENCY;
                                61 *                                SET WITH ACKNOWLEDGE MODE
000108 9C50           8         62        MOV     S0,#OWNAD       SET OWN SLAVE ADDRESS
00010A 9D18           9         63        MOV     S1,#PIN+ESO     INITALIZE SIO STATUS
00010C 85            10         64        EN      SI
                                65 *
```

Main program:

With this software, data with the format in Fig. 3.31 can be received.

S	OWN ADR W A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 A	P

Fig. 3.31 Data format of transfer to slave receiver.

S	is the START condition
OWN ADR	is the device's own slave address
W	is the read/write bit in the write state (= 0)
A	is the acknowledge bit; this has to be zero
P	is the STOP condition

The main program tests the slave receiver data valid "SRDAVF" flag which is set in the serial I/O interrupt service subroutine, if its own slave address, followed by "NRSR" data bytes, is received.

If this flag is set the main program handles the received data stored in register SRDTR1-4, and resets the flag.

If the microcontroller receives its own slave address while the "SRDAVF" flag is set, it releases the bus again and generates a not acknowledge after the first following data byte.

When more than "NRSR" data bytes are received, these data bytes are ignored.

Since the microcontroller cannot function as master in this program, the MST and AL bits are not tested in the interrupt service subroutine.

At the reception of a general call message (ADO = 1), the bus is released every time by reading the register SO. The received data itself is ignored.

This interrupt service subroutine does not serve a slave transmitter. If it is addressed as a slave transmitter, it outputs code FFH.

```
00010D B81C          11      98 MAIN    MOV     R0,#R14
                             99 *
00010F F0            12     100 MAIN1   MOV     A,@R0           FETCH STATUS
000110 37            13     101         CPL     A
000111 920F          14     102         JB4     MAIN1           JUMP IF SRDAVF IS NOT SET
000113 B83C          15     103         MOV     R0,#SRDTR1
000115 B904          16     104         MOV     R1,#4           TRANSFER DATA FROM SRDTR TO R4-R7
000117 BA04          17     105         MOV     R2,#4
                            106 *
000119 F0            18     107 MAIN2   MOV     A,@R0           TRANSFER DATA
00011A A1            19     108         MOV     @R1,A
00011B 19            20     109         INC     R1

00011C 18            21     110         INC     R0
00011D EA19          22     111         DJNZ    R2,MAIN2
00011F B81C          23     112         MOV     R0,#R14
000121 23EF          24     113         MOV     A,#.NOT.SRDAVF
000123 95            25     114         DIS     SI
000124 50            26     115         ANL     A,@R0           RESET SRDAVF
000125 A0            27     116         MOV     @R0,A
000126 85            28     117         EN      SI
000127 240F          29     118         JMP     MAIN1
                            119 *
                            120 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                            121 *
000129                      122         ORG     H'200'
                            123 *
                            124 *
000200 D5            30     125 SIOIN1  SEL     RB1             SELECT REGISTER BANK 1
000201 AF            31     126         MOV     R7,A            SAVE ACCU
000202 0D            32     127         MOV     A,S1            FETCH BUS STATUS
000203 D221          33     128         JB6     STERR           JUMP IF TRANSMITTER
000205 321E          34     129         JB1     SRENA           JUMP IF GENERAL CALL
000207 5217          35     130         JB2     SRADR           JUMP IF ADDRESSED AS SLAVE
                            131 *
000209 FB            36     132 SRDAT   MOV     A,R3            FETCH DATA BYTE COUNTER
00020A C61E          37     133         JZ      SRENA           JUMP IF MORE THAN "NRSR" DATA BYTES RECEIVED
00020C EB12          38     134         DJNZ    R3,SRDAT1       JUMP IF LESS THAN "NRSR" DATA BYTES RECEIVED
00020E FC            39     135         MOV     A,R4
00020F 4310          40     136         ORL     A,#SRDAVF       SET SRDAVF
000211 AC            41     137         MOV     R4,A
                            138 *
000212 0C            42     139 SRDAT1  MOV     A,S0            FETCH RECEIVED DATA BYTE
000213 A1            43     140         MOV     @R1,A           SAVE DATA BYTE IN "SRDTR" REGISTERS
000214 19            44     141         INC     R1              INCR. DATA BYTE INDEX
000215 FF            45     142         MOV     A,R7            RESTORE ACCU
000216 93            46     143         RETR                    RETURN TO MAIN PROGRAM
                            144 *
000217 FC            47     145 SRADR   MOV     A,R4
000218 9223          48     146         JB4     SRNAC           JUMP IF SRDAVF IS ALREADY SET
00021A B93C          49     147         MOV     R1,#SRDTR1      SET DATA BYTE INDEX
00021C BB04          50     148         MOV     R3,#NRSR        SET DATA BYTE COUNTER
                            149 *
00021E 0C            51     150 SRENA   MOV     A,S0            RELEASE BUS
00021F FF            52     151         MOV     A,R7
000220 93            53     152         RETR
                            153 *
000221 1229          54     154 STERR   JB0     STERR1          JUMP IF NO ACKN RECEIVED
                            155 *
000223 9D68          55     156 SRNAC   MOV     S1,#TRX+BB+ESO  GENERATE NOT ACKNOWLEDGE ON NEXT BYTE
000225 9CFF          56     157         MOV     S0,#H'FF'       OUTPUT H'FF'
000227 FF            57     158         MOV     A,R7
000228 93            58     159         RETR
                            160 *
000229 9D38          59     161 STERR1  MOV     S1,#PIN+ESO+BB  SET IN SLV/REC MODE
00022B FF            60     162         MOV     A,R7
00022C 93            61     163         RETR
                            164 *
00022D                      165         END
```

3.2.2 Slave transmitter routine (MAB84XX - master)

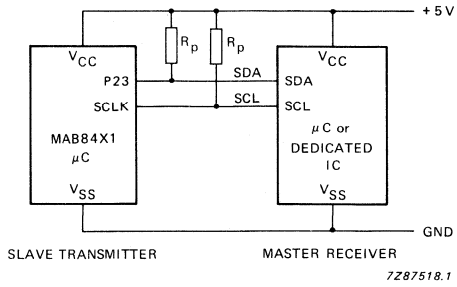


Fig. 3.32 Block diagram MAB84XX - master configuration.

Symbol definition:

```
10 IICFR  EQU     H'02'   Fscl = 98,4 kHz @Fcrystal = 4.43 MHz
11 OWNAD  EQU     H'50'   Own slave address
12 NRST   EQU     4       number of data bytes to be transmitted in SLV/TRM mode
13 *
```

Data register definition:

```
16 * Register Bank 1
17 R11    EQU     H'19'   SIO interrupt data index register
18 R17    EQU     H'1F'   accu save register during interrupt service subroutines
```

slave transmitter data registers:

```
20 * slave transmitter data registers:
21 STDTR1 EQU     H'38'             slave transmitter data byte 1 register
22 STDTR2 EQU     STDTR1+1                      ''              2   ''
23 * to
24 STDTRN EQU     STDTR1+NRST-1                 ''              N   ''
```

Initialization:

```
000000                    31        ORG     H'000'
                          32 *
000000 2400          1    33 RESET  JMP     INIT
                          34 *
                          35 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                          36 *
000002                    37        ORG     H'005'
                          38 *
000005 4400          2    39 SIOINT JMP     SIOIN1
                          40 *
                          41 * POWER-ON RESET INITIALISATION ROUTINE:
                          42 *
000007                    43        ORG     H'100'
                          44 *
000100 B83F          3    45 INIT   MOV     R0,#H'3F'
000102 27            4    46        CLR     A
                          47 *
000103 A0            5    48 INIT1  MOV     @R0,A           CLEAR ALL DATA REGISTERS
000104 E803          6    49        DJNZ    R0,INIT1
                          50 *
                          51 * INITIALIZE SIO ROUTINE:
                          52 *
000106 9E42          7    53        MOV     S2,#IICFR+ACK   INITIALISE IIC-BUS FREQUENCY;
                          54 *                               SET WITH ACKNOWLEDGE MODE
000108 9C50          8    55        MOV     S0,#OWNAD       SET OWN SLAVE ADDRESS
00010A 9D18          9    56        MOV     S1,#PIN+ESO     INITALIZE SIO STATUS
00010C 85           10    57        EN      SI
```

Main program:

With this software, data can be transmitted according the format in Fig. 3.33:

S	OWN ADR. R A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 NA	P

Fig. 3.33 Format of a slave transmitter routine

S	is the START condition
OWN ADR	is the device's own slave address
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be '0'
NA	is the not acknowledge bit; this has to be '1'
P	is the STOP condition

The main program increments the contents of the slave transmitter data registers STDTR1-STDTRN when the bus is free.

The serial I/O interrupt service subroutine outputs the contents of the slave transmitter data registers if the microcomputer is addressed as slave transmitter.

If a "not acknowledge" is received, it switches to the slave receiver mode to enable the master to generate a STOP condition.

If the microcomputer is addressed as slave receiver (own address or general call address) it releases the bus and ignores the received data.

In this program the microcomputer cannot function as a master; therefore the MST and AL bits are not tested in the interrupt service subroutine.

```
                        83 *
000010D 0D              11  84 MAIN    MOV     A,S1
00010E B20D             12  85         JB5     MAIN            WAIT FOR BB = 0
000110 95               13  86         DIS     SI              DISABLE INTERRUPT
000111 B93B             14  87         MOV     R1,#STDTRN
000113 BA04             15  88         MOV     R2,#NRST
000115 97               16  89         CLR     C
000116 A7               17  90         CPL     C
                        91 *
000117 F1               18  92 MAIN1   MOV     A,@R1           INCR. CONTENTS OF REGISTERS STDTR1-N
000118 1300             19  93         ADDC    A,#0
00011A A1               20  94         MOV     @R1,A
00011B C9               21  95         DEC     R1
00011C EA17             22  96         DJNZ    R2,MAIN1
00011E 85               23  97         EN      SI
00011F EA1F             24  98         DJNZ    R2,$
000121 240D             25  99         JMP     MAIN
                       100 *
                       101 *
                       102 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                       103 *
000123                 104         ORG     H'200'
                       105 *
000200 D5              26  106 SIOIN1 SEL     RB1             SELECT REGISTER BANK 1
000201 AF              27  107        MOV     R7,A            SAVE ACCU
000202 0D              28  108        MOV     A,S1            FETCH BUS STATUS
000203 D208            29  109        JB6     SLVTRM          JUMP IF TRANSMITTER

000205 0C              30  110        MOV     A,S0            RELEASE SCL
000206 FF              31  111        MOV     A,R7
000207 93              32  112        RETR
                       113 *
000208 5211            33  114 SLVTRM JB2     STADR           JUMP IF ADDRESSED AS SLAVE
00020A 37              34  115        CPL     A
00020B 1213            35  116        JB0     STDAT           JUMP IF ACKN RECEIVED
00020D 9D38            36  117        MOV     S1,#PIN+ESO+BB  SET IN SLV/REC MODE
00020F FF              37  118        MOV     A,R7
000210 93              38  119        RETR
                       120 *
000211 B938            39  121 STADR  MOV     R1,#STDTR1      SET DATA BYTE INDEX
                       122 *
000213 F1              40  123 STDAT  MOV     A,@R1           FETCH DATA BYTE
000214 3C              41  124        MOV     S0,A            TRM DATA BYTE
000215 19              42  125        INC     R1              INCR. DATA BYTE INDEX
000216 FF              43  126        MOV     A,R7
000217 93              44  127        RETR
                       128 *
000218                 129         END
```

3.2.3 Slave transmitter/receiver routine including general call reception (MAB84X1 - master)

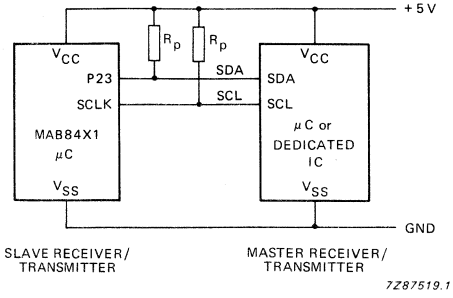


Fig. 3.34 Block diagram MAB84X1 - master configuration.

Symbol definition:

```
IICFR   EQU  H'02'    Fscl = 98,4 kHz @ fcrystal = 4,43 MHz.
NRSR    EQU  4        Number of data bytes which have to be received in
                      SLV/REC mode.
NRGC    EQU  4        Number of data bytes which have to be received in
                      GENERAL CALL mode.
NRST    EQU  4        Number of data bytes to be transmitted in SLV/TRM
                      mode.
OWNAD   EQU  H'50'    Own slave address.
```

Data register definition:

```
Register Bank 1

R11     EQU  H'19'    SIO interrupt data index register
R13     EQU  H'1B'    SIO interrupt data byte counter register
R14     EQU  H'1C'    SIO interrupt I²C-bus status register

Status flag definition of status register R14:

SRDAVF  EQU  H'10'    SLV/REC data valid flag
```

The flag SRDAVF is set in the serial I/O interrupt service subroutine, if the computer has received its own address and "NRSR" data bytes. It is cleared in the main program when the received data has been handled.

```
GCDAVF  EQU  H'08'    GENERAL CALL data valid flag
```

The flag GCDAVF is set in the serial I/O interrupt service subroutine, if the computer has received the general call address and "NRGC" data bytes. It is cleared in the main program when the received data has been handled.

```
R17     EQU  H'1F'    Accumulator-save register during interrupt service
                      subroutines.
```

Slave transmitter data registers:

```
                        36 STDTR1 EQU     H'38'          slave transmitter data byte 1 register
                        37 STDTR2 EQU     STDTR1+1                    ''              2   ''
                        38 * to
                        39 STDTRN EQU     STDTR1+NRST-1               ''              N   ''
                        40 *
                        41 * slave receiver data registers:
                        42 SRDTR1 EQU     H'3C'          slave receiver data byte 1 register
                        43 SRDTR2 EQU     SRDTR1+1                    ''              2   ''
                        44 SRDTR3 EQU     SRDTR2+1                    ''              3   ''
                        45 SRDTR4 EQU     SRDTR3+1                    ''              4   ''
```

Initialization:

```
                        51 *
000000                  52        ORG     H'000'
                        53 *
000000 2400        1    54 RESET  JMP     INIT
                        55 *
                        56 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                        57 *
000002                  58        ORG     H'005'
                        59 *
000005 4400        2    60 SIOINT JMP     SIOIN1
                        61 *
                        62 * POWER-ON RESET INITIALISATION ROUTINE:
                        63 *
000007                  64        ORG     H'100'
                        65 *
000100 B83F        3    66 INIT   MOV     R0,#H'3F'
000102 27          4    67        CLR     A
                        68 *
000103 A0          5    69 INIT1  MOV     @R0,A          CLEAR ALL DATA REGISTERS
000104 E803        6    70        DJNZ    R0,INIT1
                        71 *
                        72 * INITIALIZE SIO ROUTINE:
                        73 *
000106 9E42        7    74        MOV     S2,#IICFR+ACK  INITIALISE IIC-BUS FREQUENCY;
                        75 *                               SET WITH ACKNOWLEDGE MODE
000108 9C50        8    76        MOV     S0,#OWNAD      SET OWN SLAVE ADDRESS
00010A 9D18        9    77        MOV     S1,#PIN+ESO    INITALIZE SIO STATUS
00010C 85         10    78        EN      SI
                        79 *
```

Main program:

This program is a combination of the routines given is Sections 3.2.1/ 3.1.2 extended by dealing with the reception of the GENERAL CALL DATA.

The microcomputer can function as slave according to the following formats:

- Slave receiver general call address

S	GC ADR W A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 A	P

Fig. 3.35 Slave receiver general call format.

- Slave receiver

S	OWN ADR W A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 A	P

Fig. 3.36 Slave receiver format.

- Slave transmitter

S	OWN ADR R A	DATA 1 A	DATA 2 A	DATA 3 A	DATA 4 NA	P

Fig. 3.37 Slave transmitter format.

S	is the START condition
OWN ADR	is the device's own slave address
GC ADR	is general call address
W	is read/write bit in write state (=0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

If the microcomputer is addressed with the GENERAL CALL ADDRESS it receives the data bytes and sets the GENERAL CALL DATA valid flag "GCDAVF".

If the "SRDAVF" flag is already set and the microcomputer is addressed with the GENERAL CALL address, this flag is cleared and data is overwritten.

The flag "GCDAVF" is tested in the main program. If set, it is cleared when the received data has been handled.

```
00010D B81C          11   109 MAIN    MOV     R0,#R14
00010F F0            12   110         MOV     A,@R0           FETCH STATUS
000110 922A          13   111         JB4     MAIN1           JUMP IF SRDAVF IS SET
000112 722A          14   112         JB3     MAIN1           JUMP IF GCDAVF IS SET
000114 0D            15   113         MOV     A,S1
000115 B20D          16   114         JB5     MAIN            JUMP IF BB = 1
000117 95            17   115         DIS     SI              DISABLE INTERRUPT
000118 B93B          18   116         MOV     R1,#STDTRN
00011A BA04          19   117         MOV     R2,#NRST
00011C 97            20   118         CLR     C
00011D A7            21   119         CPL     C
                          120 *
00011E F1            22   121 MAIN0   MOV     A,@R1           INCR. CONTENTS OF REGISTERS STDTR1-N
00011F 1300          23   122         ADDC    A,#0
000121 A1            24   123         MOV     @R1,A
000122 C9            25   124         DEC     R1
000123 EA1E          26   125         DJNZ    R2,MAIN0
000125 85            27   126         EN      SI
000126 EA26          28   127         DJNZ    R2,$            DELAY
000128 240D          29   128         JMP     MAIN
                          129 *
00012A B83C          30   130 MAIN1   MOV     R0,#SRDTR1
00012C B904          31   131         MOV     R1,#4           TRANSFER DATA FROM SRDTR TO R4-R7
00012E BA04          32   132         MOV     R2,#4
                          133 *
000130 F0            33   134 MAIN2   MOV     A,@R0           TRANSFER DATA
000131 A1            34   135         MOV     @R1,A
000132 19            35   136         INC     R1
000133 18            36   137         INC     R0
000134 EA30          37   138         DJNZ    R2,MAIN2
000136 B81C          38   139         MOV     R0,#R14
000138 23E7          39   140         MOV     A,#.NOT.(SRDAVF+GCDAVF)
00013A 95            40   141         DIS     SI
00013B 50            41   142         ANL     A,@R0           RESET SRDAVF OR GCDAVF
00013C A0            42   143         MOV     @R0,A
00013D 85            43   144         EN      SI
00013E 240D          44   145         JMP     MAIN
                          146 *
                          147 *
000140                    148         ORG     H'200'
                          149 *
                          150 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                          151 *
000200 D5            45   152 SIOIN1  SEL     RB1             SELECT REGISTER BANK 1
000201 AF            46   153         MOV     R7,A            SAVE ACCU
000202 0D            47   154         MOV     A,S1            FETCH SIO STATUS
000203 D237          48   155         JB6     SLVTR           JUMP IF TRX = 1
000205 521D          49   156         JB2     SRADR           JUMP IF ADDRESSED AS SLAVE RECEIVER
                          157 *
                          158 * SLV/REC; INPUT DATA SUBROUTINE:
                          159 *
000207 67            50   160 SRDAT   RRC     A
000208 67            51   161         RRC     A               ADD IN CARRY
000209 FB            52   162         MOV     A,R3
00020A C634          53   163         JZ      SRENA           JUMP IF MORE THAN NRSR OR NRGC DATA
                          164 *                                BYTES RECEIVED
00020C EB16          54   165         DJNZ    R3,SRDAT2       DECR. AND TEST DATA BYTE COUNTER
```

```
00020E 2310          55    166          MOV     A,#SRDAVF       SET SRDAVF
000210 E614          56    167          JNC     SRDAT1          JUMP IF ADO = 0
000212 2308          57    168          MOV     A,#GCDAVF       SET GCDAVF
                           169 *
000214 4C            58    170 SRDAT1   ORL     A,R4            SET SRDAVF OR GCDAVF FLAG
000215 AC            59    171          MOV     R4,A
                           172 *
000216 0C            60    173 SRDAT2   MOV     A,S0            FETCH RECEIVED DATA BYTE
000217 A1            61    174          MOV     @R1,A           SAVE RECEIVED DATA BYTE
000218 19            62    175          INC     R1              INCR. DATA BYTE INDEX
000219 FF            63    176          MOV     A,R7
00021A 93            64    177          RETR
                           178 *
                           179 * SLV; ADDRESSED AS SLAVE SUBROUTINE:
                           180 *
00021B D240          65    181 SLADR    JB6     STADR           JUMP IF ADDRESSED AS SLAVE TRANSMITTER
                           182 *
                           183 * SLV/REC; ADDRESSED AS SLAVE RECEIVER SUBROUTINE:
                           184 *
00021D B93C          66    185 SRADR    MOV     R1,#SRDTR1      SET DATA REGISTER INDEX
00021F 322E          67    186          JB1     GCADR           JUMP IF GENERAL SLAVE ADDRESS
000221 BB04          68    187          MOV     R3,#NRSR        SET SLV/REC DATA BYTE COUNTER
000223 FC            69    188          MOV     A,R4
000224 5318          70    189          ANL     A,#SRDAVF+GCDAVF
000226 C634          71    190          JZ      SRENA           JUMP IF SRDAVF AND GCDAVF ARE NOT SET
                           191 *                                  CONT. WITH RECEPTION OF DATA
000228 9D68          72    192          MOV     S1,#TRX+BB+ESO  SET SLAVE TRANSMITTER MODE
00022A 9CFF          73    193          MOV     S0,#H'FF'       OUTPUT H'FF' AND NEGATIVE ACKN.
00022C FF            74    194          MOV     A,R7
00022D 93            75    195          RETR
                           196 *
                           197 * SLV/REC; ADDRESSED WITH GENERAL CALL ADDRESS SUBROUTINE:
                           198 *
00022E BB04          76    199 GCADR    MOV     R3,#NRGC        SET GENERAL CALL ADDRESS DATA BYTE
                           200 *                                  COUNTER
000230 FC            77    201          MOV     A,R4
000231 53E7          78    202          ANL     A,#.NOT.(SRDAVF+GCDAVF) RESET SRDAVF AND GCDAVF FLAGS
000233 AC            79    203          MOV     R4,A
                           204 *
                           205 * SLV/REC; ENABLE SLC SUBROUTINE:
                           206 *
000234 0C            80    207 SRENA    MOV     A,S0            RELEASE SCL
000235 FF            81    208          MOV     A,R7
000236 93            82    209          RETR
                           210 *
                           211 * SLAVE TRANSMITTER SUBROUTINE:
                           212 *
000237 37            83    213 SLVTR    CPL     A               COMPL. STATUS BITS
000238 123E          84    214          JB0     SLVTR1          JUMP IF ACKN. RECEIVED
                           215 *
                           216 * SLV/TRX; NEGATIVE ACKNOWLEDGE RECEIVED SUBROUTINE:
                           217 *
00023A 9D38          85    218 STNAC    MOV     S1,#BB+PIN+ESO  SET SLV/REC MODE
00023C FF            86    219          MOV     A,R7
00023D 93            87    220          RETR
                           221 *
                           222 * SLV/TRX; ACKNOWLEDGE RECEIVED SUBROUTINE:
                           223 *
00023E 5242          88    224 SLVTR1   JB2     STDAT           JUMP IF AAS = 0
                           225 *
                           226 * SLV/TRX; ADDRESSED AS SLAVE TRANSMITTER SUBROUTINE:
                           227 *
000240 B938          89    228 STADR    MOV     R1,#STDTR1      SET DATA BYTE REG. INDEX
                           229 *
000242 F1            90    230 STDAT    MOV     A,@R1           FETCH NEW DATA BYTE
000243 3C            91    231          MOV     S0,A            OUTPUT DATA BYTE
000244 19            92    232          INC     R1              INCR. DATA BYTE INDEX
000245 FF            93    233          MOV     A,R7
000246 93            94    234          RETR
                           235 *
000247                     236          END
```

3.3 Multi - master routines

3.3.1 Master transmitter/receiver routine with repeated start (MAB84X1-PCD8571)

These routines can be used in a multi-master system.

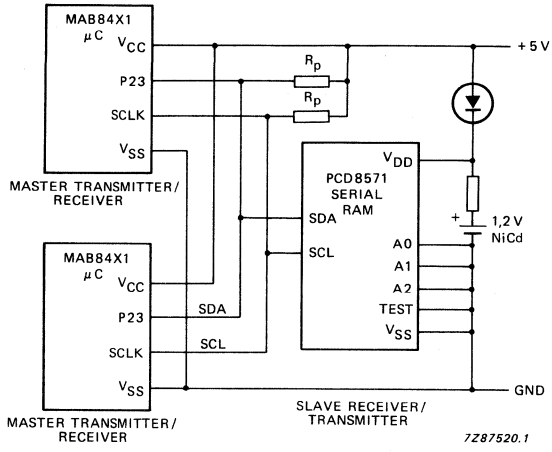


Fig. 3.38 Block diagram multi-master MAB84X1 - PCD8571 configuration.

Initialization:

In a multi master system a microcomputer can be addressed with the general call address and with its own address. In this example the computer can not function as slave so no own slave address is defined. After power-on reset the contents of slave address register is 00H which means that its own slave address is equal to the general call address.

```
000000                      23          ORG     H'000'
                            24  *
000000 2400          1      25  RESET   JMP     INIT            JUMP TO INITIALIZE ROUTINE
                            26  *
                            27  *
                            28  * SERIAL I/O INTERRUPT SERVICE SUBROUTINE ENTRY
                            29  *
000002                      30          ORG     H'005'
                            31  *
000005 24C2          2      32  SIOINT JMP     SIOIN1          JUMP TO SIO INTERRUPT SERVICE SUBROUTINE
                            33  *
000007                      41          ORG     H'100'
                            42  *
000100 9E44          3      43  INIT    MOV     S2,#IICFR+ACK   LOAD SCL FREQUENCY WORD
                            44  *                               SET WITH ACKNOWLEDGE MODE
000102 9D38          4      45          MOV     S1,#PIN+ESO+BB  ENABLE SIO
                            46  *                               SET SLAVE RECEIVER MODE
000104 23FF          5      47          MOV     A,#H'FF'
000106 AD            6      48          MOV     R5,A            SET POINTER VALUE
000107 A9            7      49          MOV     R1,A            SET DATA BYTE 1 TO BE WRITTEN
000108 AA            8      50          MOV     R2,A                        2
000109 85            9      51          EN      SI              ENABLE SIO INTERRUPTS
```

Main program:

The main program calls the two following I^2C-bus transmission subroutines:

- subroutine WMEM which writes the pointer value and two data bytes to the PCD8571 CMOS memory.
  The format of this transmission is shown in Fig. 3.39:

S	MEM. ADR.	W	A	POINT	A	DATA 1	A	DATA 2	A	P

Fig. 3.39 Format of pointer value and 2 data bytes write to PCD8571.

- subroutine RMEM which writes the pointer value to the PCD8571 memory and reads two data bytes out of it.
  The format of this transmission is shown in Fig. 3.40:

S	MEM. ADR.	W	A	POINT	A	S	MEM. ADR.	R	A	DATA 1	A	DATA 2	NA	P

Fig. 3.40 Format of pointer value write 2 data bytes read with PCD8571.

S	is the START condition
MEM ADR	is the slave address of the CMOS memory PCD8571
POINT	is the pointer address (internal RAM location address)
W	is the read/write bit in write state (= 0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

The subroutines given in these examples are the same as given in Section 3.1.4 but are extended to a multi-master system.

If one of the transmissions does not succeed (A ≠ 0), this transmission is repeated.

```
                          82 * The slave address of the CMOS memory is 1010 XXX.
                          83 MEMAD  EQU     H'A0'
                          84 *
00010A 1D           10    85 MAIN   INC     R5              INCR. POINTER VALUE
00010B 2301         11    86        MOV     A,#1            INCR. DATA TO BE WRITTEN
00010D 6A           12    87        ADD     A,R2             R1,R2 + 1 --> R1,R2
00010E AA           13    88        MOV     R2,A
00010F 27           14    89        CLR     A
000110 79           15    90        ADDC    A,R1
000111 A9           16    91        MOV     R1,A
                          92 *
000112 3420         17    93 MAIN1  CALL    WMEM            WRITE DATA TO CMOS MEMORY
000114 9612         18    94        JNZ     MAIN1           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000116 EE16         19    95        DJNZ    R6,$            DELAY
                          96 *
000118 346A         20    97 MAIN2  CALL    RMEM            READ DATA OUT OF CMOS MEMORY
00011A 9618         21    98        JNZ     MAIN2           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
00011C EE1C         22    99        DJNZ    R6,$            DELAY
00011E 240A         23   100        JMP     MAIN            CONTINUE
                         101 *
```

Master writes two data bytes to CMOS memory (PCD8571) subroutine:

Before starting the transmission this subroutine tests for a maximum time of 7 x 256 x 30 x txtal to see if the bus has become free. If not, it resets the bus-busy bit "BB" with the MOV S1, #PIN + ESO instruction and initializes the other status bits with a second MOV S1, #PIN + ESO instruction. To release an addresed slave transmitter the subroutine BBTST starts a dummy master transmission with slave address (and R/W bit) H'FF'. This dummy transmission is repeated until the bus is free again.

During the initialization, the serial I/O interrupt is enabled to release the bus if the computer receives the general call address or its data.

As the subroutines WMEM and RMEM are based on polling the PIN bit, the serial I/O interrupt is disabled during the execution of these subroutines.

If during the transmission, a not-acknowledge is received the transfer is terminated with a STOP condition and with Accu ≠ 0.

If an arbitration lost situation is detected, the subroutine terminates with Accu ≠ 0. It also enables the serial I/O interrupt again. Now the interrupt service subroutine will release the bus and reset the PIN bit.

entry: R5 contains the pointer value to be transmitted
R1 contains the first data byte which has to be written in the memory
R2 contains the second data byte which has to be written in the memory
exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful.
The contents of register R0 are modified.

```
000120 3445          24    129 WMEM   CALL    BBTST           TEST BUS BUSY
000122 95            25    130        DIS     SI              DISABLE INTERRUPT
000123 98FB          26    131        ANL     P0,#.NOT.TRIG1  TRIGGER !!!!!!
000125 8804          27    132        ORL     P0,#TRIG1       TRIGGER !!!!!!
000127 9CA0          28    133        MOV     S0,#MEMAD       LOAD MEMORY SLAVE ADDRESS AND WRITE BIT
000129 9DF8          29    134        MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
00012B 3461          30    135        CALL    MTTBS           TEST BUS STATUS
00012D 96B3          31    136        JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
00012F FD            32    137        MOV     A,R5            FETCH POINTER VALUE
000130 3C            33    138        MOV     S0,A            TRANSMIT POINTER VALUE
000131 3461          34    139        CALL    MTTBS           TEST BUS STATUS
000133 96B3          35    140        JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
000135 F9            36    141        MOV     A,R1            FETCH FIRST DATA BYTE
000136 3C            37    142        MOV     S0,A            TRANSMIT DATA BYTE 1
000137 3461          38    143        CALL    MTTBS           TEST BUS STATUS
000139 96B3          39    144        JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
00013B FA            40    145        MOV     A,R2            FETCH SECOND DATA BYTE
00013C 3C            41    146        MOV     S0,A            TRANSMIT DATA BYTE 2
00013D 3461          42    147        CALL    MTTBS           TEST BUS STATUS
00013F 96B3          43    148        JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
                           149 *
000141 9DD8          44    150 STPC   MOV     S1,#STOPC       OUTPUT STOP CONDITION
000143 85            45    151        EN      SI              ENABLE SIO INTERRUPTS
000144 83            46    152        RET
```

Bus busy test subroutine:

```
000145 B880          47    156 BBTST   MOV     R0,#H'80'       LOAD BUS BUSY COUNTER
000147 85            48    157         EN      SI
                           158 *
000148 0D            49    159 BBTST1  MOV     A,S1            WAIT UNTILL BUS IS FREE FOR MAXIMUM
000149 37            50    160         CPL     A                7 X 256 X 30 X Txtal
00014A B269          51    161         JB5     BBTST2          JUMP IF BUS IS FREE
00014C E848          52    162         DJNZ    R0,BBTST1       DECR AND TEST BUS BUSY COUNTER
00014E 9D18          53    163         MOV     S1,#PIN+ESO     RESET BB BIT; BUS IS FREE NOW
000150 9D18          54    164         MOV     S1,#PIN+ESO     INITIALIZE OTHER STATUS BITS
000152 9E44          55    165         MOV     S2,#IICFR+ACK   INITIALIZE IIC-FREQ. AND ACKN. MODE BIT
000154 95            56    166         DIS     SI              DISABLE SERIAL I/O INTERRUPTS
000155 9CFF          57    167         MOV     S0,#H'FF'       LOAD SIO DATA REGISTER
000157 9DF8          58    168         MOV     S1,#STRTC       OUTPUT START CONDITION AND H'FF' AS SLAVE
                           169 *                                ADDRESS
000159 34B9          59    170         CALL    MRTBS           TEST BUS STATUS
00015B D301          60    171         XRL     A,#LRB          INVERT ACKN. BIT
00015D 9645          61    172         JNZ     BBTST           JUMP IF ERROR
00015F 2441          62    173         JMP     STPC            OUTPUT STOP CONDITION
```

Master transmitter test bus status subroutine:

```
000161 0D            63    177 MTTBS   MOV     A,S1            FETCH BUS STATUS
000162 F265          64    178         JB7     MTTBS1          JUMP IF MST = 1
000164 83            65    179         RET
                           180 *
000165 9261          66    181 MTTBS1  JB4     MTTBS           JUMP IF PIN = 1
000167 D3E0          67    182         XRL     A,#MTTST        TEST MST/TRX BUS STATUS
                           183 *
000169 83            68    184 BBTST2  RET
```

Master reads two data bytes out of CMOS memory (PCD8571) subroutine:

entry: R5 contains the pointer value to be written to the memory.
exit: A = 0 if transmission is successful and the received data is stored
in registers R3 and R4.
A ≠ 0 if the transmission is not successful.
The contents of register R0, R3 and R4 are modified.

```
00016A 3445          69    194 RMEM    CALL    BBTST           TEST BUS BUSY
00016C 95            70    195         DIS     SI              DISABLE INTERRUPT
00016D 98F7          71    196         ANL     P0,#.NOT.TRIG2  TRIGGER !!!!!!
00016F 8808          72    197         ORL     P0,#TRIG2       TRIGGER !!!!!!
000171 9CAD          73    198         MOV     S0,#MEMAD       LOAD SLAVE ADDRESS AND WRITE BIT
000173 9DF8          74    199         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000175 3461          75    200         CALL    MTTBS           TEST BUS STATUS
000177 96B3          76    201         JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
000179 FD            77    202         MOV     A,R5            FETCH POINTER VALUE
00017A 3C            78    203         MOV     S0,A            TRANSMIT POINTER VALUE
00017B 3461          79    204         CALL    MTTBS           TEST BUS STATUS
00017D 96B3          80    205         JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
00017F 9D18          81    206         MOV     S1,#PIN+ESO     RELEASE SDA AND SCL LINE
                           207 *                                NO STOP CONDITION
000181 B80A          82    208         MOV     R0,#10          LOAD REPEATED START COUNTER
                           209 *
000183 0D            83    210 RPSTR1  MOV     A,S1            FETCH BUS STATUS
000184 B288          84    211         JB5     RPSTR2          JUMP IF BB = 1
000186 128C          85    212         JB0     RPSTR3          JUMP IF BB = 0 AND LRB = 1
                           213 *                                SCL HAS BECOME HIGH
000188 E883          86    214 RPSTR2  DJNZ    R0,RPSTR1       DECR. AND TEST REP. START COUNTER
00018A 24B3          87    215         JMP     ERMEM2          JUMP IF ERROR
                           216 *
00018C 9CA1          88    217 RPSTR3  MOV     S0,#MEMAD+RW    LOAD SLAVE ADDRESS AND READ BIT
00018E 9DF8          89    218         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND READ BIT
000190 34B9          90    219         CALL    MRTBS           TEST BUS STATUS
000192 96B3          91    220         JNZ     ERMEM2          JUMP IF NO ACKN. RECEIVED OR ERROR
000194 0C            92    221         MOV     A,S0            START RECEPTION OF DATA
000195 34B9          93    222         CALL    MRTBS           TEST BUS STATUS
000197 96B3          94    223         JNZ     ERMEM2          JUMP IF ERROR
000199 9E04          95    224         MOV     S2,#IICFR       SET WITHOUT ACKNOWLEDGE MODE
00019B 0C            96    225         MOV     A,S0            FETCH FIRST RECEIVED DATA BYTE
00019C AB            97    226         MOV     R3,A            SAVE
00019D 34B9          98    227         CALL    MRTBS           TEST BUS STATUS
00019F 53FE          99    228         ANL     A,#.NOT.LRB     CLEAR LRB
```

```
0001A1 96B1          100     229           JNZ     ERMEM1              JUMP IF ERROR
0001A3 9DA9          101     230           MOV     S1,#MST+BB+ESO+BC0  SET BIT COUNTER TO ONE
0001A5 0C            102     231           MOV     A,S0                FETCH SECOND DATA BYTE
0001A6 AC            103     232           MOV     R4,A                SAVE DATA BYTE
0001A7 34B9          104     233           CALL    MRTBS               TEST BUS STATUS
                             234 *                                     GENERATE NOT ACKNOWLEDGE
0001A9 D301          105     235           XRL     A,#LRB              INVERT NOT ACKNOWLEDGE BIT
0001AB 96B1          106     236           JNZ     ERMEM1              JUMP IF ERROR
0001AD 9E44          107     237           MOV     S2,#IICFR+ACK       SET WITH ACKNOWLEDGE MODE
0001AF 2441          108     238           JMP     STPC                OUTPUT STOP CONDITION
                             239 *
0001B1 9E44          109     240 ERMEM1    MOV     S2,#IICFR+ACK       SET WITH ACKNOWLEDGE MODE
                             241 *
0001B3 0D            110     242 ERMEM2    MOV     A,S1                FETCH SIO STATUS
0001B4 F241          111     243           JB7     STPC                JUMP IF MASTER; NO ACKN. RECEIVED
0001B6 85            112     244           EN      SI                  ENABLE SIO INTERRUPTS
0001B7 37            113     245           CPL     A                   A # 0; TRANSMISSION NOT SUCCESSFUL
0001B8 83            114     246           RET
```

Master receiver test bus status subroutine:

```
0001B9 0D            115     250 MRTBS     MOV     A,S1                FETCH BUS STATUS
0001BA F2BD          116     251           JB7     MRTBS1              JUMP IF MST = 1
0001BC 83            117     252           RET
                             253 *
0001BD 92B9          118     254 MRTBS1    JB4     MRTBS               JUMP IF PIN = 1
0001BF D3A0          119     255           XRL     A,#MRTST            TEST MST/REC BUS STATUS
0001C1 83            120     256           RET
```

Serial I/O interrupt service subroutine:

In a multi-master system a general call message could be transmitted; every MAB84X1 microcontroller has to respond to this. In this example it is done by enabling the SIO interrupt and setting the PIN bit with the instruction MOV SO,A in the SIO interrupt service subroutine.

In the case of an arbitration lost, this instruction also resets the bits AL and AAS.

```
0001C2 3C            121     266 SIOIN1    MOV     S0,A                SET PIN, RESET AL AND AAS BITS;
                             267 *                                      ENABLE SCL IF GENERAL CALL ADDRESS OR ITS
                             268 *                                      DATA RECEIVED OR IF ARBITRATION LOST IN THE
                             269 *                                      MASTER TRANSMITTER MODE
0001C3 93            122     270           RETR
                             271 *
0001C4                       272           END
```

3.3.2 Master transmitter/receiver routine; slave transmitter/receiver routine (MAB84X1 - MAB84X1)

These routines can be used in a multi-master system.

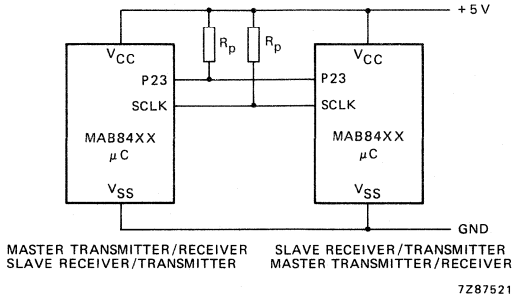


Fig. 3.41 Block diagram multi-master MAB84X1 - MAB84X1 configuration.

Symbol definition:

```
IICFR   EQU  H'02'    fscl = 98,4 kHz @ fcrystal = 4,43 MHz.
NRSR    EQU  4        Number of data bytes which have to be received in
                      SLV/REC mode.
NRGC    EQU  4        Number of data bytes to be transmitted in SLV/TRM
                      mode.
NRST    EQU  4        Number of data bytes which have to be received in
                      GENERAL CALL mode.
```

Data register definition:

```
Register Bank 1

R11     EQU  H'19'    SIO interrupt data index register
R13     EQU  H'1B'    SIO interrupt data byte counter register
R14     EQU  H'1C'    SIO interrupt I2C-bus status register

Status flag definition of status register R14:

SRDAVF  EQU  H'10'    SLV/REC data valid flag
```

The flag SRDAVF is set in the serial I/O interrupt service subroutine if the microcomputer has received its own address and "NRSR" data bytes. It is cleared in the main program when the received data has been handled.

```
GCDAVF  EQU  H'08'    GENERAL CALL data valid flag
```

The flag GCDAVF is set in the serial I/O interrupt service subroutine if the microcomputer has received the general call address and "NRGC" data bytes. It is cleared in the main program when the received data has been handled.

```
R17     EQU  H'1F'     Accumulator-save register during interrupt service
                       subroutines.

master transmitter data registers:

MTDTR1  EQU  H'30'     Master transmitter data byte 1 register
MTDTR2  EQU  MTDTR1+1          "                    2    "
MTDTR3  EQU  MTDTR2+1          "                    3    "
MTDTR4  EQU  MTDTR3+1          "                    4    "

master receiver data registers:

MRDTR1  EQU  H'34'     Master receiver data byte 1 register
MRDTR2  EQU  MRDTR1+1          "                    2    "
MRDTR3  EQU  MRDTR2+1          "                    3    "
MRDTR4  EQU  SRDTR3+1          "                    4    "

slave transmitter data registers:

STDTR1  EQU  H'38'     Slave transmitter data byte 1 register
STDTR2  EQU  STDTR1+1          "                    2    "
STDTR3  EQU  STDTR2+1          "                    3    "
STDTR4  EQU  STDTR3+1          "                    4    "

slave receiver data registers:

SRDTR1  EQU  H'3C'     Slave receiver data byte 1 register
SRDTR2  EQU  SRDTR1+1          "                    2    "
SRDTR3  EQU  SRDTR2+1          "                    3    "
SRDTR4  EQU  SRDTR3+1          "                    4    "
```

Initialization:

```
000000                      74          ORG     H'000'
                            75 *
000000 2400          1      76 RESET    JMP     INIT            JUMP TO INITIALIZE ROUTINE
                            77 *
                            78 *
                            79 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE ENTRY
                            80 *
000002                      81          ORG     H'005'
                            82 *
000005 4400          2      83 SIOINT   JMP     SIOIN1          JUMP TO SIO INTERRUPT SERVICE SUBROUTINE
                            84 *
                            85 * POWER-ON RESET INITIALISATION SUBROUTINE:
                            86 *
000007                      87          ORG     H'100'
                            88 *
000100 B83F          3      89 INIT     MOV     R0,#H'3F'
000102 27            4      90          CLR     A
                            91 *
000103 A0            5      92 INIT1    MOV     @R0,A           CLEAR ALL DATA REGISTERS
000104 E803          6      93          DJNZ    R0,INIT1
                            94 *
                            95 * INITIALIZE SIO ROUTINE:
                            96 *
000106 9E42          7      97          MOV     S2,#IICFR+ACK   INITIALISE IIC-BUS FREQUENCY;
                            98 *                                 SET WITH ACKNOWLEDGE MODE
000108 9C50          8      99          MOV     S0,#OWNAD       SET OWN SLAVE ADDRESS
00010A 9D18          9     100          MOV     S1,#PIN+ESO     INITALIZE SIO STATUS
00010C 85           10     101          EN      SI
```

Main program:

The software given in this example is meant for a MAB84X1 microcontroller which has to function in a multi-master system and which has to receive and transmit data according the following formats:

- Master transmitter

S	uC ADR.	W	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	A	P

Fig. 3.42 Master transmitter format.

- Master receiver

S	uC ADR.	R	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	NA	P

Fig. 3.43 Master receiver format.

- Slave receiver general call address

S	GC ADR.	W	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	A	P

Fig. 3.44 Slave receiver general call format.

- Slave receiver

S	OWN ADR.	W	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	A	P

Fig. 3.45 Slave receiver format.

- Slave transmitter

S	OWN ADR.	R	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	NA	P

Fig. 3.46 Slave transmitter format.

S	is the START condition
uC ADR	is the slave address of the other microcomputer
OWN ADR	is the own slave address
GC ADR	is the general call address
W	is the read/write bit in write state (= 0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

```
UCAD    EQU     H'60'
GCAD    EQU     H'00'
OWNAD   EQU     H'50'
```

The main program increments the four data bytes, stored in register MTDTR1-4, and writes these data bytes to another microcomputer. It also reads four data bytes out of this microcomputer and stores these data bytes in register MRDTR1-4.

In slave mode the program functions as described in Section 3.3.2.

```
00010D B933          11   141 MAIN   MOV     R1,#MTDTR4
00010F BA04          12   142        MOV     R2,#4
000111 97            13   143        CLR     C
000112 A7            14   144        CPL     C
                          145 *
000113 F1            15   146 MAIN1  MOV     A,@R1           INCR. CONTENTS OF REGISTERS MTDTR1-4
000114 1300          16   147        ADDC    A,#0
000116 A1            17   148        MOV     @R1,A
000117 C9            18   149        DEC     R1
000118 EA13          19   150        DJNZ    R2,MAIN1
                          151 *
00011A 3428          20   152 MAIN2  CALL    SUBR            TEST SRDAVF AND GCDAVF FLAGS
00011C 3445          21   153        CALL    WUC             WRITE MTDTR1-4 TO OTHER UC
00011E 961A          22   154        JNZ     MAIN2           REPEAT IF ERROR
                          155 *
000120 3428          23   156 MAIN3  CALL    SUBR            TEST SRDAVF AND GCDAVF FLAGS
000122 348A          24   157        CALL    RUC             READ DATA OUT OF OTHER UC
000124 9620          25   158        JNZ     MAIN3           REPEAT IF ERROR
000126 2400          26   159        JMP     MAIN
                          160 *
                          161 *
000128 B81C          27   162 SUBR   MOV     R0,#R14         SET SYSTEM STATUS REGISTER INDEX
00012A F0            28   163        MOV     A,@R0           FETCH SYSTEM STATUS
00012B 9230          29   164        JB4     SUBR1           JUMP IF SRDAVF IS SET
00012D 7230          30   165        JB3     SUBR1           JUMP IF GCDAVF IS SET
00012F 83            31   166        RET
                          167 *
000130 B83C          32   168 SUBR1  MOV     R0,#SRDTR1
000132 B938          33   169        MOV     R1,#STDTR1      TRANSFER DATA FROM SRDTR1-4 TO STDTR1-4
000134 BB04          34   170        MOV     R3,#4
                          171 *
000136 F0            35   172 SUBR2  MOV     A,@R0           TRANSFER DATA
000137 A1            36   173        MOV     @R1,A
000138 19            37   174        INC     R1
000139 18            38   175        INC     R0
00013A EB36          39   176        DJNZ    R3,SUBR2
00013C B81C          40   177        MOV     R0,#R14
00013E 23E7          41   178        MOV     A,#.NOT.(SRDAVF+GCDAVF)
000140 95            42   179        DIS     SI
000141 50            43   180        ANL     A,@R0           RESET SRDAVF OR GCDAVF
000142 A0            44   181        MOV     @R0,A
000143 85            45   182        EN      SI
000144 83            46   183        RET
```

Master writes four bytes of data to other microcomputer subroutine:

```
entry: MTDTR1-4 contains data to be written
exit:  A = 0 if the transmission is successful
       A ≠ 0 if the transmission is not successful

The contents of register R0 are modified.
```

```
000145 3465          47    191 WUC     CALL    BBTST           BUS BUSY TEST ROUTINE
000147 95            48    192         DIS     SI              DISABLE INTERRUPT
000148 98FB          49    193         ANL     P0,#.NOT.TRIG1  TRIGGER !!!!!!
00014A 8804          50    194         ORL     P0,#TRIG1       TRIGGER !!!!!!
00014C 9C60          51    195         MOV     S0,#UCAD        LOAD UC SLAVE ADDRESS AND WRITE BIT
00014E 9DF8          52    196         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND WRITE BIT
000150 3481          53    197         CALL    MTTBS           TEST BUS STATUS
000152 96C6          54    198         JNZ     ERUC2           JUMP IF NO ACKN. RECEIVED OR ERROR
000154 B830          55    199         MOV     R0,#MTDTR1      SET DATA REGISTER INDEX
000156 BA04          56    200         MOV     R2,#4
                           201 *
000158 F0            57    202 WUC1    MOV     A,@R0           FETCH DATA BYTE
000159 3C            58    203         MOV     S0,A            TRANSMIT DATA BYTE
00015A 18            59    204         INC     R0              INCR INDEX
00015B 3481          60    205         CALL    MTTBS           TEST BUS STATUS
00015D 96C6          61    206         JNZ     ERUC2           JUMP IF NO ACKN. RECEIVED OR ERROR
00015F EA58          62    207         DJNZ    R2,WUC1         DECR DATA BYTE COUNTER
                           208 *
000161 9DD8          63    209 STPC    MOV     S1,#STOPC       OUTPUT STOP CONDITION
000163 85            64    210         EN      SI              ENABLE SIO INTERRUPTS
000164 83            65    211         RET
                           212 *
                           213 * BUS BUSY TEST SUBROUTINE
                           214 *
000165 B880          66    215 BBTST   MOV     R0,#H'80'       LOAD BUS BUSY COUNTER
000167 85            67    216         EN      SI
                           217 *
000168 0D            68    218 BBTST1  MOV     A,S1            WAIT UNTILL BUS IS FREE FOR MAXIMUM
000169 37            69    219         CPL     A                7 X 256 X 30 X Txtal
00016A B289          70    220         JB5     BBTST2          JUMP IF BUS IS FREE
00016C E868          71    221         DJNZ    R0,BBTST1       DECR AND TEST BUS BUSY COUNTER
00016E 9D18          72    222         MOV     S1,#PIN+ESO     RESET BB BIT; BUS IS FREE NOW
000170 9D18          73    223         MOV     S1,#PIN+ESO     INITIALIZE OTHER STATUS BITS
000172 9E42          74    224         MOV     S2,#IICFR+ACK   INITIALIZE IIC-FREQ. AND ACKN. MODE BIT
000174 95            75    225         DIS     SI              DISABLE SERIAL I/O INTERRUPTS
000175 9CFF          76    226         MOV     S0,#H'FF'       LOAD SIO DATA REGISTER
000177 9DF8          77    227         MOV     S1,#STRTC       OUTPUT START CONDITION AND H'FF' AS SLAVE
                           228 *                                ADDRESS
000179 3481          78    229         CALL    MTTBS           TEST BUS STATUS
00017B D301          79    230         XRL     A,#LRB          INVERT ACKN. BIT
00017D 9665          80    231         JNZ     BBTST           JUMP IF ERROR
00017F 2461          81    232         JMP     STPC            OUTPUT STOP CONDITION
```

Master transmitter test bus status subroutine:

```
000181 0D            82    236 MTTBS   MOV     A,S1            FETCH BUS STATUS
000182 F285          83    237         JB7     MTTBS1          JUMP IF MST = 1
000184 83            84    238         RET
                           239 *
000185 9281          85    240 MTTBS1  JB4     MTTBS           JUMP IF PIN = 1
000187 D3E0          86    241         XRL     A,#MTTST        TEST MST/TRX BUS STATUS
                           242 *
000189 83            87    243 BBTST2  RET
```

Master reads four data bytes out of other microcomputer subroutine:

exit: A = 0 if the transmission is successful
Received data bytes are sorted in register MRDTR1-4
A ≠ 0 if the transmission is not successful

The contents of register R0 are modified.

```
00018A 3465          88    251 RUC     CALL    BBTST           TEST BUS BUSY
00018C 95            89    252         DIS     SI              DISABLE INTERRUPT
00018D 98F7          90    253         ANL     P0,#.NOT.TRIG2  TRIGGER !!!!!!
00018F 8808          91    254         ORL     P0,#TRIG2       TRIGGER !!!!!!
000191 9C61          92    255         MOV     S0,#UCAD+RW     LOAD SLAVE ADDRESS AND READ BIT
000193 9DF8          93    256         MOV     S1,#STRTC       OUTPUT START COND, SLAVE ADDRESS AND READ BIT
000195 34CC          94    257         CALL    MRTBS           TEST BUS STATUS
000197 96C6          95    258         JNZ     ERUC2           JUMP IF NO ACKN. RECEIVED OR ERROR
000199 0C            96    259         MOV     A,S0            DUMMY READ; START RECEPTION OF DATA
00019A BA02          97    260         MOV     R2,#4-2         SET DATA BYTE COUNTER
00019C B834          98    261         MOV     R0,#MRDTR1      SET DATA BYTE INDEX
                           262 *
00019E 34CC          99    263 RUC1    CALL    MRTBS           TEST BUS STATUS
0001A0 96C6         100    264         JNZ     ERUC2           JUMP IF NO ACKN. RECEIVED OR ERROR
0001A2 0C           101    265         MOV     A,S0            FETCH RECEIVED DATA BYTE
0001A3 A0           102    266         MOV     @R0,A           STORE RECEIVED DATA BYTE
0001A4 18           103    267         INC     R0              INCR DATA BYTE INDEX
0001A5 EA9E         104    268         DJNZ    R2,RUC1         DECR. DATA BYTE COUNTER
0001A7 34CC         105    269         CALL    MRTBS           TEST BUS STATUS
0001A9 96C6         106    270         JNZ     ERUC2           JUMP IF ERROR
0001AB 9E02         107    271         MOV     S2,#IICFR       SET WITHOUT ACKNOWLEDGE MODE
0001AD 0C           108    272         MOV     A,S0            FETCH LAST BUT ONE TO BE RECEIVED DATA BYTE
0001AE A0           109    273         MOV     @R0,A           SAVE
0001AF 18           110    274         INC     R0              INCR. DATA BYTE INDEX
0001B0 34CC         111    275         CALL    MRTBS           TEST BUS STATUS
0001B2 53FE         112    276         ANL     A,#.NOT.LRB     CLEAR LRB
0001B4 96C4         113    277         JNZ     ERUC1           JUMP IF ERROR
0001B6 9DA9         114    278         MOV     S1,#MST+BB+ESO+BC0 SET BIT COUNTER TO ONE
0001B8 0C           115    279         MOV     A,S0            FETCH LAST TO BE RECEIVED DATA BYTE
0001B9 A0           116    280         MOV     @R0,A           SAVE
0001BA 34CC         117    281         CALL    MRTBS           TEST BUS STATUS
                           282 *                               GENERATE NOT ACKNOWLEDGE
0001BC D301         118    283         XRL     A,#LRB          INVERT NOT ACKNOWLEDGE BIT
0001BE 96C4         119    284         JNZ     ERUC1           JUMP IF ERROR
0001C0 9E42         120    285         MOV     S2,#IICFR+ACK   SET WITH ACKNOWLEDGE MODE
0001C2 2461         121    286         JMP     STPC            OUTPUT STOP CONDITION
                           287 *
0001C4 9E42         122    288 ERUC1   MOV     S2,#IICFR+ACK   SET WITH ACKNOWLEDGE MODE
                           289 *
0001C6 0D           123    290 ERUC2   MOV     A,S1            FETCH SIO STATUS
0001C7 F261         124    291         JB7     STPC            JUMP IF MASTER; NO ACKN. RECEIVED
0001C9 85           125    292         EN      SI              ENABLE SIO INTERRUPTS
0001CA 37           126    293         CPL     A               A # 0; TRANSMISSION NOT SUCCESSFUL
0001CB 83           127    294         RET
                           295 *
                           296 * MASTER RECEIVER TEST BUS STATUS SUBROUTINE:
                           297 *
0001CC 0D           128    298 MRTBS   MOV     A,S1            FETCH BUS STATUS
0001CD F2D0         129    299         JB7     MRTBS1          JUMP IF MST = 1
0001CF 83           130    300         RET
                           301 *
0001D0 92CC         131    302 MRTBS1  JB4     MRTBS           JUMP IF PIN = 1
0001D2 D3A0         132    303         XRL     A,#MRTST        TEST MST/REC BUS STATUS
0001D4 83           133    304         RET
                           305 *
                           306 *
                           307 *
0001D5                     308         ORG     H'200'
                           309 *
                           310 * SERIAL I/O INTERRUPT SERVICE SUBROUTINE:
                           311 *
000200 D5           134    312 SIOIN1  SEL     RB1             SELECT REGISTER BANK 1
000201 AF           135    313         MOV     R7,A            SAVE ACCU
000202 0D           136    314         MOV     A,S1            FETCH SIO STATUS
000203 721D         137    315         JB3     MSTAL           JUMP IF ARB LOST
                           316 *
000205 D23D         138    317 SIOIN2  JB6     SLVTR           JUMP IF TRX = 1
000207 5223         139    318         JB2     SRADR           JUMP IF ADDRESSED AS SLAVE RECEIVER
                           319 *
                           320 * SLV/REC; INPUT DATA SUBROUTINE:
                           321 *
000209 67           140    322 SRDAT   RRC     A
```

```
00020A 67          141   323         RRC     A                ADD IN CARRY
00020B FB          142   324         MOV     A,R3
00020C C63A        143   325         JZ      SRENA            JUMP IF MORE THAN NRSR OR NRGC DATA
                         326 *                                 BYTES RECEIVED
00020E EB18        144   327         DJNZ    R3,SRDAT2        DECR. AND TEST DATA BYTE COUNTER
000210 2310        145   328         MOV     A,#SRDAVF        SET SRDAVF
000212 E616        146   329         JNC     SRDAT1           JUMP IF ADD = 0
000214 2308        147   330         MOV     A,#GCDAVF        SET GCDAVF
                         331 *
000216 4C          148   332 SRDAT1  ORL     A,R4             SET SRDAVF OR GCDAVF FLAG
000217 AC          149   333         MOV     R4,A
                         334 *
000218 0C          150   335 SRDAT2  MOV     A,S0             FETCH RECEIVED DATA BYTE
000219 A1          151   336         MOV     @R1,A            SAVE RECEIVED DATA BYTE
00021A 19          152   337         INC     R1               INCR. DATA BYTE INDEX
00021B FF          153   338         MOV     A,R7
00021C 93          154   339         RETR
                         340 *
                         341 * MASTER HAS LOST ARBITRATION SUBROUTINE:
                         342 *
00021D 5205        155   343 MSTAL   JB2     SIOIN2           JUMP IF ADDRESSED AS SLAVE
00021F 443A        156   344         JMP     SRENA            RELEASE BUS
                         345 *
                         346 * SLV; ADDRESSED AS SLAVE SUBROUTINE:
                         347 *
000221 D246        157   348 SLADR   JB6     STADR            JUMP IF ADDRESSED AS SLAVE TRANSMITTER
                         349 *
                         350 * SLV/REC; ADDRESSED AS SLAVE RECEIVER SUBROUTINE:
                         351 *
000223 B93C        158   352 SRADR   MOV     R1,#SRDTR1       SET DATA REGISTER INDEX
000225 3234        159   353         JB1     GCADR            JUMP IF GENERAL SLAVE ADDRESS
000227 BB04        160   354         MOV     R3,#NRSR         SET SLV/REC DATA BYTE COUNTER
000229 FC          161   355         MOV     A,R4
00022A 5318        162   356         ANL     A,#SRDAVF+GCDAVF
00022C C63A        163   357         JZ      SRENA            JUMP IF SRDAVF AND GCDAVF ARE NOT SET
                         358 *                                 CONT. WITH RECEPTION OF DATA
00022E 9D68        164   359         MOV     S1,#TRX+BB+ESO   SET SLAVE TRANSMITTER MODE
000230 9CFF        165   360         MOV     S0,#H'FF'        OUTPUT H'FF' AND NEGATIVE ACKN.
000232 FF          166   361         MOV     A,R7
000233 93          167   362         RETR
                         363 *
                         364 * SLV/REC; ADDRESSED WITH GENERAL CALL ADDRESS SUBROUTINE:
                         365 *
000234 BB04        168   366 GCADR   MOV     R3,#NRGC         SET GENERAL CALL ADDRESS DATA BYTE
                         367 *                                 COUNTER
000236 FC          169   368         MOV     A,R4
000237 53E7        170   369         ANL     A,#.NOT.(SRDAVF+GCDAVF) RESET SRDAVF AND GCDAVF FLAGS
000239 AC          171   370         MOV     R4,A
                         371 *
                         372 * SLV/REC; ENABLE SLC SUBROUTINE:
                         373 *
00023A 0C          172   374 SRENA   MOV     A,S0             RELEASE SCL
00023B FF          173   375         MOV     A,R7
00023C 93          174   376         RETR
                         377 *
                         378 * SLAVE TRANSMITTER SUBROUTINE:
                         379 *
00023D 37          175   380 SLVTR   CPL     A                COMPL. STATUS BITS
00023E 1244        176   381         JB0     SLVTR1           JUMP IF ACKN. RECEIVED
                         382 *
                         383 * SLV/TRX; NEGATIVE ACKNOWLEDGE RECEIVED SUBROUTINE:
                         384 *
000240 9D38        177   385 STNAC   MOV     S1,#BB+PIN+ESO   SET SLV/REC MODE
000242 FF          178   386         MOV     A,R7
000243 93          179   387         RETR
                         388 *
                         389 * SLV/TRX; ACKNOWLEDGE RECEIVED SUBROUTINE:
                         390 *
000244 5248        180   391 SLVTR1 JB2     STDAT            JUMP IF AAS = 0
                         392 *
                         393 * SLV/TRX; ADDRESSED AS SLAVE TRANSMITTER SUBROUTINE:
                         394 *
000246 B938        181   395 STADR   MOV     R1,#STDTR1       SET DATA BYTE REG. INDEX
                         396 *
000248 F1          182   397 STDAT   MOV     A,@R1            FETCH NEW DATA BYTE
000249 3C          183   398         MOV     S0,A             OUTPUT DATA BYTE
00024A 19          184   399         INC     R1               INCR. DATA BYTE INDEX
00024B FF          185   400         MOV     A,R7
00024C 93          186   401         RETR
                         402 *
00024D                   403         END
```

3.3.3 Further Multi-master routines

The following is an example of an I^2C multi-master routine. The example shown is divided into two sets of routines:

1) The first set (section 3.3.3.1) is the basic multi-master communication routine used by successful masters and called for I^2C-bus communications. The program involves the use of the SAA5240 Euro CCT chip.

2) The second set of routines (section 3.3.3.2) are utility sub-routine examples for certain types of multi-master communication. They are given to illustrate the treatment of the bus at system level.

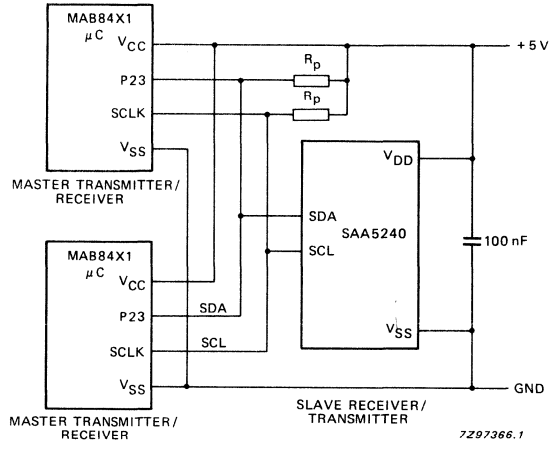


Fig. 3.47 Block diagram of multi-master MAB84X1 - SAA5240 CCT.

All equate listings are shown at the rear of section 3.3.3.2

3.3.3.1 I²C-bus communications multi-master/slave routine

This routine is used by utility subroutines to direct read and write communication between master/slaves in a multi-master environment. The routine begins by saving registers R0 to R6 and setting up the re-try register R5. In register 6 is the number of bytes to be transmitted plus a 'hold' bit. If, after reading the status of the bus the program finds the bus is not engaged, it will take control and read or write data according to the requirement. If the bus is busy i.e. BUSHLD=0, then the SIO interface will release the bus with a stop condition.

If the 'hold' bit is a '1', then the bus is held after a single transmission, the PIN + ESO bits of register S1 of the SIO are set accordingly and the clock line SCL is held HIGH for a repeated start condition. If the 'hold' bit is '0', then the bus is released with a stop condition.

Register R7 is used as the transmission/reception success flag. The routine concludes by restoring the concerned registers and enabling the SIO interrupt.

A pseudo-high level description precedes the program.

```
12          ENTRY   I2C,SIOIN                       Bus Comms
13          ENTRY   ICCS8                           CCT routines
14          ENTRY   ICWR1,ICWR47,ICWR23,ICR23D
15          ENTRY   SETCUR,ZERCUR
16          ENTRY   ICWDCN,ICWUBT,ICWBTI,ICRCB
17          ENTRY   INVWR,INVRD                     NV ram routines
18 *
```

```
                                        print on
**********************************
* Subroutine: IIC Bus Comms.
**********************************
*
* Input : RA (bit 7-1 = Device Address ; bit 0 = Read/Write)
*         TEMP0 (bit 7 = hold bus, bit 6-0 = no. of bytes)
*         TEMP1 - TEMP12 = Data for transmission)
* Output: TEMP0 = No. of bytes (For RD only. Not Kept for WR )
*         TEMP1 - TEMP12 = Data read in
*         R7     = Success flag
* Used  : R7
* Kept  : R0-R6
*
*  Routine description:
*
*       Save registers
*       Save byte_count & retry_countdown
*       repeat:
*           Write ACK_MODE to bus
*           Read Status
*           if (bus_was_not_held {BUSHLD=0}) then:
*               FREE_BUS
*           Disable SIO interrupts
*           Send device_address & Start_condition to bus
*           if (write_mode) then
*               WRITE_BLOCK
*           else
*               READ_BLOCK
*       until (no error OR failed_5_times OR bus_was_held)
*       if (no errors found) then
*           if HOLD_BUS required then
*               Send PIN + ES0 to bus    {clear bus & hold it}
*           else
*               Send STOP_CONDITION to bus {clear bus and free it}
*       Save HOLD_REQUIRED condition in BUSHLD
*       Restore registers
*       Enable interrupts
*
*    Procedure WRITE_BLOCK
*    --  Writes <byte_count> bytes to bus
*
*       Test Status of bus  {routine TBSMT}
*       if (bus_error) then
*           ERROR_ROUTINE
*       else begin:
*           repeat:
*               Send data_byte
*               Decrement byte_count
*               Test Status of bus  {routine TBSMT}
*               if (bus_error) then
*                   ERROR_ROUTINE
*           until (byte_count=0 OR error)
*       end
*
*    Procedure READ_BLOCK
*    --  Reads <byte_count> bytes from the bus
*
*       if (last byte to be read {byte_count=1}) then
*           Write NO_ACK, 8_bit_mode to bus
*       Read data to start reception
*       if (not last byte) {byte_count>1} then begin:
*           repeat:
*               Test status of bus   {routine RBSMT}
*               if (bus_error) then
*                   ERROR_ROUTINE
*               else begin:
*                   if penultimate byte {byte_count=2} then
*                       Set SIO to NO_ACK
*                   Transfer data byte from bus to data block
*               end
*           until last byte {byte_count=1}
*       end
*       if (no errors found) then begin:
*           Test status of bus   {routine RBSMT}
*           if (bus_error other than ACK) then
*               ERROR_ROUTINE
*           else begin:
*               Write 1_bit_byte for -ve ACK to bus
*               Transfer last data byte from bus to data block
*               Test status of bus   {routine RBSMT}
*               Write ACK_MODE to bus
*               If (bus_error) then
*                   ERROR_ROUTINE
*           end
*       end
```

```
                                                                eject
                         I2CSEC  RSECT   ROM,INPAGE
                                 PAGE    256
                         *
                         *-------------------------------
                         * Save R0-R6 ; Not R7
                         *-------------------------------
                         *
000000 AF            1   I2C     MOV     R7,A               R7 = address & R/W
000001 F8            2           MOV     A,R0               Hold R0
000002 B868          3           MOV     R0,#SAVRG0         Buffer pointer
000004 A0            4           MOV     @R0,A              Save R0 - SAVRG0
000005 F9            5           MOV     A,R1               Get R1
000006 B906          6           MOV     R1,#6              Set count to 6 regs
                         *
                         *-------------------------------
                         * Save R1 1st time in, and then
                         * R6 to R2
                         *-------------------------------
                         *
000008 18            7   ISRL1   INC     R0                 Next SAVRG
000009 A0            8           MOV     @R0,A              Save Rn (1,6-2)
00000A F1            9           MOV     A,@R1              Get next reg
00000B E908    R    10           DJNZ    R1,ISRL1           Do 6 times
                         *
                         *-------------------------------
                         * Set up data used for restarts
                         *-------------------------------
                         *
00000D B87F         11           MOV     R0,#TEMP0
00000F F0           12           MOV     A,@R0
000010 AE           13           MOV     R6,A               R6 = Hold & no. of bytes
000011 BD05         14           MOV     R5,#5              R5 = No. of retries
                         *
                         *-------------------------------
                         * Set up data for each try
                         *-------------------------------
                         *
000013 FE           15   I2CSUP  MOV     A,R6
000014 537F         16           ANL     A,#.not.HLD
000016 AC           17           MOV     R4,A               R4 = No. of bytes
000017 B87E         18           MOV     R0,#TEMP1          R0 -> Start of buffer for data

                         *-------------------------------
                         * Test for bus being free
                         *-------------------------------
                         *
000019 9E42         19           MOV     S2,#H'42'          Set ack mode
00001B B920         20           MOV     R1,#BUSHLD         Test whether bus was held
00001D F1           21           MOV     A,@R1
00001E 9622    R    22           JNZ     I2CSFR             Jump if was held
000020 1400    R    23   I2C1    CALL    FREBUS             Free bus if busy
                         *
                         *-------------------------------
                         * Send slave address & start
                         * condition. Test arbitration.
                         *-------------------------------
                         *
000022 95           24   I2CSFR  DIS     SI                 Disable SIO interrupt
000023 FF           25           MOV     A,R7               Get device address
000024 3C           26           MOV     S0,A               Send address
000025 9DF8         27           MOV     S1,#STARTC         Send start condition
000027 125B    R    28   I2C2    JB0     I2CRDT             Jump if reading
000029 1400    R    29           CALL    TBSMT              Test bus status
00002B 968E    R    30           JNZ     MERROR             JMP if error
                         *
                         *-------------------------------
                         * Send data block
                         *-------------------------------
                         *
00002D F0           31   I2CWR   MOV     A,@R0              Get data byte
00002E 3C           32           MOV     S0,A               Send byte
00002F C8           33           DEC     R0                 Point to next data byte
000030 1400    R    34   I2CWR1  CALL    TBSMT              Test bus status
000032 968E    R    35           JNZ     MERROR             JMP if error
000034 EC2D    R    36           DJNZ    R4,I2CWR           Loop until all data bytes gone

                         *-------------------------------
                         * Comms successful.
                         * Control bus holding.
                         *-------------------------------
                         *
000036 BF00         37   I2CCHL  MOV     R7,#0              Set successful flag
000038 FE           38           MOV     A,R6               Get byte with hold flag
000039 5380         39           ANL     A,#HLD             Keep hold bit only
00003B 9641    R    40           JNZ     I2CBHG             Skip bus closing if hold needed
                         *
                         *-------------------------------
                         * Close bus down
                         *-------------------------------
                         *
00003D 9DD8         41   I2CEX   MOV     S1,#STOPC          Send stop condition
00003F 0443    R    42           JMP     I2CWND
```

```
                                204 *
                                205 *--------------------------------
                                206 * Set SDA & SCL high
                                207 * for repeated start
                                208 *--------------------------------
                                209 *
000041 9D18             43      210 I2CBHG  MOV     S1,#PIN+ESO

                                213 *--------------------------------
                                214 * Remember whether the bus was
                                215 * held or released
                                216 *--------------------------------
                                217 *
                                218 * Input: (RA=0) = not held
                                219 *        (R7=0) = successful
                                220 *
000043 B820             44      221 I2CWND  MOV     R0,#BUSHLD
000045 A0               45      222         MOV     @R0,A                   (@BUSHLD > 0) = Held
                                223 *
                                224 *--------------------------------
                                225 * Restore R0-R6 ; Not R7
                                226 *--------------------------------
                                227 *
000046 B869             46      228         MOV     R0,#SAVRG1              Buffer pointer
000048 B905             47      229         MOV     R1,#5                   Register pointer & counter
                                230 *
                                231 *--------------------------------
                                232 * Restore R6 to R2 and then R1,R0
                                233 *--------------------------------
                                234 *
00004A 19               48      235 ISRL2   INC     R1                      Compensate DEC fiddle below
00004B 18               49      236         INC     R0                      Next SAVRG
00004C F0               50      237         MOV     A,@R0                   Get next reg
00004D A1               51      238         MOV     @R1,A                   Save it
00004E C9               52      239         DEC     R1
00004F E94A         R   53      240         DJNZ    R1,ISRL2                Do 5 times R6-R2
000051 B869             54      241         MOV     R0,#SAVRG1              Point to buffer
000053 F0               55      242         MOV     A,@R0
000054 A9               56      243         MOV     R1,A                    Restore R1
000055 C8               57      244         DEC     R0
000056 F0               58      245         MOV     A,@R0                   Get saved R0
000057 A8               59      246         MOV     R0,A                    Replace R0
                                247 *
000058 FF               60      248         MOV     A,R7                    RA = success
000059 85               61      249         EN      SI                      Enable SIO interrupt
00005A 83               62      250 I2CRT1  RET                             EXIT

                                253 *--------------------------------
                                254 * Test read bus status
                                255 *--------------------------------
                                256 *
00005B 1400         R   63      257 I2CRDT  CALL    RBSMT
00005D 968E         R   64      258         JNZ     MERROR                  Jump if error
                                259 *
                                260 *--------------------------------
                                261 * Read data into data block
                                262 *--------------------------------
                                263 *
00005F EC63         R   65      264 I2CRD   DJNZ    R4,I2CRD0               1-n changed to 0-(n-1) & jump if > 0
000061 9E02             66      265         MOV     S2,#2                   SIO = no ACK-8 bit
000063 0C               67      266 I2CRD0  MOV     A,S0                    Dummy read of address
000064 FC               68      267         MOV     A,R4                    JMP if 1 byte RD
000065 C67A         R   69      268         JZ      LASBY0
000067 1400         R   70      269 I2CRD1  CALL    RBSMT                   Test bus status
000069 968E         R   71      270         JNZ     MERROR                  Jump if MST RX error
00006B 2301             72      271         MOV     A,#1
00006D DC               73      272         XRL     A,R4
00006E 9672         R   74      273         JNZ     I2CRD2                  Jump if not penultimate byte
000070 9E02             75      274         MOV     S2,#2                   Set SIO to no acknowledge
000072 FC               76      275 I2CRD2  MOV     A,R4
                                276 *       JZ      LASTBY          >>Is this jump ever active?<< lets see !!
000073 CC               77      277         DEC     R4                      DEC byte count
000074 0C               78      278         MOV     A,S0                    Get byte
000075 A0               79      279         MOV     @R0,A                   Write byte into data block
000076 C8               80      280         DEC     R0                      Point to next block location
000077 FC               81      281         MOV     A,R4
000078 9667         R   82      282         JNZ     I2CRD1                  Repeat for all bytes except 0,1
00007A 1400         R   83      283 LASBY0  CALL    RBSMT                   Test bus status
00007C 53FE             84      284         ANL     A,#.not.LRB             Ignore ACK
00007E 968E         R   85      285         JNZ     MERROR                  JMP if error
                                286 *
000080 9DA9             86      287 LASTBY  MOV     S1,#H'A9'               1 bit byte for -ve ACK
000082 0C               87      288         MOV     A,S0                    Get last data byte
000083 A0               88      289         MOV     @R0,A                   Write last byte into data block
000084 1400         R   89      290 LASBY1  CALL    RBSMT                   Test bus status
000086 9E42             90      291         MOV     S2,#H'42'               Set acknowledge mode
000088 D301             91      292         XRL     A,#LRB                  Invert LRB for -ve ACK
00008A 968E         R   92      293         JNZ     MERROR                  JMP if MST RX error
00008C 0436         R   93      294         JMP     I2CCHL                  Jump to hold control for R/W
```

Master transmitter bus error subroutine

This subroutine MERROR is used when a bus error is detected. This routine sets the acknowledge code and then checks the AAS, AL and MST bits in S1. Depending upon the result of the tests on these bits the program will:

1) AAS = 1, Jump to routine SLAVE

   AAS = 0, Test MST bit

2) MST = 1, Send a stop condition (hardware reset MST = 0)

If MST = 0, then the program will proceed to test the AL bit (arbitration lost), If AL = 1, the device has genuinely lost an arbitration and a re-try is invoked. If AL = 0 an acknowledge is transmitted and the SIO is re-initialized.

```
                                  295
                                  296 *                                                    eject
                                  297 *---------------------------------
                                  298 * MST TX/RX error routine
                                  299 *---------------------------------
                                  300 *
                                  301 * Input : IIC
                                  302 * Output: Error code in RA
                                  303 * Used  :
                                  304 * Kept  :
                                  305 *
                                  332 *
00008E 9E42          94  333 MERROR  MOV     S2,#H'42'          Set ack mode
000090 A9            95  334         MOV     R1,A               Save error code
000091 0D            96  335         MOV     A,S1               Get status
000092 52BA    R     97  336         JB2     ADDSLV             Jump if addressed as slave
000094 F2A5    R     98  337         JB7     I2CSST             Jump if MST=1
000096 72A2    R     99  338         JB3     MERR4              Jump if AL=1
000098 9E02         100  339         MOV     S2,#H'02'          Clear bus method if AL=0
00009A 9E42         101  340         MOV     S2,#H'42'
00009C 9D18         102  341         MOV     S1,#PIN+ESO
00009E 9D18         103  342         MOV     S1,#PIN+ESO
0000A0 04A7    R    104  343         JMP     MERR1
0000A2 0C           105  344 MERR4   MOV     A,S0               Clear bus method if AL=1
0000A3 04A7    R    106  345         JMP     MERR1
0000A5 9DD8         107  346 I2CSST  MOV     S1,#STOPC          Send stop condition
                        347 *
0000A7 B820         108  348 MERR1   MOV     R0,#BUSHLD         Test whether bus was held
0000A9 F0           109  349         MOV     A,@R0
0000AA 96B5    R    110  350         JNZ     MERR2              Jump if hold req. (give up)
0000AC F9           111  351         MOV     A,R1               Get back error code
0000AD D301         112  352         XRL     A,#LRB             Test for no ack error only
0000AF C6B3    R    113  353         JZ      MERR3              Jump if no ack
0000B1 0413    R    114  354         JMP     I2CSUP             Try again without decrementing counter
0000B3 ED13    R    115  355 MERR3   DJNZ    R5,I2CSUP          Jump if tried < R5 times (try again)
0000B5 BFFF         116  356 MERR2   MOV     R7,#255            Set failed flag
0000B7 27           117  357         CLR     A                  Set not held flag
0000B8 0443    R    118  358         JMP     I2CWND             Leave
                        359 *
                        360 *---------------------------------
                        361 * Address as slave handling
                        362 *---------------------------------
                        363 *
0000BA 1400    R    119  364 ADDSLV  CALL    SLAVE              Call slave routine
0000BC 04A7    R    120  365         JMP     MERR1              Test whether to try again or give up
                        366 *
```

I²C bus crash protection subroutine

The aim of this routine is to free the bus in the event of a data clash. The program polls the BB bit until BB is reset to '0' or a timer counts out. If BB=1 after the timer has underflowed, the bus is released by the program re-initializing the SIO interface.

```
                                 369 ************************************************************
                                 370 * Subroutine: I2C bus crash protection:-Bus busy tests.
                                 371 ************************************************************
                                 372 *
                                 373 * Input : I2C
                                 374 * Output: SIO disabled
                                 375 * Used  : R1
                                 376 * Kept  : R0, R2-R7
                                 377 *
                                 378 *       Repeat
                                 379 *          TIME=0
                                 380 *          Repeat
                                 381 *             Read Bus status
                                 382 *          Until (BB=0 or TIME>MAX_BUSY_TIME)
                                 383 *          If (BB=1) then begin {need to free bus}
                                 384 *             Reset BB & Initialise status
                                 385 *             Initialise frequ.
                                 386 *             Send H'FF'
                                 387 *             Repeat
                                 388 *                Read status
                                 389 *             Until (MST=0 or PIN=0)
                                 390 *          End
                                 391 *       Until (MST=1 and TRX=0 and BB=1  and PIN=0 and
                                 392 *              AL=0  and AAS=0 and ADO=0 and LRB=1)
                                 393 *       Send STOP condition
                                 394 *
000000 B9FA          121         399 FREBUS  MOV     R1,#250                 Load bus busy counter (~13ms)
000002 85            122         400         EN      SI                      Enable SIO interrupt (allow for PIN=0)
000003 0D            123         401 BBTST1  MOV     A,S1                    Get status
000004 37            124         402         CPL     A
000005 B223      R   125         403         JB5     BBTST2                  JMP if bus is free
000007 E9Q3      R   126         404         DJNZ    R1,BBTST1               Wait until bus free or waited to long
000009 9D18          127         405         MOV     S1,#PIN+ESO             Reset Bus Busy bit: Bus is free now!
00000B 9D18          128         406         MOV     S1,#PIN+ESO             Iniatialise other status bits
00000D 9E42          129         407         MOV     S2,#IICFR+ACK           Initialise I2C Freq. & ACK bit
00000F 95            130         408         DIS     SI                      Disable SIO interrupt
000010 9CFF          131         409         MOV     S0,#255                 Load SIO data reg.
000012 9DF8          132         410         MOV     S1,#STARTC              O/P Start cond. & H'FF' as Slave addr.
                                 411 *
                                 412 *--------------------------------
                                 413 * Test Bus Status.
                                 414 *--------------------------------
                                 415 *
000014 0D            133         416 FTTBS   MOV     A,S1                    Get Bus status
000015 F219      R   134         417         JB7     FTTBS1                  JMP if MST=1
000017 041D      R   135         418         JMP     MTTCN1                  Continue
000019 9214      R   136         419 FTTBS1  JB4     FTTBS                   JMP if PIN=1
00001B D3A0          137         420         XRL     A,#MST+BB               Test MST Bus status
                                 421 *
00001D D301          138         422 MTTCN1  XRL     A,#LRB                  Invert ACK bit
00001F 9600      R   139         423         JNZ     FREBUS                  JMP if error
000021 9DD8          140         424         MOV     S1,#STOPC               O/P STOP Condition
000023 83            141         425 BBTST2  RET                             EXIT
                                 426 *
```

Test master transmitter status subroutine

This subroutine simply reads the status register S1 and polls the MST and PIN bits until MST or PIN = 0. If MST = 1 when PIN = 0, the Master/Transmitter bit pattern is exclusive OR'd with the present status byte. If the result in the accumulator is zero the status is correct. In either case the error status is retuned via the accumulator.

```
                                 429 ************************************
                                 430 * Subroutine: Test MST TX status
                                 431 ************************************
                                 432 *
                                 433 * Input :
                                 434 * Output: RA=0 if O.K.
                                 435 * Used  :
                                 436 * Kept  : R0-R7
                                 446 *
000000 0D            142         447 TBSMT   MOV     A,S1                    Get bus status
000001 F204      R   143         448         JB7     TBSMT1                  JMP if MST=1
000003 83            144         449         RET                             EXIT
000004 9200      R   145         450 TBSMT1  JB4     TBSMT                   JMP if PIN=1
000006 D3E0          146         451         XRL     A,#MST+TRX+BB           Test MST/TRX status
000008 83            147         452         RET                             EXIT
                                 453 *
```

Test master receiver status subroutine

Same as MST TX status routine with the exception of line 479 which exclusive-OR's only bit pattern MST + BB with status byte.

```
                                454 *
                                455 *
                                456 *
                                457 ************************************
                                458 * Subroutine: Test MST RX status
                                459 ************************************
                                460 *
                                461 * Input :
                                462 * Output: RA=0 if O.K.
                                463 * Used  :
                                464 * Kept  : R0-R7
                                465 *
                                466 *       Repeat
                                467 *          Read Status
                                468 *       Until (MST=0 or PIN=0)
                                469 *       Return TRX+MST+BB if MST=0
                                470 *       Return TRX+NOT(MST+BB) if PIN=0
                                471 *
                                472 RBTSEC  RSECT   ROM,INPAGE
                                473       PAGE    256
                                474 *
000000 0D           148         475 RBSMT   MOV     A,S1                    Get bus status
000001 F204      R  149         476         JB7     RBSMT1                  JMP if MST=1
000003 83          150         477         RET                             EXIT
000004 9200      R  151         478 RBSMT1  JB4     RBSMT                   JMP if PIN=1
000006 D3A0        152         479         XRL     A,#MST+BB               Test MST/RX status
000008 83          153         480         RET                             EXIT
                               481 *
```

SIO interrupt routine

When a genuine serial I/O interrupt is received, the processor enters the following routine. The program selects register bank 1 and saves the value of the accumulator before calling routine SLAVE.

```
                                489 ********************************************
                                490 * SIO Interrupt routine
                                491 ********************************************
                                492 *
                                493 * Input : IIC, TXTREQ
                                494 * Output: WR mss-IBUFL bytes in IBUF & set CMDPRE or SYSPRE
                                495 *         NEWCMD, TUNST1, CMDPRE
                                496 *         SYCMD1, SYCMD2, TUNST2, SYSPRE
                                497 *         RD mss-Byte O/P on IIC
                                498 * Used  : Bank1: R0,R1,R2,R4,R7
                                499 * Kept  : Bank0 & Bank1: RA,R2,R3,R5,R6
                                500 *
                                501 *--------------------------------
                                502 * Interrupt entry to slave routine
                                503 *--------------------------------
                                504 *
                                505 * Input : IIC,
                                506 * Output: R1 = error code = AAS
                                507 *         IIC, NEWCMD, TUNST1, CMDPRE
                                508 *              SYCMD1, SYCMD2, TUNST2, SYSPRE
                                509 * Used  :R1,R2,R4,R7
                                510 * Kept  :RA,R3,R5,R6
                                511 *
                                512 *       Preserve Register A
                                513 *       Call SLAVE
                                514 *
                                515 SINSEC  RSECT   ROM
                                516       PAGE    256
                                517 *
000000 D5           154         518 SIOIN   SEL     RB1
000001 AF           155         519         MOV     R7,A                    Save RA
000002 1400      R  156         520         CALL    SLAVE
000004 FF           157         521         MOV     A,R7                    Restore RA
000005 93           158         522         RETR                            EXIT Interrupt routine (restoring RB0)
```

Slave routine

This subroutine tests the BB and AAS bits in S1. If these bits are set the device is addressed as "slave" and the program goes on to determine whether the device is in the slave transmitter or slave receiver mode. If the program detects that the device has not been addressed as slave but ADO = 1, due to a previous address, then the controller will read the byte and release the bus. After a complete transmission or reception the status of the bus is obtained and a test of BB and PIN is carried out, if BB = 0 the device will exit the program setting the error code. For as long as BB = 1 and PIN = 0 the program will loop back to SLABBL.

Note: This routine can only store bytes received in a single transmission, any previous bytes will be lost.

A psuedo-high level description of the routine precedes the program.

```
525 **********************************
526 * Subroutine: SLAVE
527 **********************************
528 *
529 * Input:  IIC, SLAVE
530 * Output: R1 = error code + AAS
531 *         IIC, NEWCMD, TUNST1, CMDPRE
532 *              SYCMD1, SYCMD2, TUNST2, SYSPRE
533 * Used:   R0,R1,R2,R4 (of current register bank)
534 * Kept:   R3,R5,R6,R7 (of current register bank) & all of other bank
535 *
537 * Routine description:
538 *
539 *       Read status {S1}
540 *       if (BB=0) {self generated interrupt} then
541 *          Do nothing                                        ??
542 *       else begin
543 *          if (AAS=0) {arbitration lost only} then
544 *             Read byte to release SCL {S0}
545 *          else begin {addressed as slave}
546 *             repeat
547 *                Read status {S1}
548 *                if (TRX=0) {receiver} then
549 *                   begin
550 *                      if (AAS=0) {data byte received} then
551 *                         begin
552 *                            if (No. wanted > 0) and (ADO=0) then
553 *                               begin
554 *                                  Read byte {S0}
555 *                                  Store byte
556 *                                  Decrement no. wanted
557 *                                  if (No. wanted = 0) then
558 *                                     Store received packet
559 *                               end
560 *                            else
561 *                               Read byte to release SCL {S0}
562 *                         end
563 *                      else begin {addressed as slave}
564 *                         Read data to start reception {S0}
565 *                         Initialise no. wanted
566 *                      end
567 *                   end
568 *                else begin {transmitter}
569 *                   if (AAS=1) {addressed as slave} then
570 *                      Initialise read pointer
571 *                   if (LRB=0) {ack} then
572 *                      Send data {S0}
573 *                   else {no ack}
574 *                      Quit using bus
575 *                end
576 *                repeat
577 *                   Read status {S1}
578 *                until (BB=0 or PIN=0)
579 *             until (BB=0) {stop condition}
580 *          end
581 *       end
582 *
583 *
584 * Note: The read pointer is not initialised in this routine because
585 *       there is only one byte to send.
```

```
                                588 SLASEC   RSECT   ROM,INPAGE
                                589          PAGE    256
                                590 *
000000 9E42           159       591 SLAVE    MOV     S2,#H'42'        Set ack mode
000002 0D             160       592          MOV     A,S1             Read status
000003 B206     R     161       593          JB5     SLABB            if (BB=0) then
000005 83             162       594          RET                         Self generated interrupt
000006 520A     R     163       595 SLABB    JB2     SLABBL           else if (AAS=0) then
000008 0C             164       596          MOV     A,S0                Release bus
000009 83             165       597          RET
                                598 *
00000A 0D             166       599 SLABBL   MOV     A,S1             else repeat
00000B D227     R     167       600          JB6     SLATX               Read status
00000D 5220     R     168       601          JB2     SLAASR              if (TRX=0) {receiver} then
00000F 321D     R     169       602          JB1     SLARX1                 if (AA0S=0) then
000011 FC             170       603          MOV     A,R4                      if (AD0=0) and
000012 C61D     R     171       604          JZ      SLARX1
000014 0C             172       605          MOV     A,S0                         and (no. wanted > 0) then
000015 A1             173       606          MOV     @R1,A                           Read byte
000016 CC             174       607          DEC     R4                              Save byte
000017 C9             175       608          DEC     R1                              Decrement no. wanted
000018 FC             176       609          MOV     A,R4                            Update destination
000019 9635     R     177       610          JNZ     SLAEVL
00001B 0400     R     178       611          JMP     SLAALL                          if (No. wanted = 0) then
                                612 *                                                   Store packet
00001D 0C             179       613 SLARX1   MOV     A,S0                        else
00001E 0435     R     180       614          JMP     SLAEVL                         Read byte to release SCL
                                615 *
000020 0C             181       616 SLAASR   MOV     A,S0                     else
000021 BC03           182       617          MOV     R4,#IBUFL                   Read byte to start reception
000023 B972           183       618          MOV     R1,#RXBUF0                  Initialise no. wanted
000025 0435     R     184       619          JMP     SLAEVL                      Initialise destination
                                620 *
000027 1232     R     185       621 SLATX    JB0     SLATX1           else {tranBsmitter}
000029 B92B           186       622          MOV     R1,#TXTREQ          if (LRB=0) {ack} then
00002B F1             187       623          MOV     A,@R1
00002C 3C             188       624          MOV     S0,A                        Send byte from TXTREQ
00002D 5305           189       625          ANL     A,#SGNPRE+TXONLY            Clear transient bits
00002F A1             190       626          MOV     @R1,A
000030 0435     R     191       627          JMP     SLAEVL
                                628 *                                        else {no ack}
000032 9D28           192       629 SLATX1   MOV     S1,#BB+ESO                  Quit Busing bus
000034 0C             193       630          MOV     A,S0
                                631 *                                     repeat
000035 0D             194       632 SLAEVL   MOV     A,S1                       Read status
000036 37             195       633          CPL     A
000037 B23D     R     196       634          JB5     SLAEXI                  until (BB=0) or
000039 920A     R     197       635          JB4     SLABBL                        (PIN=0)
00003B 0435     R     198       636          JMP     SLAEVL
                                637 *                                   until (BB=0)
00003D B904           199       638 SLAEXI   MOV     R1,#AAS          Set error code used in I2C to AAS
00003F 83             200       639          RET
```

3.3.3.2 I^2C system level utility routines

These utility routines are representitive of I^2C assembler programs at system level. Taken as examples here are routines for SAA5240 CCT (computer controlled teletext see figure 3.47) and Non-volatile RAM (NVR). Other utility routines listed here deal with problems such as bus transmission errors and status reads. To transmit buffered data from the SIO interface to the addressed device, these subroutines call the I^2C communications routine.

Note: These subroutines are called in whole or in part by other routines.

Store received packet subroutine

This routine stores received bytes in data memory. The number of data bytes in the packet is determined by R4. Register R1 is used as the buffer pointer and on completion the program returns control to the SLAVE subroutine. This subroutine distinguishes between user and system commands. User commands are those commands invoked directly by the user e.g. 'volume up' and 'volume down', system commands could occur at any time and are invisible to the user.

```
                                641 *
                                642 *********************************
                                643 * Store received packet
                                644 *********************************
                                645 *
                                646 * Input : RXBUF*
                                647 * Output: NEWCMD, TUNST2, CMDPRE          {user command}
                                648 *         SYCMD1, SYCMD2, TUNST2, SYSPRE {system command}
                                649 * Used  : R0,R1,R2
                                650 * Kept  : R3-R7
                                651 *
                                652 SLLSEC  RSECT   ROM,INPAGE
                                653         PAGE    256
                                654 *
000000 B972          201       655 SLAALL  MOV     R1,#RXBUF0              RX buffer pointer
000002 F1            202       656         MOV     A,@R1
000003 37            203       657         CPL     A
000004 B828          204       658         MOV     R0,#TUNST2
000006 5213      R   205       659         JB2     MSASYS                  JMP if System Command
                                660 *
                                661 *--------------------------------
                                662 * Message is USER command.
                                663 *--------------------------------
                                664 *
000008 F1            206       665 MSAUSR  MOV     A,@R1                   Get 1st byte
000009 A0            207       666         MOV     @R0,A                   Store 1st byte
00000A C9            208       667         DEC     R1
00000B F1            209       668         MOV     A,@R1                   Get 2nd byte
00000C B82C          210       669         MOV     R0,#NEWCMD              New-command addr.
00000E A0            211       670         MOV     @R0,A                   Store 2nd byte
00000F 2301          212       671         MOV     A,#CMDPRE               Command Present mask
000011 041D      R   213       672         JMP     MSAOUT                  Pre-exit
                                673 *
                                674 *--------------------------------
                                675 * Message is SYSTEM command.
                                676 *--------------------------------
                                677 *
000013 BA03          214       678 MSASYS  MOV     R2,#3                   Loop counter
000015 F1            215       679 MSALP1  MOV     A,@R1                   Get byte from buffer
000016 A0            216       680         MOV     @R0,A                   Write byte to destination
000017 18            217       681         INC     R0                      Point to destination
000018 C9            218       682         DEC     R1                      Point to next source byte
000019 EA15      R   219       683         DJNZ    R2,MSALP1               Until 3 bytes transfered
00001B 2302          220       684         MOV     A,#SYSPRE               SYS Present mask
                                685 *
                                686 *---------------------------------
                                687 * Set appropriate cmd present flag
                                688 *---------------------------------
                                689 *
00001D B830          221       690 MSAOUT  MOV     R0,#CMDSTA              Command status address
00001F 40            222       691         ORL     A,@R0                   Set bit
000020 A0            223       692         MOV     @R0,A                   Replace CMDSTA
000021 0435      R   224       693         JMP     SLAEVL                  Jump back to slave routine
```

Subroutine -to set up I²C buffer and pointer for CCT(SAA5042)

This routine sets up the data buffer used by the I²C routine, and initiates a cursor set-up for the CCT. Cursor or pointer set-up means writing to a register in order to control the destination of following data.

Register R0 is used as pointer for the memory locations used in the buffer. The routine exits via a common exit routine I2CCM1 which transmits the data to the CCT.

```
                                701 *
                                702 ************************************************
                                703 * Subroutine: Sets up IIC buff. to set CCT cursor.
                                704 ************************************************
                                705 *
                                706 * Input : R5=Chapt. no.,R6=Row no.,R7=Column no.
                                707 * Output: RA=CCT WR addr.,TEMP0-TEMP4
                                708 * Used  : R0
                                709 * Kept  :
                                710 *
                                711 *       Write into buffer:  {TEMP0}
                                712 *          4 byte mess & no_hold
                                713 *          CCT sub-address=8
                                714 *          CCT Active Chapter from R5
                                715 *          CCT Row from R6
                                716 *          CCT Col from R5
                                717 *       Point to CCT and send to IIC  {routine I2C}
                                718 *
                                719 ICCSEX  RSECT   ROM
                                720 *
000000 B87F          225       721 ICCS8   MOV     R0,#TEMP0              Pointer
000002 B004          226       722         MOV     @R0,#4                 Write no hold & 4 byte MS
000004 C8            227       723         DEC     R0
000005 B008          228       724         MOV     @R0,#8                 CCT Sub-addr to buff.
000007 C8            229       725         DEC     R0
000008 FD            230       726         MOV     A,R5                   CCT Active chapt.
000009 A0            231       727         MOV     @R0,A                  to buff.
00000A C8            232       728         DEC     R0
00000B FE            233       729         MOV     A,R6                   CCT Row no.
00000C A0            234       730         MOV     @R0,A                  to buff
00000D C8            235       731         DEC     R0
00000E FF            236       732         MOV     A,R7                   CCT Column no.
00000F 041F      R   237       733         JMP     I2CCM1                 Jump to common exit
```

Subroutine -writes to CCT (SAA5042) register1

This subroutine writes a 2-byte message plus a 'no-hold' bit to register 1 of the CCT chip. Register R0 is used as a pointer, and the data is transmitted via the common exit routine I2CCM1.

Note: Registers R1 to R10 of the CCT are write only, R11 is read/write.

```
                                736 *********************************
                                737 * Subroutine: Write CCT R1.
                                738 *********************************
                                739 *
                                740 * Input : RA=Data
                                741 * Output: I2C
                                742 * Used  : R0,R7
                                743 * Kept  : R1-R6
                                744 *
                                745 *       Write to buffer:  {TEMP0}
                                746 *          2 byte message & no_hold
                                747 *          CCT sub_address = 1
                                748 *          data from RA
                                749 *       Point to CCT and send to IIC
                                750 *
                                751 ICRSEX  RSECT   ROM
                                752 *
000000 B87F          238       753 ICWR1   MOV     R0,#TEMP0              IIC buff. pointer
000002 B002          239       754         MOV     @R0,#2                 Write no hold & 2 byte msg
000004 C8            240       755         DEC     R0
000005 B001          241       756         MOV     @R0,#1                 CCT Reg 1 sub-addr
000007 C8            242       757         DEC     R0
000008 041F      R   243       758         JMP     I2CCM1                 Jump to common exit
```

Subroutine -writes to CCT (SAA5042) registers 4-7

This routine writes 5-bytes plus a 'no-hold' bit to the I^2C buffer. Register R0 is used as the buffer pointer and R1 is used as CCT register map pointer.Value TEMP0 is the initial buffer pointer and the message is transmitted by the common exit routine I2CCM1.

```
                                761 ********************************
                                762 * Subroutine: Write CCT Reg 4-7
                                763 ********************************
                                764 *
                                765 * Input : CCTRG5-CCTRG7,DISPG5(bits5-4)
                                766 * Output: I2C
                                767 * Used  : R0,R1,R7
                                768 * Kept  : R2-R6
                                769 *
                                770 *       Write to buffer:   {TEMP0}
                                771 *          5 byte message + no_hold
                                772 *          CCT Sub_address = 4
                                773 *          Display chapter from DISPG5
                                774 *          data from CCTRG5-7
                                775 *       Point to CCT & send to IIC
                                776 *
                                777 ICRSEC  RSECT   ROM
                                778 *
                                779 *--------------------------------
                                780 * Set up msg length & sub-addr.
                                781 *--------------------------------
                                782 *
000000 B87F          244       783 ICWR47  MOV     R0,#TEMP0              R0 = IIC buff pointer
000002 8005          245       784         MOV     @R0,#5                 Write no hold & 5 byte msg
000004 C8            246       785         DEC     R0
000005 8004          247       786         MOV     @R0,#4                 CCT Reg Sub-addr
000007 C8            248       787         DEC     R0
                               788 *
                               789 *--------------------------------
                               790 * Set up msg in IIC buff.
                               791 *--------------------------------
                               792 *
000008 B94B          249       793         MOV     R1,#DISPG5             Display chapt. pointer
00000A 2330          250       794         MOV     A,#H'30'               Disp. chap. mask
00000C 51            251       795         ANL     A,@R1
00000D 47            252       796         SWAP    A                      RA=Display chapt. no.
00000E A0            253       797         MOV     @R0,A                  CCT R4 data
00000F C8            254       798         DEC     R0
000010 B94C          255       799         MOV     R1,#CCTRG5             R5 map pointer
000012 F1            256       800         MOV     A,@R1                  Get R5 data
000013 A0            257       801         MOV     @R0,A                  CCT R5 to buff
000014 C8            258       802         DEC     R0
000015 19            259       803         INC     R1                     R6 map pointer
000016 F1            260       804         MOV     A,@R1                  Get R6 data
000017 A0            261       805         MOV     @R0,A                  CCT R6 to buff
000018 C8            262       806         DEC     R0
000019 19            263       807         INC     R1                     R7 map pointer
00001A F1            264       808         MOV     A,@R1                  Get R7 data
00001B 041F      R   265       809         JMP     I2CCM1                 Jump to common exit
```

Subroutine -writes page request to CCT

This routine writes to the buffer a 9-byte message plus a 'no-hold' bit. R1 is used to set up the message length, R0 contains the teletext page number. The routine sets up the bit pattern for the CCT register 2 page request byte.

```
                                812 ******************************************************
                                813 * Write Page Request to CCT.
                                814 *       Page DO CARE=R7;Sub-code DO CARE=*PG5.SUBDIS
                                815 ******************************************************
                                816 *
                                817 * Input :R0=Page No. block(DISPG5 or BACPG5),R7=Page Do Care mask
                                818 * Output: I2C
                                819 * Used  : R0-R2,R7
                                820 * Kept  : R3-R6
                                821 *
                                822 *       Write to buffer:   {TEMP0}
                                823 *          9 byte message & no_hold
                                824 *          CCT sub_address = 2
                                825 *          8 nibbles from Page block + Do_care bit
                                826 *       Point to CCT & send to IIC
                                827 *
                                828 R23SEC  RSECT   ROM,INPAGE
                                829         PAGE    256
                                830 *
```

```
                                831 *-----------------------------------------------------
                                832 * Set up Msg length & sub-addr. & get sub-code DO CARE
                                833 *-----------------------------------------------------
                                834 *
000000 BF10           266       835 ICWR23  MOV     R7,#DOCARE              Entry with do care always
                                836 *
000002 B97F           267       837 ICR23D  MOV     R1,#TEMP0               IIC buffer pointer
000004 B109           268       838         MOV     @R1,#9                  Write no hold & 9 byte msg
000006 C9             269       839         DEC     R1
000007 B102           270       840         MOV     @R1,#2                  CCT R2 sub-addr
000009 C9             271       841         DEC     R1
00000A 2380           272       842         MOV     A,#SUBDIS               Sub-code DO CARE
00000C 50             273       843         ANL     A,@R0                   DO CARE only
00000D 47             274       844         SWAP    A
00000E E7             275       845         RL      A                       DO CARE now in bit4
00000F AA             276       846         MOV     R2,A                    Save in R2
                                847 *
                                848 *-------------------------------------
                                849 * Set up Chapt & Magazine in IIC buff.
                                850 *-------------------------------------
                                851 *
000010 2330           277       852         MOV     A,#H'30'                Chapt. mask
000012 50             278       853         ANL     A,@R0                   RA=CCT Reg2 data
000013 1405     R     279       854         CALL    ACNBL2                  Act on nibble

                                857 *-----------------------------------------------
                                858 * Set  up Page tens & units.
                                859 *-----------------------------------------------
                                860 *
000015 1400     R     280       861         CALL    ACNBL1                  Act on nibble
                                862 *
                                863 *--------------------------------
                                864 * Set up Hours/Mins tens & units
                                865 *--------------------------------
                                866 *
000017 FA             281       867         MOV     A,R2                    Get sub-code DO CARE
000018 AF             282       868         MOV     R7,A                    Put in R7
000019 1400     R     283       869         CALL    ACNBL1                  Act on nibble
00001B 1400     R     284       870         CALL    ACNBL1                  Act on nibble
00001D 0420     R     285       871         JMP     I2CCM2                  Jump to common exit
```

Subroutine -get display chapter number

This subroutine extracts bits 4 and 5 from displayed chapter flag byte and returns them as bits 0 & 1 of R5.

```
                                874 ************************************
                                875 * Subroutine: Get display Chapt. No.
                                876 ************************************
                                877 *
                                878 *
                                879 * Input : R0->Byte containing chapter no.(DISPG5/BACPG5)
                                880 * Output: R5=Display Chapt.No.
                                881 * Used  : R0,R5
                                882 * Kept  : R1-R4,R6,R7
                                883 *
                                886 *
                                887 IDCSEC  RSECT   ROM
                                888 *
000000 2330          286       889 ICWDCN  MOV     A,#H'30'              Disp Chapt. mask
000002 50            287       890         ANL     A,@R0
000003 47            288       891         SWAP    A                     RA=Disp. Chapt. No.
000004 AD            289       892         MOV     R5,A                  Put in R5
000005 83            290       893         RET                           EXIT
```

Subroutine -unbox time to CCT

This subroutine writes 3-bytes plus a 'no-hold' bit to the buffer pointered by R0. This routine calls another subroutine; SET030, which sets up the cursor to Ch(DISP), Row(0), Col(30). The data is transmitted to the CCT via the common exit routine I2CCM1.

```
                                895 *
                                896 ********************************
                                897 * Subroutine: Unbox Time to CCT
                                898 ********************************
                                899 *
                                900 * Input : R0C30,R0C31
                                901 * Output: I2C
                                902 * Used  : R0,R1,R5-R7
                                903 * Kept  : R2-R4
                                904 *
                                905 *       Set cursor to (0,30)   {Routine SET030}
                                906 *       Write to buffer:   {TEMP0}
                                907 *          3 bytes + no_hold
                                908 *          CCT sub_address = 11
                                909 *          2 bytes R0C30/31
                                910 *       Point to CCT & send to IIC
                                911 *
                                912 ICUSEC  RSECT   ROM
                                913 *
                                914 *-----------------------------------------------
                                915 * Set up cursor pos. to Ch(DISP),Row(0),Col(30)
                                916 *-----------------------------------------------
                                917 *
000000 1400      R     291      918 ICWUBT  CALL    SET030                Common bit of code 1
                                919 *
                                920 *--------------------------------
                                921 * Write R0C30,R0C31 to CCT
                                922 *--------------------------------
                                923 *
000002 B87F          292       924         MOV     R0,#TEMP0             IIC buff pointer
000004 B003          293       925         MOV     @R0,#3                Write no hold & 3 byte msg
000006 C8            294       926         DEC     R0
000007 B008          295       927         MOV     @R0,#11               CCT sub addr
000009 C8            296       928         DEC     R0                    Set destination pointer
00000A B925          297       929         MOV     R1,#R0C30             Set source pointer
00000C F1            298       930         MOV     A,@R1                 Copy 1st byte
00000D A0            299       931         MOV     @R0,A
00000E C8            300       932         DEC     R0
00000F 19            301       933         INC     R1
000010 F1            302       934         MOV     A,@R1                 Copy 2nd byte
000011 A0            303       935         MOV     @R0,A
000012 041F      R   304       936         JMP     I2CCM1                Jump to common exit
```

Subroutine -box time to CCT

This routine begins by setting the cursor calling SET030, then a 2-byte message with 'no-hold' is sent for a read of the CCT. R1 is js used as the bus buffer pointer and the read message is sent by calling the I^2C bus communication routine. When reading, R0 is used as the destination pointer for the received data and R1 is the source pointer, the two received bytes are copied into data memory. The cursor is then reset and two 'Open-Box' bytes are sent to the CCT.

```
                                939 ************************************************************
                                940 * Subroutine: Box Time to CCT,save current vals to R0C30,31
                                941 ************************************************************
                                942 *
                                943 * Input : I2C
                                944 * Output: I2C,R0C30,R0C31
                                945 * Used  : R0,R1,R5-R7
                                946 * Kept  : R2-R4
                                947 *
                                948 *       Set cursor to (0,30)   {Routine SET030}
                                949 *       Write to buffer:   {TEMP0}
                                950 *          2 byte message + no_hold
                                951 *          CCT sub_address = 2
                                952 *       Point to CCT + READ & send to IIC
                                953 *       Store same two bytes in R0C30/31
                                954 *       Reset cursor to (0,30)  {Routine SET030}
                                955 *       Write to buffer:   {TEMP0}
                                956 *          3 byte message + no_hold
                                957 *          CCT sub_address = 11
                                958 *          2 x OPEN_BOX
                                959 *       Point to CCT & send to IIC
                                960 *
                                961 ICBSEC  RSECT   ROM
                                962 *
                                963 *--------------------------------
                                964 * Set up CCT cursor position
                                965 *--------------------------------
                                966 *
000000 1400      R   305       967 ICWBTI  CALL    SET030
                                968 *
                                969 *--------------------------------
                                970 * Send RD 2 byte msg
                                971 *--------------------------------
                                972 *
000002 B97F          306       973         MOV     R1,#TEMP0               IIC buff. pointer
000004 B102          307       974         MOV     @R1,#2                  Write no hold & 2 byte msg
000006 2323          308       975         MOV     A,#CCTAD+RD             CCT IIC addr & RD
000008 1400      R   309       976         CALL    I2C                     Send msg & RD (keep R1)
                                977 *
                                978 *--------------------------------
                                979 * Save read bytes in R0C30/R0C31
                                980 *--------------------------------
                                981 *
00000A B825          310       982         MOV     R0,#R0C30               Set destination pointer
00000C C9            311       983         DEC     R1                      Set source pointer
00000D F1            312       984         MOV     A,@R1                   Copy 1st byte
00000E A0            313       985         MOV     @R0,A
00000F 18            314       986         INC     R0
000010 C9            315       987         DEC     R1
000011 F1            316       988         MOV     A,@R1                   Copy 2nd byte
000012 A0            317       989         MOV     @R0,A


                                992 *--------------------------------
                                993 * Set up cursor pos. again
                                994 *--------------------------------
                                995 *
000013 1400      R   318       996         CALL    SET030                  Common bit of code 1
                                997 *
                                998 *--------------------------------
                                999 * Write Open Box twice to CCT
                               1000 *--------------------------------
                               1001 *
000015 B87F          319       1002        MOV     R0,#TEMP0               IIC buff pointer
000017 B003          320       1003        MOV     @R0,#3                  Write no hold & 3 bytes
000019 C8            321       1004        DEC     R0
00001A B008          322       1005        MOV     @R0,#11                 CCT sub-addr
00001C C8            323       1006        DEC     R0
00001D B008          324       1007        MOV     @R0,#H'08'              Open Box
00001F C8            325       1008        DEC     R0
000020 B008          326       1009        MOV     @R0,#H'08'              Open Box
000022 2322          327       1010        MOV     A,#CCTAD                CCT IIC addr.
000024 1400      R   328       1011        CALL    I2C                     Send msg
000026 83            329       1012        RET                             EXIT
```

Multiple CCT cursor set-up subroutine

This routine sets up three separate CCT pointers for entry into other routines. The program calls external routines ICWDCN and ICCS8.

```
                        1014 *
                        1015 ***********************************
                        1016 * Multiple entry subroutine:
                        1017 *       SET030, SETCUR, ZERCUR
                        1018 *       Set CCT cursor position
                        1019 ***********************************
                        1020 *
                        1021 * SET030:               ZERCUR:                 SETCUR:
                        1022 *
                        1023 * Input :               Input: R6 = row         Input: R6 = row, R7 = col
                        1024 * Output: CCT cursor    Output: CCT cursor      Output: CCT cursor
                        1025 * Used  : R0,R6,R7      Used: R0,R7             Used: R0
                        1026 * Kept  :
                        1027 *
                        1028 *       Set cursor to :
                        1029 *          (0,30)    {SET030}
                        1030 *          (@R6,0)   {ZERCUR}
                        1031 *          (@R6,@R5) {SETCUR}
                        1032 *
                        1033 SC2SEC  RSECT   ROM
                        1034 *
000000 BF1E      330    1035 SET030  MOV     R7,#30                  CCT col 30
000002 BE00      331    1036         MOV     R6,#0                   CCT row 0
000004 0408    R 332    1037         JMP     SETCUR
                        1038 *
000006 BF00      333    1039 ZERCUR  MOV     R7,#0                   Set Column 0
                        1040 *
000008 B84B      334    1041 SETCUR  MOV     R0,#DISPG5              Pointer for ICWDCN
00000A 1400    R 335    1042         CALL    ICWDCN                  Get display chapt. no.
00000C 1400    R 336    1043         CALL    ICCS8                   Set up IIC buffer
00000E 83        337    1044         RET                             EXIT
```

Subroutine -read control bits of CCT

First, the cursor is initialized and the program calls ICWDCN, ICCS8, these subroutines obtain the chapter number and set up the I^2C buffer respectively. The CCT is addressed for a RD operation and 10 bytes from the CCT are read into memory locations TEMP1 to TEMP10.

```
                                1047 ********************************************************
                                1048 * Subroutine: Read Control Bits of CCT (Row25,Col0-25)
                                1049 ********************************************************
                                1050 *
                                1051 * Input : I2C, R0 -> Chapt. to be read.
                                1052 * Output: TEMP1-TEMP10
                                1053 * Used  : R0, R1, R5-R7
                                1054 * Kept  : R2-R4
                                1055 *
                                1062 *
                                1063 ICYSEX  RSECT   ROM
                                1064 *
                                1065 *---------------------------------
                                1066 * Set up CCT Cursor position
                                1067 *---------------------------------
                                1068 *
000000 BE19          338        1069 ICRCB   MOV     R6,#25              CCT Row 25 in R6
000002 BF00          339        1070         MOV     R7,#0               CCT Col 0 in R7
000004 1400    R     340        1071         CALL    ICWDCN              Get chapter no. into R5
000006 1400    R     341        1072         CALL    ICCS8               Set up IIC buffer for cursor write
                                1073 *
                                1074 *---------------------------------
                                1075 * Send Read 10 byte Msg.
                                1076 *---------------------------------
                                1077 *
000008 B97F          342        1078         MOV     R1,#TEMP0           IIC buff pointer
00000A B10A          343        1079         MOV     @R1,#10             Write no hold & 10 byte msg
00000C 2323          344        1080         MOV     A,#CCTAD+RD         CCT IIC addr + RD
00000E 1400    R     345        1081         CALL    I2C                 Get 10 bytes into TEMP1-10
000010 83            346        1082         RET                         EXIT
```

Subroutine -write to NVR (Non-Volatile RAM) 1 or 2 bytes

This routine uses R1 as a buffer pointer and R2 as the byte counter. The program begins by setting up the message length and NVR address in the I^2C buffer. Next, data bytes to be transmitted to the NVR are placed in the buffer using R1 again, and the complete package is sent to the CCT by calling the I^2C routine. If only one byte of data is to be sent then the program jumps at line 1126 to INVWR1.

```
                                1085 ********************************************
                                1086 * Subroutine: Write to NVR(am) 1 or 2 bytes
                                1087 ********************************************
                                1088 *
                                1089 * Input : @R0,@R0-1=Data1,2;R2=No.of bytes(1or2 only)
                                1090 *         R3=NVR sub-addr.
                                1091 * Output: I2C
                                1092 * Used  : R0-R3,R7
                                1093 * Kept  : R4-R6
                                1094 *
                                1095 *       Write to buffer:   {TEMP0}
                                1096 *          no of bytes = @R2+1  & no_hold
                                1097 *          NVR susb_address from R3
                                1098 *          data bytes from @R0 & @(R0-1)
                                1099 *       Point to NVRAM address from R3 & send to IIC
                                1100 *
                                1101 *
                                1102 IVWSEC  RSECT   ROM,INPAGE
                                1103         PAGE    256
                                1104 *
                                1105 *---------------------------------------
                                1106 * Put msg length & sub-addr in IIC buff.
                                1107 *---------------------------------------
                                1108 *
000000 B97F          347        1109 INVWR   MOV     R1,#TEMP0           IIC buff pointer
000002 FA            348        1110         MOV     A,R2                No. of bytes
000003 17            349        1111         INC     A                   Msg length=sub.addr.+data
000004 CA            350        1112         DEC     R2                  R2=0 or 1
000005 A1            351        1113         MOV     @R1,A               Write no hold & RA byte msg.
000006 C9            352        1114         DEC     R1
000007 FB            353        1115         MOV     A,R3                Get NVR sub-addr
000008 A1            354        1116         MOV     @R1,A               To IIC buff
000009 C9            355        1117         DEC     R1
```

```
                        1118 *
                        1119 *--------------------------------
                        1120 * Put data in IIC buff. & send msg
                        1121 *--------------------------------
                        1122 *
00000A F0               356  1123          MOV     A,@R0          Get 1st data byte
000008 A1               357  1124          MOV     @R1,A          Put in IIC buff.
00000C FA               358  1125          MOV     A,R2           No. of data bytes -1
00000D C613       R     359  1126          JZ      INVWR1         JMP if 1 data byte
00000F C9               360  1127          DEC     R1
000010 C8               361  1128          DEC     R0
000011 F0               362  1129          MOV     A,@R0          Get 2nd data byte
000012 A1               363  1130          MOV     @R1,A          Put in buff
000013 23A0             364  1131 INVWR1   MOV     A,#NVRAD       NVR IIC addr.
000015 1400       R     365  1132          CALL    I2C            Send msg.
000017 83               366  1133          RET                    EXIT
```

Subroutine -read from NVR 1 or 2 bytes

The buffer pointer R0 is initialized and a 1-byte message with a hold bit is written to the I^2C buffer. Next, the buffer pointer is decremented and the NVR address is buffered. This complete packet is sent to the CCT followed by a a 'no-hold' signal, the CCT is addressed with RD and 1 or 2 bytes are read into the microcontroller's data memory using R1 as the destination pointer.

```
                                1135 *
                                1136 ********************************************
                                1137 * Subroutine: Read 1 OR 2 bytes from NVR(am)
                                1138 ********************************************
                                1139 *
                                1140 * Input : I2C, R3=NVR sub-addr, (R4=0)=1 Byte
                                1141 * Output: @R1,@R1-1(2 bytes) OR R3(1 byte)
                                1142 * Used  : R0,R1,R3,R4,R7
                                1143 * Kept  : R2,R5-R6
                                1144 *
                                1145 * Routine description:
                                1146 *
                                1147 *       Write to buffer:   {TEMP0}
                                1148 *          1 byte message + hold
                                1149 *          NVRAM sub_address from R3
                                1150 *       Point to NVRAM address & send to IIC
                                1151 *       Write to buffer:    {TEMP0}
                                1152 *          2 byte message + no_hold
                                1153 *       Point to NVRAM address + READ & send to IIC
                                1154 *       Transfer from buffer:  {TEMP0}
                                1155 *          @R4 bytes  to  @R1
                                1156 *
                                1157 INRSEC  RSECT   ROM,INPAGE
                                1158         PAGE    256
                                1159 *
                                1160 *-------------------------------------
                                1161 * Set up & send NVR sub-addr & hold bus.
                                1162 *-------------------------------------
                                1163 *
000000 B87F          367        1164 INVRD   MOV     R0,#TEMP0              IIC buff pointer
000002 B081          368        1165         MOV     @R0,#H'81'             Write Hold & 1 byte msg
000004 C8            369        1166         DEC     R0
000005 FB            370        1167         MOV     A,R3                   NVR sub-addr
000006 A0            371        1168         MOV     @R0,A                  Addr to buffer
000007 23A0          372        1169         MOV     A,#NVRAD               NVR IIC addr + WR
000009 1400      R   373        1170         CALL    I2C                    Send msg

                                1173 *---------------------------------
                                1174 * Set up & send 2 byte Read of NVR
                                1175 *---------------------------------
                                1176 *
00000B 18            374        1177         INC     R0                     R0 -> TEMP0
00000C B002          375        1178         MOV     @R0,#2                 Write no hold & 2 byte msg
00000E C8            376        1179         DEC     R0                     R0 -> TEMP1
00000F 23A1          377        1180         MOV     A,#NVRAD+RD            NVR IIC addr + RD
000011 1400      R   378        1181         CALL    I2C                    Get bytes from NVR
000013 F0            379        1182         MOV     A,@R0                  Get 1st byte at TEMP1
                                1183 *
                                1184 *-------------------------------
                                1185 * Return of 1 OR 2 bytes.
                                1186 *-------------------------------
                                1187 *
000014 AB            380        1188         MOV     R3,A                   Save byte
000015 FC            381        1189         MOV     A,R4                   1 or 2 bytes
000016 C61E      R   382        1190         JZ      INVRDX                 JMP if 1 byte
000018 FB            383        1191         MOV     A,R3                   Replace 1st byte
000019 A1            384        1192         MOV     @R1,A                  Put 1st byte
00001A C8            385        1193         DEC     R0                     R0 -> TEMP2
00001B C9            386        1194         DEC     R1                     Decrement destination pointer
00001C F0            387        1195         MOV     A,@R0                  Get 2nd byte
00001D A1            388        1196         MOV     @R1,A                  Put 2nd byte
00001E 83            389        1197 INVRDX  RET                            EXIT
                                1198 *
                                1199 *
                                1200 *
                                1201 *
                                1202 *
                                1203 *-------------------------------
                                1204 * Common exit routine
                                1205 *-------------------------------
                                1206 *
00001F A0            390        1207 I2CCM1  MOV     @R0,A                  To IIC buff
000020 2322          391        1208 I2CCM2  MOV     A,#CCTAD               CCT IIC addr
000022 1400      R   392        1209         CALL    I2C                    Send msg
000024 83            393        1210         RET
```

Multi-entry subroutine act on nibbles

This subroutine is called in the CCT page request routine described above. The routine manipulates the top and bottom nibbles of the CCT page data byte in order to place the 'do-care' bits into the bit pattern.

```
                          1211                                          eject
                          1212 *
                          1213 *********************************
                          1214 * Multi-entry subroutine:
                          1215 *       Act on Nibbles
                          1216 *********************************
                          1217 *
                          1218 * Input : R7=docare, R1 -> iic buffer
                          1219 * Output: iic buffer, R0, R1
                          1220 * Used  :
                          1221 * Kept  :
                          1222 *
                          1223 ACNSEC  RSECT   ROM
                          1224 *
                          1225 *--------------------------------
                          1226 * Subr: Act on nibble
                          1227 *--------------------------------
                          1228 *
000000 23F0        394    1229 ACNBL1  MOV     A,#H'F0'              MS nibble1
000002 50          395    1230         ANL     A,@R0                 Get MS nibble
000003 47          396    1231         SWAP    A
000004 6F          397    1232         ADD     A,R7                  Add do care & MS nibble
000005 A1          398    1233 ACNBL2  MOV     @R1,A                 To IIC buffer
000006 C9          399    1234         DEC     R1
000007 230F        400    1235         MOV     A,#H'0F'              LS nibble
000009 50          401    1236         ANL     A,@R0                 Get LS nibble
00000A 6F          402    1237         ADD     A,R7                  Add do care & LS nibble
00000B A1          403    1238         MOV     @R1,A                 To IIC buff
00000C C9          404    1239         DEC     R1
00000D C8          405    1240         DEC     R0
00000E 83          406    1241         RET
                          1242 *
00000F                    1243         END                           End of File

  NO ERRORS DETECTED
```

Symbol definition

The following is a list of all global equates used in the previous routines.

```
SYCMD1   EQU     H'29'          First system command byte
SYCMD2   EQU     H'2A'          Second system command byte
TXTREQ   EQU     H'2B'          Text uP request byte
NEWCMD   EQU     H'2C'          New command input buffer
DISPG5   EQU     H'48'          Magazine/chapter related data of forgrnd page
CCTRG5   EQU     H'4C'          CCT register 5 copy
CMDSTA   EQU     H'30'          Command status byte
IBUFL    EQU     3              Length of interrupt buffer
BUSHLD   EQU     H'20'          Bus held/not held flag
HLD      EQU     H'80'          Mask for hold bit
I2CF     EQU     H'02'          88.7 kHz IIC frequency
LRB      EQU     1              Last Received Bit
CCTAD    EQU     34             CCT address
NVRAD    EQU     H'A0'          NVRAM address
TXONLY   EQU     H'01'          Bit 0 = Text displayed only
BLPREQ   EQU     H'02'          Bit 1 = Bleep
SGNPRE   EQU     H'04'          Bit 2 = Signal present (from PL signal)
SUBDIS   EQU     H'80'          Bit 7 = Subcode displayed for display page
DOCARE   EQU     H'10'          Bit 4 = Do care bit in CCT
CMDPRE   EQU     H'01'          Bit 0 = Command Present
SYSPRE   EQU     H'02'          Bit 1 = System command present

*    for IIC routines:
*
AAS      EQU     H'04'          AAS bit
ACK      EQU     H'40'          ACK bit
BB       EQU     H'20'          BB bit
ESO      EQU     H'08'          ESO bit
IICFR    EQU     H'02'          IIC Bus Frequ.
MST      EQU     H'80'          MST bit
PIN      EQU     H'10'          PIN bit
STARTC   EQU     H'F8'          START condition for S1
STOPC    EQU     H'D8'          STOP codition for S1
TRX      EQU     H'40'          TRX bit
DLEN     EQU     4              Port 22

SAVRG0   EQU     H'68'          Save R0      Save registers R0 - R7 for
SAVRG1   EQU     H'69'          Save R1      I2C subroutine
SAVRG2   EQU     H'6A'          Save R2
SAVRG3   EQU     H'6B'          Save R3
SAVRG4   EQU     H'6C'          Save R4
SAVRG5   EQU     H'6D'          Save R5
SAVRG6   EQU     H'6E'          Save R6
RXBUF3   EQU     H'6F'
*
* 70 - 7F
*
RXBUF2   EQU     H'70'
RXBUF1   EQU     H'71'            - IIC interrupt buffer
RXBUF0   EQU     H'72'
TEMP12   EQU     H'73'
TEMP11   EQU     H'74'
TEMP10   EQU     H'75'
TEMP9    EQU     H'76'
TEMP8    EQU     H'77'
TEMP7    EQU     H'78'            - IIC buffer
TEMP6    EQU     H'79'
TEMP5    EQU     H'7A'
TEMP4    EQU     H'7B'
TEMP3    EQU     H'7C'
TEMP2    EQU     H'7D'
TEMP1    EQU     H'7E'
TEMP0    EQU     H'7F'
*
```

4.0 SERIAL I/O SOFTWARE EXAMPLES: APPLICABLE TO MAB8048

4.1 Single-master routines

4.1.1 Master transmitter routine (MAB8048 - SAA1300)

This software is intended for the MAB8048 microcomputer family which is the only master in an I^2C-bus system. In this system none of the slaves are allowed to stretch the clock low time. This software example only meets the I^2C-bus protocol functionally, electrically there are some differences.

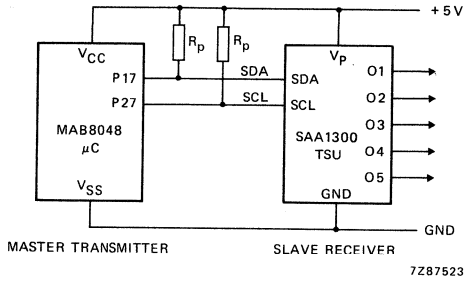


Fig. 4.1 Block diagram of MAB8048 - SAA1300 configuration.

Initialization:

```
000000                    61          ORG     H'000'
                          62 *
000000 2400          1    63 RESET    JMP     INIT            JUMP TO INITIALIZE ROUTINE
                          64 *
                          65 * INITIALISATION:
                          66 *
000002                    67          ORG     H'100'
                          68 *
000100 B8FF          2    69 INIT     MOV     R0,#H'FF'       LOAD DATA TO BE WRITTEN TO TSU
                          70 *
```

Main program:

The main program transmits data to TSU by calling the subroutine WTSU. If the transmission is not successful it repeats the data transfer, otherwise it increments the data and starts again.

The format of the data transfer to TSU is shown in Fig. 4.2:

S	TSU ADR	W	A	DATA BYTE	A	P

Fig. 4.2 Data format of transfer to TSU

S	is the START condition
TSU ADR	is the slave address of TSU
W	is the read/write bit in write state (=0)
A	is the acknowledge bit; this has to be 0
P	is the STOP condition

The slave address of TSU is 0100 0XX

```
TSUAD    EQU  H'40'
```

I^2C-bus signals of MAB8048 are:

```
SDA EQU  H'80'      Pin P17 Pin nr. 34
SCL EQU  H'80'      Pin P27 Pin nr. 38
```

Note: The bus outputs SCL and SDA of the MAB8048 do not have an open drain output as specified in the I^2C-bus protocol.

The logic levels of the MAB8048 bus signals do not match the I^2C-bus protocol:

	I^2C-bus protocol	MAB8048
VILmax	1,5 V	0,8 V
VIHmin	3,5 V	2,0 V
VOLmax	0,4 V @ 3 mA	0,45 V @ 1,6 mA

Because the MAB8048 has an internal large and small pull-up, two pins of different I/O ports (Ports 1 and 2) are taken for the I^2C-bus signals.

If the MAB8048 inputs the acknowledge bit, it has to generate a clock pulse at output SCL. This is done with the ORL Px,#SCL and the ANL Px, #.NOT.SCL instructions. While changing the output latch of the SCL pin the MAB8048 also updates the data of the other I/O pins of the same port.

Because the output latch of the SDA output is switched high, the large pull-up of this output latch of the SDA output will be activated. If the slave pulls the SDA line low to generate an acknowledge, a short circuit situation occurs. To overcome this problem two I/O pins of different ports are defined as the I^2C-bus signals SDA and SCL.

The maximum MAB8048 crystal frequency for this software example is 6 MHz to reach the minimum SCL clock low and high times as given in the I^2C-bus protocol. If a higher crystal frequency is chosen some delays (e.g. NOP instructions) have to be inserted in the master transmitter data (MTDAT) subroutine.

```
                          71 * MAIN PROGRAM:
                          72 *
000102 18            3    73 MAIN   INC    R0          INCR. DATA TO BE TRANSMITTED
                          74 *
000103 5400          4    75 MAIN1  CALL   WTSU        TRANSMIT DATA BYTE IN R0 TO TSU
000105 9603          5    76        JNZ    MAIN1       REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000107 BA00          6    77        MOV    R2,#0       SET DELAY COUNTER
000109 EA09          7    78        DJNZ   R2,$        DELAY
00010B 2402          8    79        JMP    MAIN        CONTINUE
                          80 *
00010D                    81        ORG    H'200'
```

Master writes one data byte to TSU (SAA1300) subroutine:

```
entry: R0 contents to be written to TSU
exit: A = 0 if transmission is successful
contents of register R2 are modified.
```

```
                          87 *
000200 2340                9  88 WTSU   MOV     A,#TSUAD        LOAD TSU SLAVE ADDRESS AND WRITE BIT
000202 5410               10  89        CALL    START           OUTPUT START CONDITION, SLAVE ADDRESS AND
                              90 *                                WRITE BIT; INPUT ACKNOWLEDGE BIT
000204 9609               11  91        JNZ     STOP            JUMP IF NO ACKNOWLEDGE RECEIVED
000206 F8                 12  92        MOV     A,R0            FETCH DATA BYTE TO BE TRANSMITTED
000207 5414               13  93        CALL    MTDAT           OUTPUT DATA BYTE
```

Output stop condition subroutine:

```
                          96 *
000209 997F               14  97 STOP   ANL     P1,#.NOT.SDA    OUTPUT STOP CONDITION
00020B 8A80               15  98        ORL     P2,#SCL
00020D 8980               16  99        ORL     P1,#SDA
00020F 83                 17 100        RET
```

Master outputs start condition, slave address, R/W bit and inputs acknowledge bit subroutine:

```
entry: A contains slave address and R/W bit
exit:  A = 0 if acknowledge is received
       contents of register R2 are modified.
```

```
                         107 *
000210 997F               18 108 START  ANL     P1,#.NOT.SDA    OUTPUT START CONDITION
000212 9A7F               19 109        ANL     P2,#.NOT.SCL
```

Master outputs data byte and inputs acknowledge bit subroutine:

```
entry: A contains data byte to be transmitted
exit:  A = 0 if acknowledge is received
       contents of register R2 are modified.
```

```
000214 BA08               20 116 MTDAT  MOV     R2,#8           SET BIT COUNTER
                             117 *
000216 E7                 21 118 MTDAT1 RL      A               SHIFT DATA BYTE
000217 121D               22 119        JB0     MTDAT2          JUMP IF TO BE TRANSMITTED BIT IS HIGH
000219 997F               23 120        ANL     P1,#.NOT.SDA    RESET SDA OUTPUT
00021B 4421               24 121        JMP     MTDAT3          CONTINUE
                             122 *
00021D 8980               25 123 MTDAT2 ORL     P1,#SDA         SET SDA OUTPUT
00021F 00                 26 124        NOP                     DELAY  ### CAN BE DELETED ###
000220 00                 27 125        NOP                     DELAY  ### CAN BE DELETED ###
                             126 *
000221 8A80               28 127 MTDAT3 ORL     P2,#SCL         SET SCL OUTPUT
000223 9A7F               29 128        ANL     P2,#.NOT.SCL    RESET SCL OUTPUT
000225 EA16               30 129        DJNZ    R2,MTDAT1       DECR. AND TEST BIT COUNTER
000227 8980               31 130        ORL     P1,#SDA         SET SDA OUTPUT TO INPUT MODE
000229 8A80               32 131        ORL     P2,#SCL         SET SCL OUTPUT
00022B 09                 33 132        IN      A,P1            INPUT ACKNOWLEDGE BIT
00022C 9A7F               34 133        ANL     P2,#.NOT.SCL    RESET SCL OUTPUT
00022E 5380               35 134        ANL     A,#SDA          MASK ACKN. BIT
000230 83                 36 135        RET
                             136 *
000231                       137        END

  NO ERRORS DETECTED
```

4.1.2 Master receiver routine (MAB8048 - SAB3028)

This software is intended for members of the MAB8048 microcomputer family where there is only one master in an I^2C-bus system. In this system none of the slaves are allowed to stretch the clock low time. This software example only meets the I^2C-bus protocol functionally, electrically there are some differences. (See section 4.1.1).

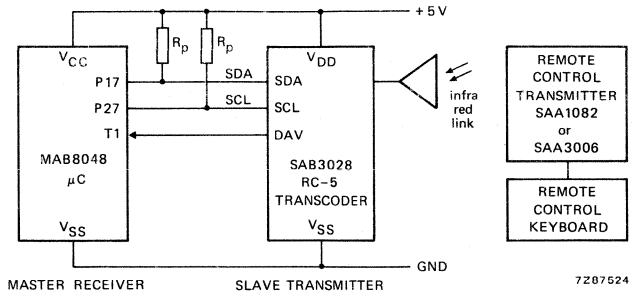


Fig. 4.3 Block diagram of MAB8048 - SAB3028 configuration.

Initialization:

```
000000                    52          ORG     H'000'
                          53 *
000000 2400          1    54 RESET    JMP     INIT          JUMP TO INITIALIZE ROUTINE
000002                    58          ORG     H'100'
                          59 *
000100 00            2    60 INIT     NOP                   NO INITIALISATION
                          61 *
```

Main program:

The main program polls the input T1, which is connected to the data valid output (DAV) of the RC-5 transcoder SAB3028. This output goes low if the RC-5 transcoder has received a valid remote control message. If low the main program calls subroutine RRCT, which reads the data out of the RC-5 transcoder via the I^2C-bus.

The subroutine RRCT returns with Accu = 0 if the data transfer is successful.

The received data is stored in the registers MRDTR1-4. If the transfer is successful the signal DAV is set to 1 again. If the data transfer is not successful ($A \neq 0$), the transmission is repeated.

The received data is stored in the registers MRDTR.

```
MRDTR1   EQU  H'04'     MASTER RECEIVER DATA BYTE 1 REGISTER
MRDTR2   EQU  MRDTR1+1            "               2     "
MRDTR3   EQU  MRDTR2+1            "               3     "
MRDTR4   EQU  MRDTR3+1            "               4     "
```

The format of the data transfer from the RC-5 transcoder SAB3028 is shown in Fig. 4.4:

S	RCT ADR	R	A	DATA 1	A	DATA 2	A	DATA 3	A	DATA 4	NA	P

Fig. 4.4 RC-5 transcoder data transfer.

S	is the START condition
RCT ADR	is the slave address of RC-5 transcoder
R	is the read/write bit in read state (=1)
A	is the acknowledge bit; this has to be 0
DATA	is data byte
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

The slave address of RC-5 transcoder is 0100 110.

```
RCTAD    EQU  H'4C'
```

The read bit position in the slave address is:

```
RW       EQU  H'01'
```

I^2C-bus signals of MAB8048 are:

```
SDA      EQU  H'80'      Pin P17  Pin nr. 34
SCL      EQU  H'80'      Pin P27  Pin nr. 38
```

```
000101 5601          3     64 MAIN   JT1     MAIN          WAIT FOR T1 = DAV = 0
                           65 *
000103 5400          4     66 MAIN1  CALL    RRCT          READ DATA OUT OF RC-5 TRANSCODER
000105 9603          5     67        JNZ     MAIN1         REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000107 2401          6     68        JMP     MAIN          CONTINUE
```

Master reads data out of RC-5 transcoder (SAB3028) subroutine:

exit: A = 0 if transmission is successful
contents of registers R0 and R2 are modified

```
                           75 *
000200 234D          7     76 RRCT   MOV     A,#RCTAD+RW   LOAD RCT SLAVE ADDRESS AND READ BIT
000202 541E          8     77        CALL    START         OUTPUT START CONDITION, SLAVE ADDRESS AND
                           78 *                            READ BIT; INPUT ACKNOWLEDGE BIT
000204 9617          9     79        JNZ     STOP          JUMP IF NO ACKNOWLEDGE RECEIVED
000206 B804         10     80        MOV     R0,#MRDTR1    SET DATA REGISTER INDEX
000208 5445         11     81        CALL    MRDAT         RECEIVE AND STORE FIRST DATA BYTE
00020A 543F         12     82        CALL    MRACK         OUTPUT ACKNOWLEDGE
                           83 *                            RECEIVE AND STORE SECOND DATA BYTE
00020C 543F         13     84        CALL    MRACK         OUTPUT ACKNOWLEDGE
                           85 *                            RECEIVE AND STORE THIRD DATA BYTE
00020E 543F         14     86        CALL    MRACK         OUTPUT ACKNOWLEDGE
                           87 *                            RECEIVE AND STORE FOURTH DATA BYTE
000210 27           15     88        CLR     A             TRANSMISSION SUCCESSFUL
```

Output not acknowledge bit and stop condition subroutine:

```
000211 8980          16      92 MRACKN ORL     P1,#SDA          SET SDA OUTPUT
000213 8A80          17      93        ORL     P2,#SCL          SET SCL OUTPUT
000215 9A7F          18      94        ANL     P2,#.NOT.SCL     RESET SCL OUTPUT
```

Output stop condition subroutine:

```
000217 997F          19      98 STOP   ANL     P1,#.NOT.SDA     OUTPUT STOP CONDITION
000219 8A80          20      99        ORL     P2,#SCL
00021B 8980          21     100        ORL     P1,#SDA
00021D 83            22     101        RET
```

Master outputs start condition, slave address, R/W bit and inputs acknowledge bit subroutine:

```
entry: A contains slave address and R/W̄ bit
exit:  A = 0 if acknowledge is received
       contents of register R2 are modified
```

```
                     108 *
00021E 997F          23     109 START  ANL     P1,#.NOT.SDA     OUTPUT START CONDITION
000220 9A7F          24     110        ANL     P2,#.NOT.SCL
```

Master outputs data byte and inputs acknowledge bit subroutine:

```
entry: A contains data byte to be transmitted
exit:  A = 0 if acknowledge is received
       contents of register R2 are modified
```

```
                     116 *
000222 BA08          25     117 MTDAT  MOV     R2,#8            SET BIT COUNTER
                     118 *
000224 E7            26     119 MTDAT1 RL      A                SHIFT DATA BYTE
000225 122B          27     120        JB0     MTDAT2           JUMP IF TO BE TRANSMITTED BIT IS HIGH
000227 997F          28     121        ANL     P1,#.NOT.SDA     RESET SDA OUTPUT
000229 442F          29     122        JMP     MTDAT3           CONTINUE
                     123 *
00022B 8980          30     124 MTDAT2 ORL     P1,#SDA          SET SDA OUTPUT
00022D 00            31     125        NOP                      DELAY  ### CAN BE DELETED ###
00022E 00            32     126        NOP                      DELAY  ### CAN BE DELETED ###
                     127 *
00022F 8A80          33     128 MTDAT3 ORL     P2,#SCL          SET SCL OUTPUT
000231 9A7F          34     129        ANL     P2,#.NOT.SCL     RESET SCL OUTPUT
000233 EA24          35     130        DJNZ    R2,MTDAT1        DECR. AND TEST BIT COUNTER
000235 8980          36     131        ORL     P1,#SDA          SET SDA OUTPUT TO INPUT MODE
000237 8A80          37     132        ORL     P2,#SCL          SET SCL OUTPUT
000239 09            38     133        IN      A,P1             INPUT ACKNOWLEDGE BIT
00023A 9A7F          39     134        ANL     P2,#.NOT.SCL     RESET SCL OUTPUT
00023C 5380          40     135        ANL     A,#SDA           MASK ACKN. BIT
00023E 83            41     136        RET
```

Master outputs acknowledge bit, reads data byte and stores data byte in RAM subroutine:

```
entry: R0 contains index of data register in which received data byte has
       to be stored
exit:  A contains received data byte
       contents of register R0 and R2 are modified
```

```
00023F 997F          42     146 MRACK  ANL     P1,#.NOT.SDA     RESET SDA OUTPUT
000241 8A80          43     147        ORL     P2,#SCL          SET SCL OUTPUT
000243 9A7F          44     148        ANL     P2,#.NOT.SCL     RESET SCL OUTPUT
```

Master reads data byte and stores data byte in RAM subroutine:

```
entry: R0 contains index of data register in which received data byte has
       to be stored
exit:  A contains received data byte
       contents of registers R0 and R2 are modified
```

```
000245 8980          45    157 MRDAT   ORL     P1,#SDA          SET SDA OUTPUT TO INPUT
000247 BA01          46    158         MOV     R2,#H'01'        INITIALIZE DATA WORD
                           159 *
000249 8A80          47    160 MRDAT1  ORL     P2,#SCL          SET SCL OUTPUT
00024B 09            48    161         IN      A,P1             INPUT BIT
00024C 9A7F          49    162         ANL     P2,#.NOT.SCL     RESET SCL OUTPUT
00024E F7            50    163         RLC     A                SHIFT DATA BIT INTO C
00024F FA            51    164         MOV     A,R2             FETCH DATA REGISTER
000250 F7            52    165         RLC     A                SHIFT DATA BIT INTO DATA WORD
000251 AA            53    166         MOV     R2,A             SAVE DATA WORD
000252 E649          54    167         JNC     MRDAT1           JUMP IF NOT YET 8 BITS RECEIVED
000254 A0            55    168         MOV     @R0,A            SAVE DATA WORD IN MRDTR REGISTER

000255 18            56    169         INC     R0               INCR. DATA REGISTER INDEX
000256 83            57    170         RET
                           171 *
000257                     172         END
```

4.1.3 Master transmitter/receiver routine with repeated start (MAB8048 - PCD8578)

This software is intended for members of the MAB8048 microcomputer family where there is only one master in an I^2C-bus system. In this system none of the slaves are allowed to stretch the clock low time. This software example only meets the I^2C-bus protocol functionally, electrically there are some differences. (See 4.1.1).

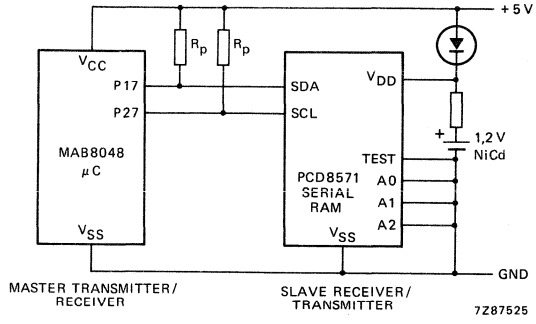


Fig. 4.5 Block diagram MAB8048 - PCD8571 configuration.

Initialization:

```
000100 23FF          2     58 INIT   MOV     A,#H'FF'
000102 AB            3     59        MOV     R3,A          SET POINTER VALUE
000103 AC            4     60        MOV     R4,A          SET DATA BYTE 1 TO BE WRITTEN
000104 AD            5     61        MOV     R5,A                  ''    2       ''
000000                     50        ORG     H'000'
                           51 *
000000 2400          1     52 RESET  JMP     INIT          JUMP TO INITIALIZE ROUTINE
                           53 *
                           54 * INITIALISATION:
                           55 *
000002                     56        ORG     H'100'
```

Main program:

The main program calls the following two I^2C-bus transmission subroutines:

- subroutine WMEM which writes the pointer value and two data bytes to the PCD8571 CMOS memory.
  The format of this transmission is shown in Fig. 4.6:

S	MEM ADR	W	A	POINT	A	DATA 1	A	DATA 2	A	P

Fig. 4.6 Format to write pointer value and two data bytes to the PCD8571.

- subroutine RMEM which writes the pointer value to the PCD8571 memory and reads two data bytes out of it.
  The format of this transmission is shown in Fig. 4.7:

S	MEM ADR	W	A	POINT	A	S	MEM ADR	R	A	DATA 1	A	DATA 2	NA	P

Fig. 4.7 Format to write pointer value and read two data bytes.

S	is the START condition
MEM ADR	is the slave address of the CMOS memory PCD8571
POINT	is the pointer address (internal RAM location address)
W	is the read/write bit in write state (= 0)
R	is the read/write bit in read state (= 1)
A	is the acknowledge bit; this has to be 0
NA	is the not acknowledge bit; this has to be 1
P	is the STOP condition

If one of the transmissions does not succeed (A = 0), this transmission is repeated.

The slave address of the CMOS memory is 1010 XXX.

```
MEMAD    EQU  H'AO'
```

The read bit position in the slave address is:

```
RW       EQU  H'01'
```

I^2C-bus signals of MAB8048 are:

```
SDA      EQU  H'80'      Pin P17  Pin nr. 34
SCL      EQU  H'80'      Pin P27  Pin nr. 38
```

```
000105 1B              6     65 MAIN   INC     R3              INCR. POINTER VALUE
000106 2301            7     66        MOV     A,#1            INCR. DATA TO BE WRITTEN
000108 6D              8     67        ADD     A,R5             R4,R5 + 1 --> R4,R5
000109 AD              9     68        MOV     R5,A
00010A 27             10     69        CLR     A
00010B 7C             11     70        ADDC    A,R4
00010C AC             12     71        MOV     R4,A
                             72 *
00010D 5400           13     73 MAIN1  CALL    WMEM            WRITE DATA TO CMOS MEMORY
00010F 960D           14     74        JNZ     MAIN1           REPEAT IF TRANMISSION IS NOT SUCCESSFUL
000111 BA00           15     75        MOV     R2,#0           SET DELAY COUNTER
000113 EA13           16     76        DJNZ    R2,$            DELAY
                             77 *
000115 5415           17     78 MAIN2  CALL    RMEM            READ DATA OUT OF CMOS MEMORY
000117 9615           18     79        JNZ     MAIN2           REPEAT IF TRANSMISSION IS NOT SUCCESSFUL
000119 BA00           19     80        MOV     R2,#0           SET DELAY COUNTER
00011B EA1B           20     81        DJNZ    R2,$            DELAY
00011D 2405           21     82        JMP     MAIN            CONTINUE
                             83 *
00011F                       84        ORG     H'200'
```

Master writes two data bytes to CMOS memory (PCD8571) subroutine:

To write data to the memory, the memory needs to know at which address (location) this data has to be stored. The CMOS memory PCD8571 is designed in such a way that the first data word after the slave address is always the internal address or pointer.

If more data bytes are written into the memory, within one transmission, this pointer is automatically incremented.

entry: R3 contains pointer value to be transmitted
R4 contains first data byte which has to be written in the memory.
R5 contains second data byte which has to be written in the memory.
exit: A = 0 if transmission is successful
A ≠ 0 if transmission is not successful
contents of register R2 are modified

```
                        100 *
000200 23A0          22 101 WMEM   MOV     A,#MEMAD        SET MEMORY ADDRESS AND WRITE BIT
000202 543C          23 102        CALL    START           OUTPUT START CONDITION, SLAVE ADDRESS AND
                        103 *                                 WRITE BIT; INPUT ACKNOWLEDGE BIT
000204 9635          24 104        JNZ     STOP            JUMP IF NO ACKNOWLEDGE RECEIVED
000206 FB            25 105        MOV     A,R3            FETCH POINTER VALUE
000207 5440          26 106        CALL    MTDAT           OUTPUT POINTER VALUE
000209 9635          27 107        JNZ     STOP            JUMP IF NO ACKNOWLEDGE
00020B FC            28 108        MOV     A,R4            FETCH FIRST DATA BYTE
00020C 5440          29 109        CALL    MTDAT           OUTPUT DATA BYTE
00020E 9635          30 110        JNZ     STOP            JUMP IF NO ACKNOWLEDGE
000210 FD            31 111        MOV     A,R5            FETCH SECOND DATA BYTE
000211 5440          32 112        CALL    MTDAT           OUTPUT DATA BYTE
000213 4435          33 113        JMP     STOP            OUTPUT STOP CONDITION
                        114 *
```

Master reads two data bytes out of CMOS memory (PCD8571) subroutine:

When the master wants to read data out of the memory, it first has to write the pointer into the memory. Because changing of the data direction is only possible after the R/W bit of the slave address, the master has to repeat the START condition, the slave address and the R/W bit (in read state) to change the direction within one transmission.

entry: R3 contains pointer value to be written in the memory.
exit: A = 0 if transmission is successful and received data is stored in registers R6 and R7.
A ≠ 0 if transmission is not successful
contents of registers R2, R6 and R7 are modified

```
000215 23A0          34 128 RMEM   MOV     A,#MEMAD        LOAD MEMORY SLAVE ADDRESS AND WRITE BIT
000217 543C          35 129        CALL    START           OUTPUT START CONDITION, SLAVE ADDRESS AND
                        130 *                                 WRITE BIT; INPUT ACKNOWLEDGE BIT
000219 9635          36 131        JNZ     STOP            JUMP IF NO ACKNOWLEDGE RECEIVED
00021B FB            37 132        MOV     A,R3            FETCH POINTER VALUE
00021C 5440          38 133        CALL    MTDAT           OUTPUT POINTER
00021E 9635          39 134        JNZ     STOP            JUMP IF NO ACKNOWLEDGE RECEIVED
000220 8A80          40 135        ORL     P2,#SCL         SET SCL OUTPUT; SCL AND SDA ARE HIGH NOW
000222 23A1          41 136        MOV     A,#MEMAD+RW     LOAD MEMORY SLAVE ADDRESS AND READ BIT
000224 543C          42 137        CALL    START           OUTPUT START CONDITION, SLAVE ADDRESS AND
                        138 *                                 READ BIT; INPUT ACKNOWLEDGE BIT
000226 9635          43 139        JNZ     STOP            JUMP IF NO ACKNOWLEDGE RECEIVED
000228 5463          44 140        CALL    MRDAT           READ FIRST DATA BYTE
00022A AE            45 141        MOV     R6,A            SAVE FIRST DATA BYTE
00022B 545D          46 142        CALL    MRACK           OUTPUT ACKNOWLEDGE
                        143 *                                 READ SECOND DATA BYTE
00022D AF            47 144        MOV     R7,A            SAVE SECOND DATA BYTE
00022E 27            48 145        CLR     A               TRANSMISSION SUCCESSFUL
```

Output not acknowledge bit and stop condition subroutine:

```
00022F 8980          49 149 MRACKN ORL     P1,#SDA         SET SDA OUTPUT
000231 8A80          50 150        ORL     P2,#SCL         SET SCL OUTPUT
000233 9A7F          51 151        ANL     P2,#.NOT.SCL    RESET SCL OUTPUT
                        152 *
```

Output stop condition subroutine:

```
000235 997F          52 155 STOP   ANL     P1,#.NOT.SDA    OUTPUT STOP CONDITION
000237 8A80          53 156        ORL     P2,#SCL
000239 8980          54 157        ORL     P1,#SDA
00023B 83            55 158        RET
```

Master outputs start condition, slave address, R/W bit and inputs acknowledge bit subroutine:

entry: A contains slave address and R/$\overline{W}$ bit
exit: A = 0 if acknowledge is received
contents of register R2 are modified

```
00023C 997F          56 166 START  ANL     P1,#.NOT.SDA    OUTPUT START CONDITION
00023E 9A7F          57 167        ANL     P2,#.NOT.SCL
                        168 *
```

Master outputs data byte and inputs acknowledge bit subroutine

```
entry: A contains data byte to be transmitted
exit:  A = 0 if acknowledge is received
       contents of register 2 are modified
```

```
                        173 *
000240 BA08          58 174 MTDAT   MOV     R2,#8             SET BIT COUNTER
                        175 *
000242 E7            59 176 MTDAT1  RL      A                 SHIFT DATA BYTE
000243 1249          60 177         JB0     MTDAT2            JUMP IF TO BE TRANSMITTED BIT IS HIGH
000245 997F          61 178         ANL     P1,#.NOT.SDA      RESET SDA OUTPUT
000247 444D          62 179         JMP     MTDAT3            CONTINUE
                        180 *
000249 8980          63 181 MTDAT2  ORL     P1,#SDA           SET SDA OUTPUT
00024B 00            64 182         NOP                       DELAY  ### CAN BE DELETED ###
00024C 00            65 183         NOP                       DELAY  ### CAN BE DELETED ###
                        184 *
00024D 8A80          66 185 MTDAT3  ORL     P2,#SCL           SET SCL OUTPUT
00024F 9A7F          67 186         ANL     P2,#.NOT.SCL      RESET SCL OUTPUT
000251 EA42          68 187         DJNZ    R2,MTDAT1         DECR. AND TEST BIT COUNTER
000253 8980          69 188         ORL     P1,#SDA           SET SDA OUTPUT TO INPUT MODE
000255 8A80          70 189         ORL     P2,#SCL           SET SCL OUTPUT
000257 09            71 190         IN      A,P1              INPUT ACKNOWLEDGE BIT
000258 9A7F          72 191         ANL     P2,#.NOT.SCL      RESET SCL OUTPUT
00025A 5380          73 192         ANL     A,#SDA            MASK ACKN. BIT
00025C 83            74 193         RET
```

Master outputs acknowledge bit and reads data byte subroutinte:

```
exit:  A contains recieved data byte
       contents of register 2 are modified
```

```
00025F 8A80          76 200         ORL     P2,#SCL           SET SCL OUTPUT
000261 9A7F          77 201         ANL     P2,#.NOT.SCL      RESET SCL OUTPUT
                        202 *
```

Master reads data byte subroutine:

```
exit:  A contains received data byte
       contents of register 2 are modified
```

```
                        206 *
000263 8980          78 207 MRDAT   ORL     P1,#SDA           SET SDA OUTPUT TO INPUT
000265 BA01          79 208         MOV     R2,#H'01'         INITIALIZE DATA WORD
                        209 *
000267 8A80          80 210 MRDAT1  ORL     P2,#SCL           SET SCL OUTPUT
000269 09            81 211         IN      A,P1              INPUT BIT
00026A 9A7F          82 212         ANL     P2,#.NOT.SCL      RESET SCL OUTPUT
00026C F7            83 213         RLC     A                 SHIFT DATA BIT INTO C
00026D FA            84 214         MOV     A,R2              FETCH DATA REGISTER
00026E F7            85 215         RLC     A                 SHIFT DATA BIT INTO DATA WORD
00026F AA            86 216         MOV     R2,A              SAVE DATA WORD
000270 E667          87 217         JNC     MRDAT1            JUMP IF NOT YET 8 BITS RECEIVED
000272 83            88 218         RET
                        219 *
000273                  220         END
```

11. The D^2B specification

CONTENTS – THE D²B SPECIFICATION

		page
1.0	INTRODUCTION	11-5
2.0	THE D²B CONCEPT	11-7
3.0	GENERAL CHARACTERISTICS	11-10
4.0	ARBITRATION	11-10
5.0	MESSAGE PROTOCOL	11-11
5.1	Start bit field	11-11
5.2	Mode field	11-11
5.3	Master field	11-12
5.4	Slave field	11-12
5.5	Control field	11-12
5.6	Data field	11-12
5.6.1	Write action on the bus	11-13
5.6.2	Read action on the bus	11-13
6.0	BIT PROTOCOL	11-14
6.1	Bit formats	11-16
6.1.1	Start bit	11-17
6.1.2	Arbitration bit	11-23
6.1.3	Master to slave bit	11-31
6.1.4	Slave to master bit	11-36
7.0	USE OF BITS	11-41
7.1	Control bits	11-41
7.2	End-of-data bit	11-42
7.3	Parity bit	11-42
7.4	Acknowledge bit	11-42
8.0	ELECTRICAL SPECIFICATION	11-43
8.1	Logical and electrical state relationship	11-44
8.2	Driver requirements	11-44
8.3	Receiver requirements	11-45
8.4	Cable characteristics	11-46
8.5	Configuration	11-46
9.0	TIMING	11-46
9.1	Bit timing	11-47
9.1.1	Start bit timing	11-47
9.1.2	Mode bit timing	11-47
9.1.3	Master address bit timing	11-48
9.1.4	Master to slave bit timing	11-48
9.1.5	Slave to master bit timing	11-50
9.2	Message times and transmission speeds	11-51

		page
10.0	DATA FIELD INTERPRETATION	11-53
10.1	Slave status	11-53
10.2	Lock address	11-54
10.3	Memory address	11-54
10.4	Data	11-55
10.5	Commands	11-55

Note: In the following chapter, most of the supplementary information on commands and address space occupation has disappeared. These items, which were in the main orientated towards audio/video applications, are independent of D^2B bus controller hardware. They will be published as part of official standards (CENELEC, IEC).

1.0 INTRODUCTION

The use of microprocessors and microcontrollers within various types of stand-alone domestic, business and industrial equipment has led to a demand for this 'isolated intelligence' to be linked.

- In the home: most modern TVs, VCRs and Hi-Fis include microcomputer-control, yet they cannot control each other. For example, when turning on a VCR to play-back a recording, the TV should also be automatically turned on and retuned to the VCR setting.

- In the office: word processors, small business computers and telex machines are usually stand-alone systems, although each may possess information that the other needs. And, if stand-alone systems become overloaded, expansion is difficult, if not impossible.

- In industrial, or automotive, applications: many different control and monitoring devices are generally implemented on one system (the motor car is a prime example). All these controls need to communicate, since one change in system status usually causes others. However, it is very important that this communication should not affect the performance of the controlling device.

Various solutions have been suggested to combat the problem of "isolated intelligence". In large businesses requiring units within an office block, or even a town, to communicate, the Local Area Network (LAN) is gaining favour.

In Local Area Networks, any unit can take control of the serial data channel and transmit data to any other unit. LANs can expand modularly, adapt to changing traffic patterns, and usually continue to operate when one of the units fails. Typical LANs contain hundreds of work stations and transmit data at rates somewhere between 1 and 10 Mbit/s, over distances ranging from 400 m to over 1 km.

However, this performance comes at a high price, in the form of complex protocols, controllers and system software; and for larger systems, the need for professional network managers. Obviously, the LAN is over-specified for the examples we have already mentioned. These applications only require enough performance to handle the throughput of a modest number of microcomputer-controlled units.

To match network capacity to actual application requirements, Philips has introduced two serial buses, the D^2B (Digital Data Bus) and I^2C (Inter-IC) bus (Fig. 1.1).

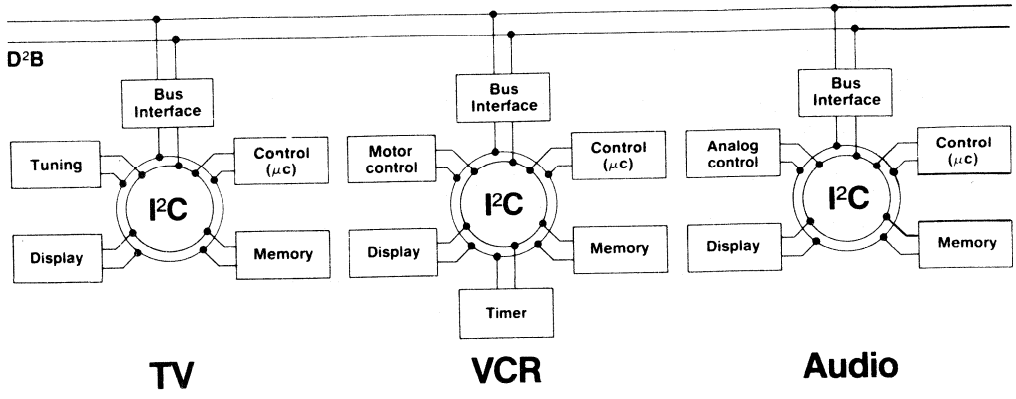


Fig. 1.1 The D^2B concept applied to an integrated Video/Audio system, each part of which employs the I^2C-bus.

The I^2C-bus connects various devices within a unit, or on a circuit board. It provides an easy means of upgrading equipment by enabling devices to be added or removed as necessary. Wiring is also reduced from the 8 wires for typical parallel data transmissions to a 2-wire serial link. This feature also saves space and simplifies circuit board layout.

The Digital Data Bus (D^2B) connects equipment rather than integrated circuits. In order to minimize interference caused by energy radiation, the D^2B is implemented as a differentially-driven cable pair. Changes of state are therefore represented as the difference between two (non-TTL level) voltages.

The D^2B complements both the LAN and I^2C-bus concepts. It can perform both as an integrated circuit communication path and as a small LAN-type network. Therefore, the D^2B is the link, between I^2C-bus and LAN performance, that allows a hierarchy of network data communications:

1) I^2C-bus - communication within equipment

(100 kbit/s max., distance = a few metres max.).

2) D^2B - communication between equipment in close proximity

(100kbits/s max., resulting in an effective character rate of 7.8 kbytes/s., distance = 150 m max.).

3) LAN - communication between widely distributed equipment

(10 Mbits/s max., distance = 1 km max.).

2.0 THE D^2B CONCEPT

The following requirements were put forward for the D^2B:

- The bus should allow any unit capable of taking control of the bus to do so. It should also permit control of one unit by another.

- The bus should allow a self-configuring network to be built up.

The bus was also required to handle the following operational constraints:

- The removal of a unit should not affect the functioning of the remaining units (nor should power switching on/off in a unit).

- The bus should fit into the audio/video Euroconnector proposal (maximum of 2 signal wires available).

- Interference from the bus should not degrade the proper functioning of equipment in its vicinity.

The D^2B is designed to perform in a number of different environments (industrial, domestic, automotive, data processing) and can transfer data at rates appropriate to each application area. There are several transfer modes. At present, three modes have been defined, but more can be introduced if they are needed.

Mode 0: Nominal transfer rate up to 209 char/s. The controller in this mode can operate with an internal clock tolerance of $\pm$ 25%, permitting the use of an RC network. Mode 0 is designed to also allow a microprocessor (with a crystal-controlled clock) which is not fully occupied with other tasks, to act as a bus controller.

Mode 1: Nominal transfer rate up to 2457 char/s. This mode allows an inexpensive hardware design for the bus controller. The controller can work with a high internal clock tolerance of $\pm$ 25%, permitting the use of an RC network rather than a crystal.

Mode 2: Nominal transfer rate up to 7760 char/s. The controller for this mode allows the use of a clock with a tolerance of $\pm$ 0,5%, permitting the use of a ceramic resonator.

These modes can be mixed within the same system. For example, a printer may be able to operate in Mode 1 and a disc drive in Mode 2, with a processing unit switching from one mode to the other to communicate with each of these peripherals. Thus any "intelligent" unit connected to the bus can communicate with any other unit.

On the D^2B, units perform as masters and slaves during data transfers. A master unit is any device, such as a microcomputer-controlled VCR, which is capable of initiating and controlling a data transfer.

Depending on the direction of transfer, units also perform as transmitters or receivers. To use the previous example, both the disc drive and the printer are slave devices. The disc drive, however, can both receive and transmit data, whilst the printer can usually only receive.

The D^2B is a multi-master bus, which means that any device capable of controlling the bus can do so, without disturbing the operation of the bus.

When a transfer is initiated, the unit(s) wishing to use the bus lays a claim by generating a unique condition - the start bit. If more than one unit tries to take control of the bus during this bit, their claims are settled in subsequent bits by an arbitration protocol.

The protocol stems from the wired-AND connection of each unit to the D^2B. Consequently, there is no central master, or exchange, required.

To try and establish the right to use the bus, each competing unit transmits the binary code associated with the mode in which it wants to work - lower modes have higher priority. If the mode is the same, arbitration continues to the next stage. In this stage, each unit transmits its own address.

Since each unit connected to the D^2B has a unique address, there will be a difference between the bits transmitted by competing masters, and the first to generate a 1 when the other generates a 0 will lose the arbitration (for more information on arbitration, see section 4.0).

The unit which wins the arbitration is now the master and controls the bus. So that one master cannot monopolise the bus, it is only allowed to control the bus for one interval, or time slot.

There are 6 bit-fields (Fig. 2.1). Three of these fields; the start, mode and master address fields have already been described; they deal with the initialization and arbitration involved in a transfer. The remaining fields are the slave address, control and data fields.

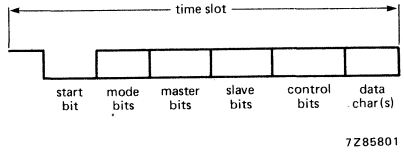


Fig. 2.1 Message format of the D^2B, showing the 6 bit-fields.

When a master has gained control of the bus, it sends the address of the unit it wants as a slave. Control bits are then sent to indicate the direction and nature of the transfer. Finally, the data bytes are either sent or received. A receiver-generated acknowledge is also necessary to indicate the correct reception of the slave address, control bits and each data byte during the transmission.

In Modes 1 and 2, the maximum number of bytes transferred depends on whether the transfer is master to slave, or slave to master. In a master to slave transfer, timing is determined by the master only - this means that the master generates synchronization and data for each bit transferred. In a slave to master transfer, the master still generates the synchronization but the slave puts the data value on to the bus. Thus, slave to master timing is determined by the combination of both units and, as a consequence, is longer (more information on bit formats will be given in Section 5.0).

The effect of finite propagation times and rise and fall times have been taken into account in calculating the number of clock pulses required for each bit field. In addition, the influence of the necessary driving and decoding logic imposes constraints on the timing. These constraints lead to the data transmission capabilities of Modes 1 and 2 depending on the direction of transfer.

The number of data bytes that can be sent in one time slot are:

- in Mode 0: 2 data bytes in either direction
- in Mode 1: 32 data bytes from master to slave
  16 data bytes from slave to master
- in Mode 2: 128 data bytes from master to slave
  64 data bytes from slave to master

A unit may use the bus for less than one time slot, but never for more. If a master still needs to transmit or receive information after its time slot expires, it must again try for control of the bus.

However, there are cases when a master cannot afford to lose control of a slave. For example, a master may have set a register within a slave before sending data; if the value of this register is changed in a subsequent time slot, the master cannot resume the transfer when it again controls the bus.

To avoid this situation, the master can request the slave to be locked to it. This is achieved by sending the appropriate control bits in the control field. Other masters are then restricted from accessing that slave until it is unlocked.

3.0 GENERAL CHARACTERISTICS

The D^2B requires only one data channel. In this specification it is defined in terms of a differentially-driven cable pair - but the D^2B can use many types of data channel as long as it can be in a wired-AND configuration.

The maximum length of the D^2B as a differentially-driven cable pair, which may be twisted to give higher noise immunity, is 150 m and up to 50 units can be connected to it. The maximum number of units, due to the capacity of the addressing allocation, that can be connected to the bus is 4096. The D^2B can be used in a bus configuration and/or in a star network.

4.0 ARBITRATION

Every unit wishing to control the D^2B has to go through an arbitration procedure. This procedure involves comparing the mode and master address bits of arbitrating-masters (Fig. 4.1), using the wired-AND connection of every unit to the D^2B.

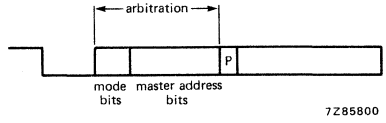


Fig. 4.1 Arbitration takes place during the mode and master fields.

- After each mode bit is placed on the bus, and no other master in a lower mode is found (i.e. the mode bit was a 1), the masters raise their transmission speed and arbitrate again. Therefore, at the end of the mode bits, the masters that remain are competing in the same mode.

- For each arbitration bit, each arbitrating-master has to check that the actual level on the bus corresponds to the level it placed on the bus. If it doesn't, the arbitrating-master has lost the arbitration and has to follow the rest of the message as a possible slave. At the end of the master address bits, there is only one master.

Due to the wired-AND property of connections to the D^2B, a 0 put on the bus will override a 1, this results in the following priority during arbitration:

- the lowest mode has the highest priority
- in the case of an equal mode, the lowest master address has the highest priority.

5.0 MESSAGE PROTOCOL

This section explains how individual bits fit into fields, and what these fields mean in the generation of a message on the D^2B (Fig. 5.1).

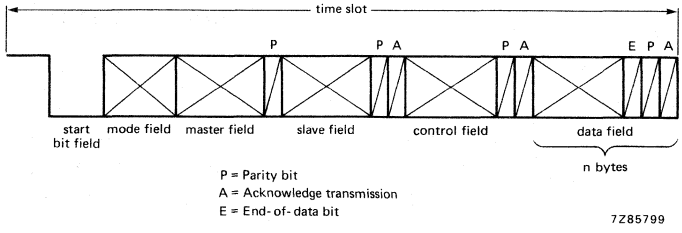


Fig. 5.1 A complete message on the D^2B.

5.1 Start bit field

A unit that wants to master the bus tests whether it is in use and attempts to capture it. If the unit does not want to become a master it monitors the bus.

To initiate a message transfer, the unit puts a LOW on the bus for a given, unique, length of time. The other units will recognize this as the start bit. At the end of this period masters competing for the bus enter the mode field. Slaves continue monitoring the bus to see if they are being addressed. If the bit is too short for a start bit then everyone returns to the monitoring state.

5.2 Mode field

On the D^2B, three different speed modes can be selected by the mode bits. At the end of the start bit competing masters (which may be in different speed modes) enter the mode field by putting a series of bits on the bus.

Mode 0 = 0
Mode 1 = 10
Mode 2 = 110

Slaves finding a higher mode on the bus than they are capable of following return to the monitoring state.

5.3 Master field

If two or more masters are transmitting in the same mode arbitration continues into the master field. Due to the wired-AND property, the master with the lowest address will win the arbitration. Masters put their address on the bus with the most significant bit (MSB) first. After each bit they read and compare. If the bus condition differs from their address bit, they relinquish their attempt for the line and continue as potential slaves. At the end of the address field of 12 bits only 1 master is still transmitting. This master then closes the address field with a parity bit enabling other devices to check the validity of the address.

5.4 Slave field

Having won the arbitration, the master transmits the address of the unit it requires as a slave. The most significant bit is sent first. As mentioned in section 5.3, each unit's address consists of 12 bits. After the address of the slave unit, a parity bit is sent to test whether the address is accurately received.

To find out whether or not the slave unit is listening on the bus, the master asks for an acknowledge. All possible slave units are obliged to read the bus at least until the slave address bits do not agree with their own address. When this happens, the unit returns to the monitoring state.

If the slave address matches the address of one of the listening units, and the parity of both master and slave addresses is found to be odd (and therefore correct), the slave unit generates an acknowledge. If the parity is even, the addressed slave does not give an acknowledge - indicating the incorrect reception of the address. It then returns to the monitoring state.

5.5 Control field

In this field the master puts a 4-bit word on the bus (MSB first). This word describes the nature of the message (e.g., direction of transfer, whether the slave should be locked to the master, or whether the data is actual data, status data, a lock address, a memory address or a command). The control bits are more fully described in Section 7.1. The control field is closed with a parity bit followed, as in the slave address field, with an acknowledge bit. If the slave acknowledges, the master proceeds to the next field, if not the master may try to repeat the transfer.

The slave reads the control bits and checks the parity. If it cannot perform the function or the parity is even, it does not acknowledge the master and returns to the monitoring state. If the parity is odd and it can execute the function required by the master it gives an acknowledge and proceeds to the next field.

5.6 Data field

In this field, the master writes data to or reads data from the slave (MSB first). Each byte consists of 8 bits followed by an end-of-data (EOD) bit, a parity bit and an acknowledge bit.

The end-of-data bit informs the receiver whether or not the present byte is the last one of the message. This bit is necessary because in each mode a message does not have to fill its allocated time slot.

Thus, if a message contains less than the maximum number of bytes, a logic zero in the end-of-data bit of the last byte indicates that the message is over, provided the following acknowledge bit is received or transmitted correctly. After the end-of-data bit, the parity bit and acknowledge bit are sent.

5.6.1 Write action on the bus

In the case of a write action on the bus, the master transmits the data bits, the end-of-data bit, the parity bit, and requests an acknowledge.

The slave collects the data, the end-of-data bit and the parity bit. The parity is then checked and if correct, the acknowledge is given.

If the parity-sum is even, the byte and EOD bit are rejected and the slave does not acknowledge.

If the byte is rejected and there is still time in the time slot for a retry, the master will send the byte again and continue to do so until the byte is accepted or the time slot has expired.

If the parity-sum is odd, the slave accepts the data and generates an acknowledge. The master then starts transmitting the next data byte, if the message wasn't finished or the time slot hadn't expired.

The end-of-data bit is generated by the master-transmitter. This value is a logic 1 when the master still has another byte to transmit and the present byte-number is not the maximum number that the message can contain. The end-of-data bit is a logic 0 when the master has no more data to send, or when the present byte number is the maximum number that the message can contain.

5.6.2 Read action on the bus

For a read action, the master sends all the synchronization edges and the acknowledge bit. During the data periods of the data bits, the end-of-data bit and the parity bit, the slave puts the required levels on the bus.

The master collects the data, the end-of-data bit and the parity bit; then the parity-sum is checked.

If it is even, the byte and EOD bit are rejected and the master-receiver doesn't acknowledge. If the byte is rejected and there is still time in the time slot then the master will read the byte again and continue to do so until either the byte is accepted or the time slot expires.

If the parity-sum is odd, the master accepts the data and generates an acknowledge.

If this byte is not accompanied by an EOD 0 and the time slot has not expired, the master reads another byte.

6.0 BIT PROTOCOL

In this section the state table given in italics details the exact state of the bus during each section of each bit. This information is not required in the first reading and may be omitted.

The bits placed upon the D^2B comprise four sections in the time domain (Fig. 6.1):

- an initial period at logic level 1, the PREPARATION PERIOD.
- a following period at logic level 0, the SYNCHRONIZATION PERIOD.
- a period containing the actual bit value, the DATA PERIOD.
- a final period at logic level 1, the STOP PERIOD.

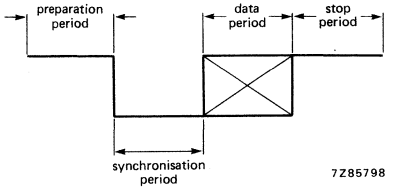


Fig. 6.1 The general bit format on the D^2B.

The duration of the bits and their subdivision depend on the mode in use, the type of bit being transmitted, the specified maximum propagation times through the bus and the maximum fall and rise times of the signal at a receiving port (see section 9.0 for more detailed timing requirements).

In the case of bits taking part in arbitration, the second and third parts of the bits are divided into two time intervals. This is to permit an arbitrating master to check the conditions on the bus before proceeding.

In the following definitions of the various types of data bits, the time intervals relating to the master are specified as T1, T2 etc. Time intervals relating to the slave are specified as Ta, Tb etc. (see Section 9.1)

Note: Where the state or counter content has an " ' " indication, this state is part of the next bit.

The calculation of the timing parameters has taken into account the delay caused by the data-acquisition circuit. In the reference circuit shown (Fig. 6.2), the first flip-flop (FF_0) synchronizes the asynchronous bus signal with the filter clock. Thereafter the other two flip-flops (FF_1 and FF_2) ensure that the signal Q2 remains stable for at least two clock cycles. Flip-flop FF_3 synchronizes the asynchronous filter output signal (Q2) with the logic clock.

The clocks of the filter and the logic may be asynchronous but their frequencies must fulfill the stated requirements for the clocks in each mode.

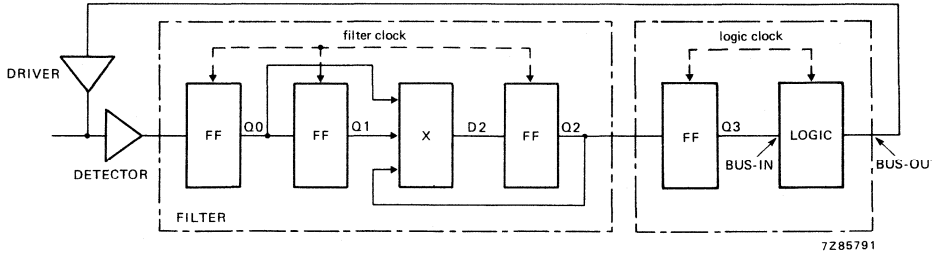


Fig. 6.2 Reference data-acquisition circuit.

Logic function X

Q0	Q1	Q2	D2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

6.1 Bit formats

The following construction is used to describe the sequence of states in the various bit types:

STATE (X) : a certain time section X of a bit

CONDITION : the conditions valid on the clock edge

ACTION : actions to be executed on the clock edge on which the conditions are true

The following constraints are imposed upon the logic handling the bit timing:

- the result of the actions and the ensuing conditions must be stable within one clock period.
- the delay between the clock edge and the bus-out signal is included in the driver delay.

The following signal procedure and data definitions are used:

Signal definition:
- bus-in : input Q3 for the logic
- bus-out : output signal of the logic
- master-request: request for transmission pending
- relevant-bit : represents the value (0 or 1) of the bit that is transmitted
- bit value : represents the value (0 or 1) of the bit that is received
- arbitration-not-lost
- arbitration-lost
- last-bit-in-the-message
- not-last-bit-in-the-message

Procedure definition:
- timing error : timing error handling and after the timing error handling go to STATE (T_a) of start bit

Data definition:
- clcnt : counter which counts clock pulses

6.1.1 Start bit

The unit wanting to be bus master tests the bus for a HIGH level. If a LOW is detected, it indicates that the bus is already in use. If the bus stays HIGH during this time, the master sends the start bit. The start bit is a signal to all other units that a transmission is beginning.

Fig. 6.3 illustrates the start bit format.

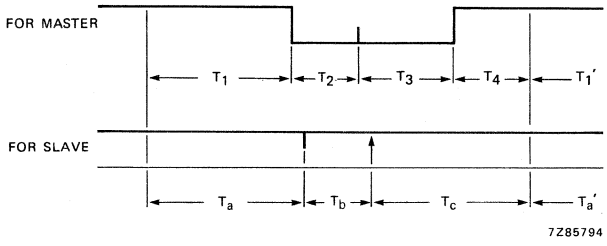


Fig. 6.3 The start bit format.

Master:

- T_1:

  If a unit wishes to become master and the bus is HIGH during T_1, it enters T_2. The HIGH indicates that no other unit has either generated a start condition, or is busy on the bus.

  If a unit wishes to become bus master during T_1 and a LOW is detected, the unit becomes arbitrating-master and enters T_b (slave timing) keeping its output HIGH.

Note: The reason an arbitrating-master can perform slave timing is so that it does not disturb a message in progress. However, the other units on the bus - whether arbitrating-masters or slaves - must be able to recognize the start bit. The only way for them to do this, is by following T_b and T_c to detect the long LOW period on the bus which indicates the generation of the start bit.

- T_2:

  When a master enters T_2, it puts a LOW on the bus and samples the bus for this LOW. When this LOW is found on the bus, the master enters T_3. If no LOW is found, a timing error has occurred.

- T_3:

  The bus should be kept LOW during T_3, the master then enters T_4 and generates a HIGH on the bus. If a HIGH is detected during T_3, a timing error has occurred - since T_3 is the interval which indicates the unique long LOW period of the start bit.

- T_4:

  When the HIGH is detected during T_4, the master enters T_1'. If the bus is still LOW at the end of T_4, a timing error has occurred.

Slave:

- T_a:

  This state is called the monitoring state. If a LOW is detected during T_a, the unit enters T_b as a possible slave.

- T_b:

  The bus should be LOW during T_b; the unit then enters T_c. If a HIGH is detected, the bit is not a start bit and the unit re-enters T_a.

- T_c:

  Although a unit can be performing the slave timing during the start bit, it is still possible for it to become a master. If during T_c, a HIGH is detected and the unit now wishes to become the master, it enters T_1' and follows the master timing.

  If a unit does not want to be a master, it enters T_a' as a possible slave and follows slave timing

  If the bus is still LOW at the end of T_c, a timing error has occurred.

For specific bit timings see section 9.1

The state sequence for the start bit is as follows:

Initial:

STATE (T_a)

CONDITION : master-request = 1 AND bus-in = 1

ACTION : bus-out = 1
clcnt = t_1
state = STATE (T_1)

CONDITION : bus-in = 0

ACTION : bus-out = 1
clcnt = t_b
state = STATE (T_b)

CONDITION : master-request = 0 AND bus-in = 1

ACTION : bus-out = 1
state = STATE (T_a)

For the master:

STATE (T_1)

CONDITION : bus-in = 0

ACTION : bus-out = 1
clcnt = t_b
state = STATE (T_b)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_1)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 0
clcnt = t_2
state = STATE (T_2)

STATE (T_2)

CONDITION : bus-in = 0

ACTION : bus-out = 0
clcnt = t_3
state = STATE (T_3)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_2)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_3)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_3)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_4
state = STATE (T_4)

CONDITION : bus-in = 1

ACTION : bus-out = 1
timing error

STATE (T_4)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_4)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1

ACTION : bus-out = 1
clcnt = t_1'
state = STATE (T_1')

For the slave:

STATE (T_b)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_b)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_c
state = STATE (T_c)

CONDITION : master-request = 0 AND bus-in = 1

ACTION : bus-out = 1
state = STATE (T_a)

CONDITION : master-request = 1 AND bus-in = 1

ACTION : bus-out = 1
clcnt = t_1
state = STATE (T_1)

STATE (T_c)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_c)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : master-request = 0 AND bus-in = 1

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

CONDITION : master-request = 1 AND bus-in = 1

ACTION : bus-out = 1
clcnt = t_1'
state = STATE (T_1')

6.1.2 Arbitration bit

For arbitration, which takes place during the mode and master address bits, the bit format is shown in Fig. 6.4.

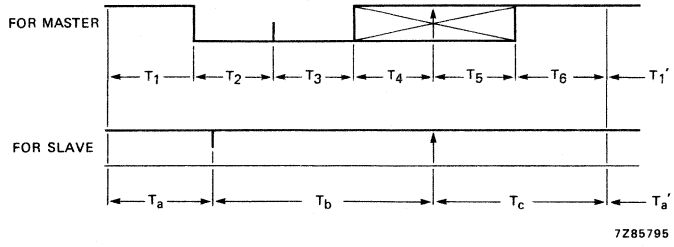


Fig. 6.4 Arbitration bit format.

- T_1:

  If the bus remains HIGH during T_1, the unit enters T_2. If a LOW is detected, a master enters T_3 keeping its output HIGH. This indicates that another master has generated the synchronization pulse for this bit.

- T_2:

  A master has to generate the synchronization pulse by pulling the bus LOW. As soon as this LOW is found on the bus, the master enters T_3 keeping its output LOW. If the LOW is not found, a timing error has occurred.

- T_3:

  At the end of T_3, the master enters T_4.

- T_4:

  At the beginning of T_4, the arbitrating-master transmits the relevant bit for arbitrating with other units. At the end of T_4, the unit reads the bus and enters T_5.

  When the bus level is the same as its bit level, the master continues arbitrating. If the levels are different (i.e. there is a LOW on the bus when a HIGH was generated), it has lost the arbitration.

- T_5:

  If the bus is LOW at the end of T_5, the unit enters T_6. If the bus is HIGH, the next state is entered (T_1'). A master which has lost the arbitration enters T_a' instead of T_1'.

- T_6:

  The master generates a HIGH and when this is detected during T_6, the arbitrating-master enters the next state (T_1'). A unit which has lost the arbitration enters T_a', as a possible slave.

  If the bus remains LOW, a timing error has occurred.

Slave:

- T_a:

  During T_a, a LOW should be detected on the bus. When it is, the unit enters T_b. If no LOW is detected, a timing error has occurred.

- T_b:

  At the end of T_b, the slave samples the bus. When the sample is HIGH, the next state is entered (T_a'). If not, the slave enters T_c.

- T_c:

  When a HIGH is detected during T_c, the next state is entered (T_a'). If the bus remains LOW, a timing error has occurred.

The state sequence for the arbitration bit is :

For the master:

STATE (T_1)

CONDITION : bus-in = 0

ACTION : bus-out = 1
clcnt = t_3
state = STATE (T_3)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_1)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 0
clcnt = t_2
state = STATE (T_2)

STATE (T_2)

CONDITION : bus-in = 0

ACTION : bus-out = 0
clcnt = t_3
state = STATE (T_3)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_2)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_3)

CONDITION : bus-out = 0 AND clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_3)

CONDITION : bus-out = 1 and clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_3)

CONDITION : clcnt = 1

ACTION : bus-out = relevant bit
clcnt = t_4
state = STATE (T_4)

STATE (T_4)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T_4)

CONDITION : bus-in = 0 AND clcnt = 1 AND relevant bit = 0

ACTION : bus-out = relevant bit
clcnt = t_5
bit value = 0
arbitration-not-lost
state = STATE (T_5)

CONDITION : bus-in = 1 AND clcnt = 1 AND relevant bit = 0

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 0 AND clcnt = 1 AND relevant bit = 1

ACTION : bus-out = relevant bit
clcnt = t_5
bit value = 0
arbitration-lost
state = STATE (T_5)

CONDITION : bus-in = 1 AND clcnt = 1 AND relevant bit = 1

ACTION : bus-out = relevant bit
clcnt = t_5
bit value = 1
arbitration-not-lost
state = STATE (T_5)

STATE (T_5)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T_5)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_6
state = STATE (T_6)

CONDITION : bus-in = 1 AND clcnt = 1 AND arbitration-not-lost

ACTION : bus-out = 1
clcnt = t_1'
state = STATE (T_1')

CONDITION : bus-in = 1 AND clcnt = 1 AND arbitration-lost

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

STATE (T_6)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_6)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1 AND arbitration-not-lost

ACTION : bus-out = 1
clcnt = t_1'
state = STATE (T_1')

CONDITION : bus-in = 1 AND arbitration-lost

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

For the slave:

STATE (T_a)

CONDITION : bus-in = 0

ACTION : bus-out = 1
clcnt = t_b
state = STATE (T_b)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_a)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_b)

CONDITION : clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_b)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_c
bit value = 0
state = STATE (T_c)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_a'
bit value = 1
state = STATE (T_a')

STATE (T_c)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_c)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

6.1.3 Master to slave bit

Master to slave bits form the address of the unit required as a slave; the control bits which indicate the type of transfer; and the actual data bits (Fig. 6.5). Note that if the control bits stipulate a slave to master transfer, the data bits will follow the format in section 6.1.4.

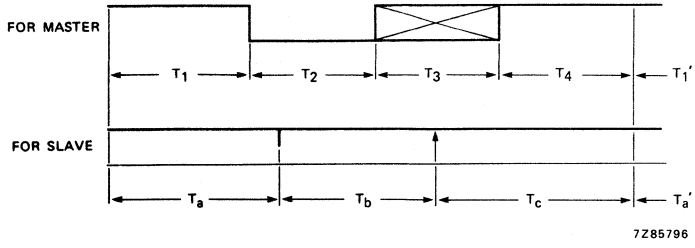


Fig. 6.5 Master to slave bit format.

Master:

- T_1:

  The bus should remain HIGH during T_1. If it does, the master enters T_2. If a LOW is detected, a timing error has occurred.

- T_2:

  The master generates the synchronization pulse by pulling the bus LOW. If the bus remains HIGH during T_2, a timing error has occurred. If not, the master enters T_3.

- T_3:

  During T_3, the master transmits the relevant data on the bus. It then enters T_4.

- T_4:

  During T_4, the master transmits a HIGH on the bus. If a LOW is detected after this period, a timing error has occurred. If not, the next state is entered (T_2').

Slave:

- T_a:

  A LOW should be detected during T_a. When it is, the slave enters T_b. If the bus remains HIGH, a timing error has occurred.

- T_b:

  At the end of T_b, the unit samples the bus to read the bit. If the bus is HIGH, the next state is entered (T_a'). If it is LOW, the unit enters T_c.

- T_c:

  As soon as a HIGH is detected, the next state is entered (T_a'). If the bus is still LOW at the end of T_c, a timing error has occurred.

The state sequence for master to slave bits is :

For the master:

STATE (T_1)

CONDITION : bus-in = 0

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_1)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 0
clcnt = t_2
state = STATE (T_2)

STATE (T_2)

CONDITION : clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_2)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = relevant bit
clcnt = t_3
state = STATE (T_3)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_3)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T_3)

CONDITION : clcnt = 1

ACTION : bus-out = 1
clcnt = t_4
state = STATE (T_4)

STATE (T_4)

CONDITION : clcnt >1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_4)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1 AND clcnt = 1 AND last-bit-in-the-message

ACTION : bus-out = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1 AND clcnt = 1 AND not-last-bit-in-the-message

ACTION : bus-out = 0
clcnt = t_2'
state = STATE (T_2')

For the slave:

STATE (T_a)

CONDITION : bus-in = 0

ACTION : bus-out = 1
clcnt = t_b
state = STATE (T_b)

CONDITION : bus-in = 1 AND clcnt >1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_a)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_b)

CONDITION : clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_b)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_c
bit value = 0
state = STATE (T_c)

CONDITION : bus-in = 1 AND clcnt = 1 AND last-bit-in-message

ACTION : bus-out = 1
bit value = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1 AND clcnt = 1 AND not-the-last-bit-in-the-message

ACTION : bus-out = 1
clcnt = t_a'
bit value = 1
state = STATE (T_a')

STATE (T_c)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_c)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1 AND last-bit-in-the-message

ACTION : bus-out = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1 AND not-last-bit-in-the-message.

ACTION : bus-out = 1
clcnt = t_a'
state = (T_a')

6.1.4 Slave to master bit

The slave to master bits form the "slave transmitter to master receiver" data transfer and acknowledge.

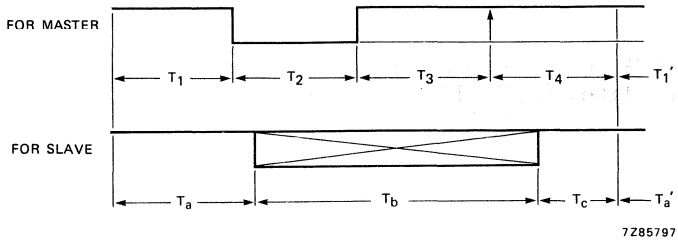


Fig. 6.6 Slave to master bit format.

Master:

- T_1:

  Does not apply (T_1 is always zero).

- T_2:

  The master transmits the synchronization pulse by pulling the bus LOW during T_2. At the end of this period, when the LOW is detected on the bus, the master enters T_3. If the bus remains high during T_2, a timing error has occurred.

- T_3:

  During T_3, the master generates a HIGH on the bus. After T_3, it samples the bus to read the bit transmitted by the slave and enters T_4.

- T_4:

  The bus should be HIGH at the end of T_4. The master then enters the next state (T_2'). If the bus is still LOW, a timing error has occurred.

Slave:

- T_a:

  When the LOW generated by the master is detected, the slave enters T_b. If the bus is still HIGH at the end of T_a, a timing error has occurred.

- T_b:

  During T_b, the slave transmits the relevant bit. If at the end of T_b the bus is HIGH, the slave enters (T_a'). If the bus is LOW, the slave enters T_c.

- T_c:

  During T_c, a HIGH should be detected on the bus. When it is, the next state is entered (T_a'). If the bus remains LOW, a timing error has occurred.

The state sequence for slave to master bits is:

For the master:

STATE (T_1)

(Does not apply)

STATE (T_2)

CONDITION : clcnt >1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T_2)

CONDITION : bus-in = 0 AND clcnt =1

ACTION : bus-out = 1
clcnt = t_3
state = STATE (T_3)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_3)

CONDITION : clcnt >1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_3)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_4
bit value = 0
state = STATE (T_4)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_4
bit value = 1
state = STATE (T_4)

STATE (T_4)

CONDITION : clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_4)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 0
timing error

CONDITION : bus-in = 1 AND clcnt = 1 AND last-bit-in-the-message

ACTION : bus-out = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1, clcnt = 1 AND not-last-bit-in-the-message

ACTION : bus-out = 0
clcnt = t_2'
state = STATE (T_2')

For the slave:

STATE (T_a)

CONDITION : bus-in = 0

ACTION : bus-out = relevant bit
clcnt = t_b
state = STATE (T_b)

CONDITION : bus-in = 1 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_a)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T_b)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T_b)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t_c
state = STATE (T_c)

CONDITION : bus-in = 1 AND clcnt = 1 AND last-bit-in-the-message

ACTION : bus-out = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1 AND clcnt = 1 AND not-last-bit-in-the-message

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

STATE (T_c)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T_c)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
timing error

CONDITION : bus-in = 1 AND last-bit-in-the-message

ACTION : bus-out = 1
state = STATE (T_a of the start bit)

CONDITION : bus-in = 1 AND not-last-bit-in-the-message

ACTION : bus-out = 1
clcnt = t_a'
state = STATE (T_a')

7.0 USE OF BITS

In this section a detailed description is given of the meaning of:

- Control bits
- End-of-data bit
- Parity bit
- Acknowledge bit

7.1 Control bits

The 4 control bits define the kind of exchange that has to take place. The following table gives the meaning of the different combinations of the control bits.

B3	B2	B1	B0	FUNCTION
0	0	0	0	Read slave status
0	0	0	1	Reserved for future use
0	0	1	0	Read slave status and lock
0	0	1	1	Read data and lock
0	1	0	0	Read medium and least significant lock address nibble*
0	1	0	1	Read most significant lock address nibble**
0	1	1	0	Read slave status and unlock
0	1	1	1	Read data and unlock
1	0	0	0	Write memory address and lock
1	0	0	1	Reserved for future use
1	0	1	0	Write commands and lock
1	0	1	1	Write data and lock
1	1	0	0	Reserved for future use
1	1	0	1	Reserved for future use
1	1	1	0	Write commands and unlock
1	1	1	1	Write data and unlock

Note: When a unit A is locked to a unit B, then:

- unit A may only execute the coded control functions:

B3	B2	B1	B0	
0	0	0	0	- Read slave status
0	1	0	0	- Read medium and least significant lock address nibble
0	1	0	1	- Read high significant lock address nibble

  asked for by units other than unit B.

  Note: the control codes 0000, 0100 and 0101 do not affect the lock status of the slave.

- unit A is allowed to execute all the control functions asked for by unit B.

* The least significant lock address nibble is transferred in the least significant part of the byte.

** The most significant lock address nible is transferred in the least significant part of the byte.

7.2 End-of-data bit

To indicate that a byte is the last data byte within the current message, a bit is added to each data byte.

End-of-data

1 - Not the last data byte.
0 - The last data byte in the current message.

Note: The end-of-data bit is included in the parity check.

7.3 Parity bit

Parity bits are used to protect:

- master bits
- slave bits
- control bits
- data + end-of-data bit

The parity defined is odd parity. If the bits protected by a parity bit comprise an even number of ones, the parity bit will be a logical 1. If the bits comprise an odd number of ones, the parity bit will be a logical 0.

7.4 Acknowledge bit

During every transfer there are three different acknowledge bits, the first after the slave address, the second after the control bits and the third after each data byte.

Every data byte has to be acknowledged:

0 indicates a positive acknowledge
1 indicates a negative acknowledge

The first acknowledge bit should not be transmitted by the addressed slave under the following conditions:

1. Parity fault on the master and/or slave address
2. Mode too high for slave
3. Timing error
4. Slave not present.

In these cases the transmission will be stopped.

The second acknowledge bit should not be transmitted by the slave under the following conditions:

1. Parity fault on the control bits.
2. Receiver buffer is not empty and bit B3 is logic 1
3. No data in data buffer, the bit B3 is logic 0 and it is not a status or lock address request.
4. The slave is locked to another master, except for status without lock/unlock or a lock-address request.
5. The slave has no memory (see Section 10.1 and 10.3) and the master is going to provide a memory address.
6. The slave is not locked when the lock-address is requested.
7. Timing error
8. Reserved control codes.

In these cases transmission will be stopped.

The third acknowledge bit should not be transmitted under the following conditions:

1. Parity fault seen on the data.
2. A timing error in the receiver during last data transfer (last = since previous acknowledge).
3. When the receiver buffer is full and it cannot accept another byte.

In these cases, the transmitter has to re-try the byte except when this byte was the last in the time slot.

8.0 ELECTRICAL SPECIFICATION

The D^2B cable consists of a differential floating pair forming the data bus (Fig. 8.1).

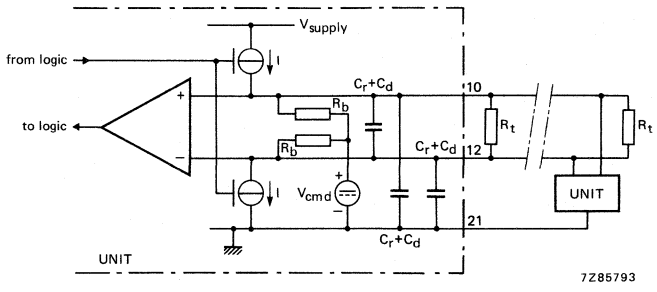


Fig. 8.1 Reference circuit for connecting units to the D^2B via a differentially driven cable.

8.1 Logical and electrical state relationship

The relationship between the logical states and the electrical levels on the data lines is given the table below:

Logical state	Electrical levels
0	$\geqslant$ + 120 mV (V_{10} - V_{12}) called ON
1	$\leqslant$ + 20 mV (V_{10} - V_{12}) called OFF

8.2 Driver requirements

The specification for the device drivers is as follows:

- Voltage V_{12} > Voltage V_{21} and $V_{10} < 5$ V.
- The common mode voltage ($Vcm_d = 0{,}5(V_{10} + V_{12})$ with respect to point 21) is 2,5 V $\pm$ 10% under all operating conditions.
- The driver current (I) is:

$$2{,}75 \text{ mA} \leqslant I \leqslant 5 \text{ mA at ON}$$
$$I < 1 \ \mu\text{A at OFF}$$

- The driver resistance ($R = 2R_b$) between point 10 and 12 must be greater than 100 kΩ for the OFF state.
- The driver load capacitance (C) is:

$C_d < 25$ pF between points 10 and 12
$C_d < 25$ pF between points 10 and 21
$C_d < 25$ pF between points 12 and 21

- The transition time values (T) are given in Fig. 8.2.

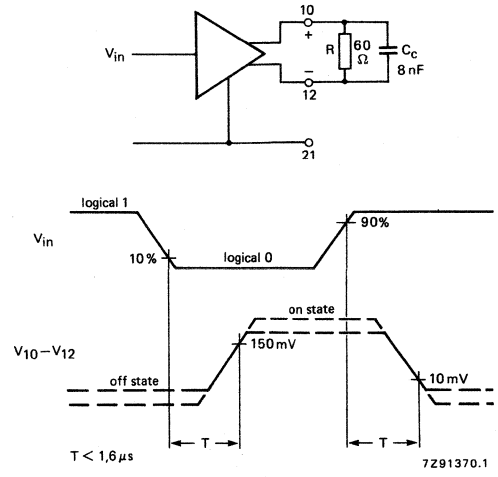


Fig. 8.2 The transition time measurement.

8.3 Receiver requirements

The receiver specification is as follows:

- The common mode voltage range ($Vcm_r = 0{,}5(V_{10} + V_{12})$ with respect to point 21) is:

  $1\ V < Vcm_r < 3{,}75\ V$ for ON
  $0\ V < Vcm_r < 5\ V$ for OFF

- The input voltage ($V_{10} - V_{12}$) is:

  $V_{10} - V_{12} \geqslant 120$ mV for ON
  $V_{10} - V_{12} \leqslant 20$ mV for OFF

- Hysteresis must be 20 mV minimum

- The input impedance is:

  $R > 2{,}5\ M\Omega$, $C_r < 25$ pF between points 10 and 21

  $R > 2{,}5\ M\Omega$, $C_r < 25$ pF between points 12 and 21

  $R > 1\ M\Omega$, $C_r < 25$ pF between points 10 and 12

- The receiver delay values are given in Fig. 8.3.

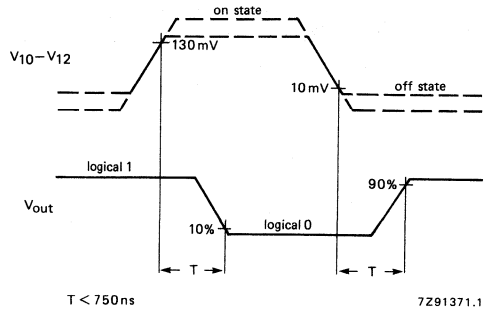


Fig. 8.3 Receiver delay measurement.

8.4 Cable characteristics

- Wire resistance $\leqslant 0{,}1\ \Omega/m$
- Characteristic impedance (Z_0) = $120\Omega \pm 20\%$

8.5 Configuration

- Bus length = 150 m (max)
- Resistive cable termination = $120\Omega \pm 5\%$ at each end
- Up to 50 units can be connected to a well matched bus. If any spurs used, they should be no longer than 2,5 m.

9.0 TIMING

The signal timing tolerances are based on a signal rise and fall time of less than 1,5 μs at any terminal connected to the bus.

The propagation delay due to the bus wires has to be less than 0,9 μs.

The delay of the driver circuitry must be less than 0,1 μs.

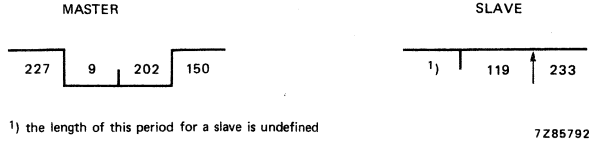
The delay of the detector circuitry must be less than 0,75 μs.

9.1 Bit timing

In this section, the timing of each division within the different bit types is indicated. The time divisions are expressed in terms of clock pulses at the frequency of the mode concerned. The meaning of each division has already been described in section 6.0.

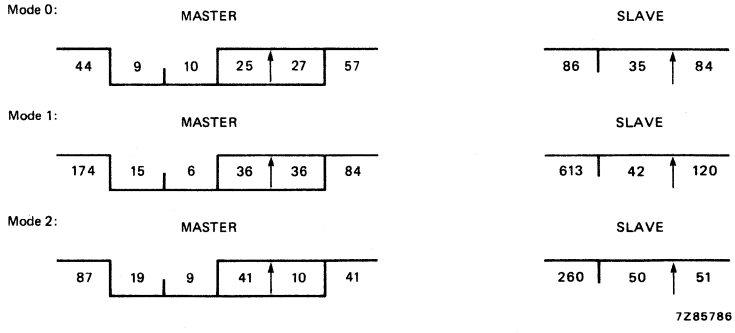
9.1.1 Start bit timing

The start bit always takes place in Mode 0.

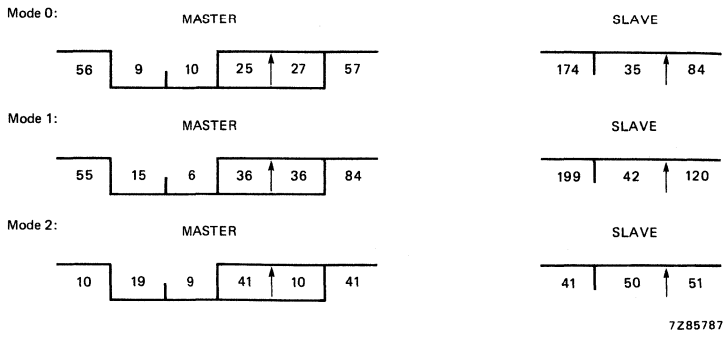


1) the length of this period for a slave is undefined

9.1.2 Mode bit timing

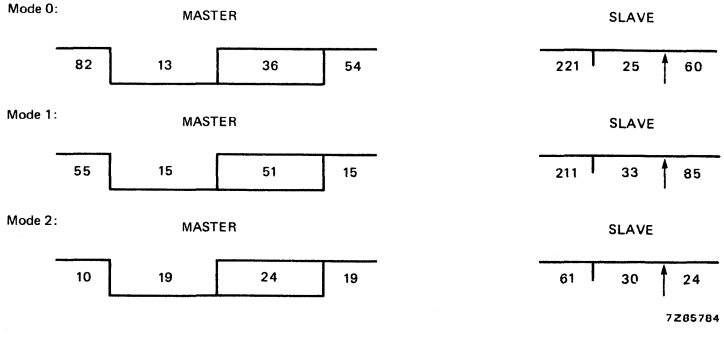


9.1.3 Master address bit timing

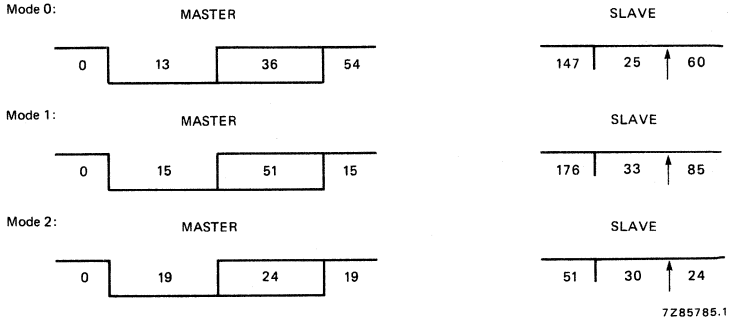


9.1.4 Master to slave bit timing

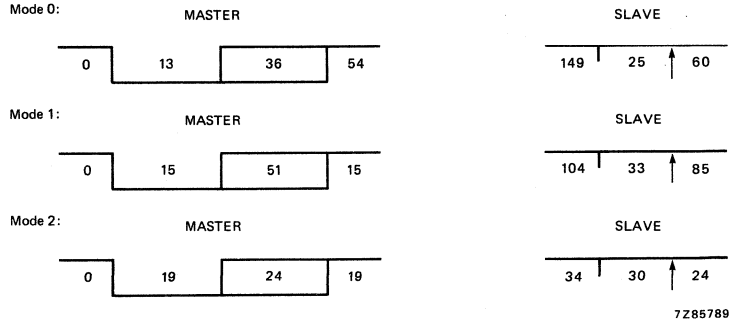
- Parity bit on master address



- First bit of the control field (B3),
- First bit of each data byte (master to slave),
- ACK bit, on data in the case of slave to master transfer.

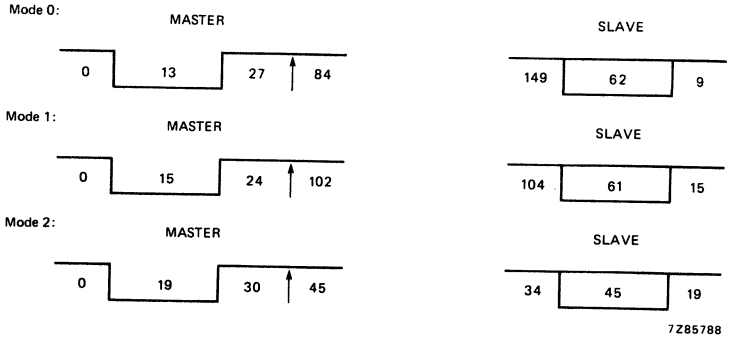


- All other master to slave bits.

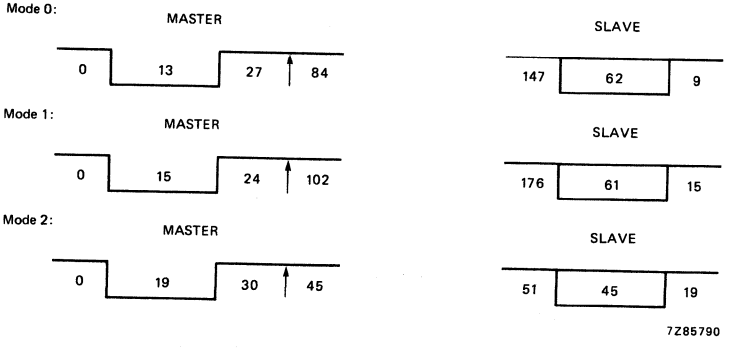


9.1.5 Slave to master bit timing

- ACK bit - slave to master,
- First bit of every data byte (slave to master).
  (except for the first data byte in the message).



- All other slave to master bits.



9.2 Message times and transmission speeds (all times given in μs)

	Mode 0 6/8 MHz ±25%		Mode 1 6/2 MHz ±25%		Mode 2 6 MHz ±0,5%	
	min	max	min	max	min	max
Startbit	464	795	464	795	577	599
Mode 0 bit	119	221	116	205	144	155
Mode 1 bit			68	126	84	90
Mode 2 bit					25	31
Master addr. + parity.	1746	3231	471	930	157	230
Slave address + parity + ACK.	1560	2601	318	531	149	151
Control + parity + ACK.	681	1136	145	243	66	68
Overhead	4570	7984	1582	2830	1202	1324
SINGLE BYTE TRANSFER						
Master to slave.						
Data + parity + ACK.	1230	2052	253	423	118	120
Overhead	4570	7984	1582	2830	1202	1324
Total message time.	5800	10036	1835	3253	1320	1444
Transmission speed (c.p.s)	99	173	307	545	692	758
Slave to master.						
Data + parity + ACK.	1432	2388	397	663	166	168
Overhead	4570	7984	1582	2830	1202	1324
Total message time.	6002	10372	1979	3493	1368	1492
Transmission speed (c.p.s)	96	167	286	506	670	731

	Mode 0 6/8 MHz ±25%		Mode 1 6/2 MHz ±25%		Mode 2 6 MHz ±0,5%	
	min	max	min	max	min	max
MULTIPLE BYTE TRANSFER						
Master to slave.						
No. of bytes						
Mode 0 = 2	2460	4104				
Mode 1 = 32			8096	13536		
Mode 2 = 128					15104	15360
Overhead	4570	7984	1582	2830	1202	1324
Total message time.	7030	12088	9678	16366	16306	16684
Transmission speed (c.p.s.)	165	285	1955	3307	7672	7850
Slave to master.						
No. of bytes						
Mode 0 = 2	2864	4776				
Mode 1 = 16			6352	10608		
Mode 2 = 64					10624	10752
Overhead	4570	7984	1582	2830	1202	1324
Total message time.	7434	12760	7934	13438	11826	12076
Transmission speed (c.p.s.)	156	270	1190	2017	5299	5412

10.0 DATA FIELD INTERPRETATION

10.1 Slave status

By requesting the slave status, it is possible for the master to determine why the slave doesn't give an acknowledge, or will not give an acknowledge.

The slave status is only valid for last slave transfer.

Every unit should possess the possibility to provide status information. The status bits have the following meaning:

Bit	Value	Meaning
0 *	0	data buffer empty
	1	data buffer not empty
1 **	0	receiver buffer empty
	1	receiver buffer not empty
2	0	unit not locked
	1	unit locked
3	0	unit has no memory
	1	unit has memory
4 ***	0	slave transmitter section disabled
	1	slave transmitter section enabled
5	0	always set to zero.
7,6	00	Mode 0
	01	Mode 1 Indicates the highest mode
	10	Mode 2 in which the unit can work
	11	reserved for future use

(Bit 7 is the MSB bit)

* The data buffer is the buffer which can be accessed by a read data action defined by the control bits.

** The receiver buffer is the buffer which can be accessed by a write action defined by the control bits.

*** If the slave transmitter section is disabled, no data is available from the data buffer. However, status data and the lock address may be read.

10.2 Lock address

When the control bits specify "read medium and least significant nibble lock addresses" the interpretation of the 8 data bits (bit 7 is the most significant bit) is:

Bit	Meaning
7	bit 7 of address-locking master

- similarly for bit numbers 6 to 0

The address-locking master contains 12 bits numbered from 0 to 11. Bit number 11 is the most significant bit.

When the control bits specify "read most significant lock address nibble" the interpretation of the 8 data bits (bit 7 is the most significant bit) is:

Bit	Meaning
7-4	undefined
3	bit 11 of address-locking master
2	bit 10 "
1	bit 9 "
0	bit 8 "

10.3 Memory address

When a unit receives and acknowledges a control code "write memory address and lock", this unit must reset its slave status bit 0 to zero; indicating the data buffer is empty. This avoids reading irrelevant data and allows time for the unit to perform the following actions before the memory data can be collected in a subsequent time slot:

- interpret the "write memory address and lock" control code.
- assemble the memory address from the received data bytes
- fetch the memory data, put it in the 'next to be sent position' and set the slave staus bit 0 to one; indicating the data buffer is not empty.

The coding of the memory address and the relevance of the memory data is beyond the scope of this book.

10.4 Data

When the control bits specify "read data ...", the data contained in the slave's buffer is received by the master.

The interpretation of the received data depends upon the specification of the unit.

When the control bits specify "write data ...", the interpretation of the received data depends upon the specification of this unit.

10.5 Commands

The control codes "write commands ...", allow the accompanying data to be treated differently from normal data. Typically these commands are standardized for a set of addresses (an application area) and may be used to delimit data or to activate functions in the addressed unit, etc.

12. Microcontroller Test and Development systems

TEST AND DEVELOPMENT SYSTEMS

page

LIST OF CONTENTS

1.0 INTRODUCTION 12-5

2.0 MICROCOMPUTER DEVELOPMENT SYSTEMS (PMDS) 12-9

2.1 Configuration of a development system 12-9

2.1.1 Cross assembler 12-9
2.1.2 The microcomputer adaptor box (MAB) 12-10
2.1.3 PMDS 2/3 Emulation and debug 12-10
2.1.3.1 Universal debug unit – UDU 12-11
2.1.4 PMDS Peripherals 12-13

2.2 Microcontroller development system – PMDS 2/PM4422 12-14

2.2.1 Hardware specification – PMDS 2 12-14
2.2.2 Hardware options – PMDS 2 12-17
2.2.3 Software specification 12-17
2.2.4 Software options 12-19
2.2.5 Ordering information 12-19

2.3 Microcontroller development system – PMDS 3/PM49XX 12-20

2.3.1 Philips integration and debug station – PIDS 12-22
2.3.2 Hardware specification – PIDS 12-22
2.3.3 Hardware options 12-23
2.3.4 Software specification – PIDS 12-23
2.3.5 Software options 12-24
2.3.6 Ordering information 12-24

3.0 SDS – THE STAND-ALONE DEBUG STATION – PM47XX 12-25

3.1 System architecture 12-25
3.2 System software 12-26
3.3 SDS 8051/80C51 12-27
3.4 SDS 84X1/84CXX 12-28
3.5 Ordering information 12-28

4.0 ENGINEERING AND DEVELOPMENT SYSTEMS – PEDS – PM48XX 12-30

4.1 Hardware specification 12-30
4.2 Hardware options 12-32
4.3 Software specification 12-33
4.4 Software options 12-34
4.5 Ordering information 12-34

5.0 THE 8051/80C51 PL/M COMPILER 12-35

5.1 Ordering information 12-36

6.0 LOGIC ANALYZERS 12-37

6.1 Introduction 12-37
6.2 PM3665/PM3570 logic analyzers 12-37
6.3 PM3632 logic analyzer 12-40

1.0 INTRODUCTION

Implementing today's microprocessors in today's electronic systems is a much more complex business when compared to the 1970s. Nowadays, very sophisticated test and development equipment is needed to ensure that full advantage is taken of the power of the microprocessor. A range of equipment including development systems, logic analyzers and oscilloscopes are required in almost every application.

Why do we need a microcontroller development system? well, When developing microcontroller systems, the same problem will always arise: how can software and hardware be tested together when neither is at any stage of completion? Hardware cannot be tested realistically until software is present and software cannot be tried out on a system that is still half on the drawing board. To wait until both are approaching 'completion' before testing is doomed to failure and would certainly result in rehashed circuits and patched-up software. The best solution is to test, find faults and 'debug' as the system develops. To make this possible, facilities are needed to 'emulate' those parts of the target microcontroller system which are still incomplete. Emulation is the operation of the prototype CPU under development system control, the CPU is removed from the prototype and is replaced by an emulation probe. Emulation and debugging require the use of a powerful 'host' computer system. If the target system requires anything more than fairly elementary programming, computer support is practically essential for efficient program development.

Our involvement in DTS dates right back to the introduction of microprocessors, as part of a deliberate policy of support as well as supply, we complement our development systems with a range of logic analyzers for hardware development and software/hardware integration and debugging. These run from low-price benchtop portables providing basic test facilities, to powerful tools with many advanced features, such as transitional timing and software performance analysis. It's still a fact that for several stages of the development cycle there really is no subsititute for a powerful logic analyzer. These offer a complete package to develop software in high and low level languages and the necessary debugging; debugging of hardware; integration of hardware and software; system testing and production. For large projects they are invaluable for dividing the project into manageable modules and for keeping track of the various completed stages.

Complete support

Projects vary enormously in scope and size. The equipment needed at each stage will vary and the relative importance of each stage of the design cycle will change. We provide the following compatible tools to service your needs:

- Microcomputer Development System (PMDS 2) _ PM4422
- Microcomputer Development System (PMDS 3) - PM49XX
- Engineering Development Systems (PEDS) - PM48XX
- Stand alone Debug Station (SDS)
- Logic analyzers

Both PMDS systems are complete tools for the development of microcontroller-based products. PMDS 2 is a multi-user system which permits multi-debug operations. Up to four Universal Debug Units (UDUs) can be operated with both PMDS and PEDS configured for different types of target CPU. The Philips systems-the multi-user PMDS 2, the P-C based PMDS 3 and the single- 2- or 4-user PEDS (Philips engineering development system) permit source code generation in high and low-level languages, simultaneous assembly and linking of modules, advanced cross-compiling, many UNIX* utilities and full symbolic debugging. Also, the SDS stand alone debug station offers a low-cost alternative to the above. This compact system provides full debug facilities, and together with the industry standard RS232 connection, it can be operated from any host computer.

Most PMDS hardware and software is microcontroller-independent, both the system facilities and the commands to obtain them are identical for each microcontroller supported. The system can also be 'field enhanced' for new microcontrollers by the addition of a relatively small amount of hardware and software.

The complete range of support for our 8-bit microcontrollers can be seen at a glance below.

Philips microcomputer development systems

PM 49XX	NEW	Single-user microcomputer development system PMDS 3
PM 4422		Multi-user microcomputer development system PMDS 2
PM 48XX		Philips engineering and development system PEDS

Software for PMDS, PEDS, IBM PC and DEC VAX:

	NEW	PL/M-51 compiler for 8051 family

Specific microprocessor and microcontroller support for PEDS/PMDS

8048 family
8051 CMOS/NMOS family
84CXX/84X1 CMOS/NMOS family

Options for PMDS/PEDS

Many options exist, including; more memory, hard disc extensions, streamer tape back-up drives, PROM programmers, etc.

PM 4496		Streamer tape backup
PM 47XX	NEW	Stand-alone Debug stations (SDS)
		SDS for 80C51
		SDS for 84C00
		SDS for 5010/5011 DSP

* Unix ix a trademark of AT&T

The items listed here are current at the time of going to press, but are continually being updated. Where an item of interest does not appear, your local Philips Organization will provide up-to-date information on the latest additions, with current prices.

Table 1 shows the complete range of microprocessor/microcontroller support using PMDS 3, PMDS 2 and PEDS

uP/uC	Cross-assembler	PL/M	Emulator (MABs)
8021	●	-	-
8022	●	-	-
8035	●	-	●
8039	●	-	●
8040	●	-	●
8041	●	-	-
8048	●	-	●
8049	●	-	●
8050	●	-	●
8051	●	●	●
80C51	●	●	●
8400 NMOS	●	-	●
84CXX	●	-	●

Legend
● = available now
- = not yet available

A number of emulators also support the CMOS versions of the processors (i.e. extra instructions for IDLE and STOP modes, higher clock speed).

PM8443/40 PCF84C12 28 to 20 pin emulation adaptor plug

PMDS, PEDS and SDS compatible emulation probes for PCF84C43 and PCF84C85, are available through ELCOMA sales offices via your local Philips national organization.

A typical development sequence

A quick look at a development sequence will illustrate what support we provide. First the needs of the product are identified, the specifications written and the initial design is undertaken, the software and hardware development phases are then separated. The software functions are defined and built up from individual modules each performing given tasks. Source code is then generated for the modules; they are assembled and compiled and the complete program debugged. The hardware engineer designs the circuit, breadboards and performs basic debugging, he then commits the design to PCB and completes any further debugging. The hardware and software are then integrated and tested again for any remaining bugs. Finally, the product undergoes a full system test and performance analysis, and is then put into production.

For the software developer, there is no reasonable altenative to a full, universal development system. For the hardware developers, they too have an excellent option - a logic analyzer. This offers more sophisticated triggering and logic analysis than a development system, and fulfils the need for an instrument to fully test the wire-wrap prototype with logic timing analysis. The more sophisticated the analysis, the fewer bugs remain hidden, saving time at integration.

At initial integration, the emulation facilities of the development systems are invaluable. Emulation of processors is fully transparent, real-time, and up to four microprocessors can be emulated simultaneously. The facilities of the logic analyzers are also very useful at the integration phase.

Some faults are extremely elusive and the very singular trace capabilities of the logic analyzer are needed. An LA has 'objective monitoring' - it can monitor any point in the circuit, whereas the trace of the MDS is dedicated to the processor bus. Also, to find faults that occur only on a particular conditional sequence you need the complex triggering capabilities of an analyzer, not just the trigger or break-point of the MDS.

The MDS and LA are complementary for many applications. For example, you can run a trace on the processor with the MDS while monitoring other logic with the LA. Very complex qualified triggering is possible, offering extremely fast system checking.

Logic analyzers

Considering logic analyzers, two may be highlighted: the PM3570, Philips' most powerful and sophisticated analyzer with the features needed for all stages of development; and the PM3632, an affordable bench top portable for personal analysis power. This is a compact unit which still satisfies many analysis needs.

Performance analysis

Before production, performance analysis is a very important step. This requires a powerful logic analyzer. With performance analysis, the working of the system can be optimized before it is operational. Specific system sections can be individually tested to locate bottlenecks and indicate corrective action.

An objective analysis by a logic analyzer is often needed at pre- and post-production because they can be used to locate bugs very quickly. The PM3632 in particular can be used by small and large groups because it was deliberately designed as a device with extensive analyzer facilities.

System managers needs are met by software packages available for the PMDS/PEDS systems, such as the SCCS - Source Code Control System - this utility logs details of any software changes and maintains an audit trail for all software releases.

We also supply a range of software for other development users. State-of-the-art compiler packages are available to allow software development on a VAX, while utilising PEDS as a hardware development station. These packages offer the user the possibility to have a cross-compiler customised to a particular environment.

2.0 MICROCOMPUTER DEVELOPMENT SYSTEMS (PMDS)

The following pages have information on our products that cover all development and test requirements. In the MDS area, PEDS stands out as a powerful and cost effective development system for small development teams, or for a more sophisticated development linked to a VAX mainframe. For medium to large development teams the successful PMDS 2 stands out as a fully integrated and versatile multi-user development system.

Whereas PMDS and PEDS provide the total concept, low-cost evaluation tools are also available. The Stand-alone Debug Stations (SDS), for example, can connect up to any host and provide emulation facilities for evaluation of the target processor. The range of microcontrollers supported by the SDS is illustrated in the relevant SDS section. (Section 3.5).

In PMDS and PEDS the software development environment is a UNIX*-based operating system - widely regarded as an industry standard. UNIX was implemented on PMDS/PEDS specifically to bring high efficiency to microcomputer development, by speeding up the development process and thus cutting costs.

Hardware debugging and the all-important software/hardware integration phase of the development cycle are efficiently covered by powerful realtime in-circuit emulation of single or multiple microprocessors.

We can still claim a unique record of progressive advancement in the PMDS product range; for instance, the single-user PMDS I introduced in 1979 can still be economically updated to become a multi-user system with all of the features described in the following pages.

2.1 Configuration of a development system

The Philips microcomputer development sytems can be supplemented by a number of hardware and software options to provide realtime emulation capabilities for a range of different microprocessors. The currently available range of support packages is being continuously developed and updated. Information on additional support facilities will be issued from time-to-time. Up-to-date information can be obtained on application to the local Philips Test and Measurement Organization.

A support package generally consists of the following:

- Software: cross-assembler(s)
- Hardware: microcomputer adapter box(es) (MABs)
- General features: test pods, emulation memory

2.1.1 Cross-assembler

The cross-assembler translates assembly language input into object code and debug files.
A range of cross-assemblers is provided for the various microprocessors available and in common use today.
Some of the many advanced feature common to all PMDS cross-assemblers are:

- Modular support to allow structured programming
- MACRO facilities
- Conditional code generation
- Memory type specification
- Paging support
- Cross reference listing
- extensive set of error messages

The standard instruction mnemonics of the supplier of the microprocessor/computer are used when creating source programs.

2.1.2 Microcomputer adapter box (MAB)

The MAB is the hardware unit acting as an interface between the PMDS and the user's prototype. It is connected to the UDU (Universal Debug Unit)in the PMDS via two flat cables and to the user's prototype via a flexible cable (approximately 48 cm) and the emulation probe which is inserted in place of the microprocessor on the prototype.

The MAB provides:

- a zero insertion force socket for the target microprocessor. Generally, the highest speed microprocessor is inserted to guarantee real time emulation.
- a zero insertion force socket (simulation socket) to provide fixed input levels to the emulation probe when no prototype is available.
- two trigger output for an oscilloscope via BNC connectors
- BNC clock input for an external pulse generator
- a socket for a PM8820 test pod (only in the case of 8-bit microprocessor/controllers)

In order to reach the top emulation speeds without insertion of WAIT STATES when addressing memory located in the PMDS, the fast static memory boards should be used.

2.1.3 Emulation and debugging

The system provides the following emulation and debugging facilities

- Universal 8/16 bit trace and emulation memories
- Realtime both in emulations/simulation
- Triggering external devices
- Tracing/triggering on external events
- Multi-processor emulation
- Target hardware checkout facilities

Emulation is the operation of the prototype CPU under PMDS control. The CPU is removed from the prototype, and replaced by an emulation probe connected to the microcomputer adapter box (MAB). The MAB contains an equivalent CPU (in some cases a special version is used).

Simulation: if no prototype is available, the emulation probe may be plugged into a simulation socket on the MAB, allowing software testing. These actions are achieved by using a universal debug unit together with an MAB which is specific to the target CPU, or to the CPU family, and with the debug software, which is supplied with the MAB. When the optional trace/counter card is present, additional functions are also provided.

2.1.3.1 Universal Debug Unit -UDU

The primary function of the Universal Debug Unit is the control of emulation processes. All control logic, emulation and trace facilities are made independent of target microcontroller architecture. The UDU consists mainly of extra circuits built into the PM4401 cabinet, plus two external units: the Microcontroller Adapter Box (MAB) and Test Pod. The MAB adapts various microcontroller architectures to the UDU. It is connected to the main unit by a dual flat cable. Test pods contain 8 test probes with programmable threshold voltages. Probes can be placed anywhere on the circuit to be tested and the detected signals are used to control emulation.

The internal hardware of the UDU is completely microcontroller independent and requires no manual operation by the user - even when changing from one test microcontroller to another.

The following facilities are provided as standard:

- Memory map: The address space is partitioned into segments (typically 256 bytes for 8-bit CPUs and 4 Kbytes for 16-bit CPUs). Each segment can be defined as being guarded, in emulation memory, or in the prototype, and separately defined as either ROM or RAM. Emulation or prototype memory can be examined, and except for prototype RAM its contents may be changed at will.
- I/O map: for target CPUs having separate I/O these can be mapped to the prototype, to the VDU or to disk files.
- Clock source can be defined as being in the prototype, provided from the MAB at a choice of frequencies, or external.
  Emulation can be done at the maximum rated clock speed for the target CPU: see the relevant support pages.
- Trigger settings: the two trigger registers can each be set to break off emulation, to produce an external pulse, for one to arm the other, or off.
- Trigger conditions: Each trigger register has a 256 times match counter and is 48 bits wide. Each bit can be specified as 1,0 or X (don't care). Address and data may be set in hex or binary, and may also be given symbolically, provided PMDS debug files are present. Control may be specified in binary or in mnemonics such as MR, MW, OPC, etc.
- Break conditions: emulation break will occur on Trigger Match, access to Guarded Memory or I/O, or on Event Counter count-down if set in this mode. Other break conditions such as Powerfail, Reset, Not Ready, are target-dependent, and most of these may be selectively turned off.
- Registers: The CPU registers may be examined and changed at will.
- Run: The CPU may be instructed to run from any specific address.
- Step: The CPU registers and flags are displayed at every step. Dependent on the target CPU, various options are available for stepping on jumps, calls, qualified steps, or external events, with the CPU running in realtime between steps.
- Resource transparency: The Universal Debug Unit does not use any of the target CPUs address space or I/O space, and all interrupt facilities of the target CPU are available for the system under test.

Extension Debug Module

Up to four universal debug units may be operated simultaneously. All debug parameters described above may be set independently for each debug unit. Two or more debug units may be controlled from a single PMDS 2 unit or from a single workstation of a PMDS 2, or they may be controlled separately from different workstations. When two or more different target CPUs are to be controlled from a single workstation, the appropriate multi-debug software must be loaded (available combinations on request).

Multi-debug facilities: Two additional features are provided for optional use in the multi-debug mode of operation:

- Master/slave: One target CPU may be defined as a 'Master' and one or more as 'Slaves'. Linking the Debug Units (by BNC cables) allows the slaves to run synchronously with the master.
- Force trade: A second link between the debug units will force a synchronized trace entry for each slave, even if the slave's own trace condition is not true.

Trace Card

This card provides a 'history' of events relating to an emulation break. The trace memory is 255 words deep x 48 bits wide. Allocation of the 48 bits depends on the target CPU, typically:

- For 8 bit CPUs: 16 bits address, 8 bits data, 8 bits control, 16 bits user-definable from two PM8820 probe pods.
- For 16 bits CPUs: 16 to 24 bits address, 16 bits data, 8 bits control, 8 bits user definable either from a PM8820 probe pod or as high-order address bits.

This facility can be used like a logic analyzer: the user may define which states are captured and this can be done by pre-, center- or post-triggering. Also, a timer/event counter and a display of the data captured by the test pod PM8820 are supplied by this card.

Emulation Memory Cards

Each universal debug unit, whether in the master system or in an extension debug module, has four slots available for emulation memory cards. The two basic types of emulation memory are:

- Fast static: access time 130 ns including mapping time
- Dynamic: access time 450 ns

Use of the fast memory will provide real time emulation, without wait states, up to the values given in the 'support' data for each target CPU. With dynamic memory, wait states may be inserted both on account of the memory access time and to allow refresh.
For certain target CPUs, only the fast emulation memory is usable, as specified in the relevant support data.
The two types of emulation memory may be mixed, subject to certain restrictions. These restrictions do not affect the possibilities of mapping to different logical addresses for the target CPU.

The following emulation memory cards are available:

32 kbytes Dynamic
64 kbytes Dynamic
8 kbytes Fast
16 kbytes Fast
8 kbytes Dynamic
16 kbytes Dynamic
256 kbytes Dynamic
64 kbytes Fast

Certain microprocessors have their own special emulation memories.

PM8440 No-break MAB Switch

The PM8440 is a switch which may be connected at the microcomputer adapter box (MAB) and allows the MAB to be disconnected while the system is still powered, thus permitting a user to exchange the MAB for one of a different type without disturbing other users.

Special trigger/trace feature PM8820 Multilead Probe Pod

One or two pods may be usable, depending on the target CPU and system configuration. Each pod provides 8 connections to user defined points. These pods can not only be used for tracing circuit activity, but also as extra trigger words: emulation can be stopped as a consequence of the pod input signal, on user specification.

Input impedance: 4 MΩ/6 pF Max. Input +50 V
Threshold: default 1.5 V, software settable per pod -3 V..+11 V in 0.1 V steps
Sensitivity: max 250 mV threshold overdrive

The debugger is a software package which controls all the debug hardware contained in the PMDS and the MAB. Most of the debugger commands are universal. However some provide information dependent on a given microprocessor architecture.

2.1.4 PMDS Peripherals

- PM4492/10 high speed printer
- PM4499/20 M User terminal with 12-in diagonal green phosphor screen, 80 cm x 24 lines. QUERTY keyboard plus numerical keypad and 8 function keys. Any terminal connected to PMDS 2 can control any debugger on the system.
- PM4494 M External cabinet containing 22 Mbytes Winchester disc.
- PM4495 M External cabinet containing 2x 22 Mbytes Winchester disc.
- PM4496 M streamer tape backup
  Users digital tape cartridges: capacity 20 Mbytes. Streaming time for 5 Mbytes; approx. 2 minutes and for 20 Mbytes; 7...8 minutes.
- PM4498 EPROM programmer
- PM4476 M ethernet controller

2.2 Microcontroller development system -PMDS 2/PM4422

PMDS 2 provides a multi-user software development environment using a UNIX-based operating system. The PMDS/PEDS implementation of UNIX has been specifically designed to bring a new efficiency to microcomputer development, to cut programming costs and those of both software and hardware development.

- UP to 7 users
- UNIX-based operating system
- Multi-tasking
- 512 kbyte to 6,2 Mbyte RAM: 22 to 156 Mbyte mass storage
- Up to 4 emulation units
- Field-expandable
- Support for wide range of MPU's/MCU's

The PM4422 multi-user development system consists of a PM4402 master system plus, optionally:

- mounted within the master system

  -Additional terminal computer cards
  -Additional system memory cards
  -Interface cards
  -Universal debug unit cards
  -Bus extender card

- Connected externally:

  -Terminals
  -Printers
  -Microcomputer adapter box
  -EPROM programmers
  -Extension debug units
  -Extention disc units
  -Streamer tape back-up unit

2.2.1 Hardware specification -PMDS 2

PMDS 2 features a multibus-multiprocessor architecture, system and emulation functions are totally separate. All benefits of the transparent architecture featured in PMDS 1 are incorporated in PMDS 2. PMDS 2 is a distributed processor system, a master 68000 CPU running a UNIX based operating system permits up to 7 independent user work stations. The processor is based on a double Eurocard with a 16 bit data path, 19 program registers, 56 instruction types and a 24 Mbyte addressing range.

Multi-user operation under a time sharing scheme is achieved by using up to 4 Tcom cards, each card housing 1 16-bit slave CPU (SP16C/10). In addition to the slave processor, the Tcom card also carries 128 Kbytes of memory and two RS232 serial interface ports. On card memory is also accessible to the 68000 CPU if it is not allocated to a particular slave processor. Figure 1 illustrates PMDS 2 hardware architecture.

The main unit houses a UDU universal debug with expansion capabilities of up to four units each with a dedicated bus driver. This allows multiprocesssor emulation by the system. Each debug unit contains a control microcontroller board and a real-time board which looks after memory and I/O mapping. The UDU card also contains dedicated emulation memory as 32 K or 64 K RAM.

In both PMDS/PEDS technologies the UDU unit is identical in function (see section on UDU debug).

The master system memory consists of 128 kbytes of RAM. Disc storage is in the form of one built in diskette drive and one 5,25 inch Winchester hard disc drive. The floppy drive provides double-sided double-density recording, resulting in a capacity of 320 Kbytes (formatted) per disc with the winchester providing from 5 to 21 Mbytes of storage area.
The PM4422 master system includes:

-An integral VDU with 12 inch diagonal screen, 24 lines each 80 characters. A video output is provided for an external monitor.
-A free-standing keyboard, standard QWERTY layout plus numeric pad, also control keys and 8 user-definable function keys.
-One mini floppy disk drive, formatted capacity 740 Kbytes, average access time 185 ms.
-One hard disk unit:
    Formatted capacity 22,3 Mbytes (PM4402/20).
    Latency 8,3 ms, head setting time 15 ms, t.t.t. step time 3 ms.
-Interface for printer, RS 232 subset, 9600 baud.
-Interface for PM4496 streamer tape unit.
-One terminal computer card 'Tcom' (see PM4477 M details.) A domestic computer (DOMCOM) with 68000 CPU, clock 8 MHz, with memory management unit controlling RAM of 512 Kbytes (PM4402/20) or 1,256 Mbyte (PM4402/40).
-A converted PMDS 1 has 192 Kbytes (PM4401/20).
-A real-time/date clock with battery back-up.

The DOMCOM (68000) can access all system RAM (but not emulation memory). UNIX runs on the DOMCOM and processes called by UNIX can run on the 68000 or can be allocated to the first available TCOM where they run on one of the SP16C/10 CPUs, depending on the process type. The RS 232 interfaces on the TCOM card(s) are not connected with the SP16C/10, they connect directly to the CPU system bus. As the system is expanded with additional TCOM cords, both CPU and memory resources are increased to cope with the possible additional demands of extra users.

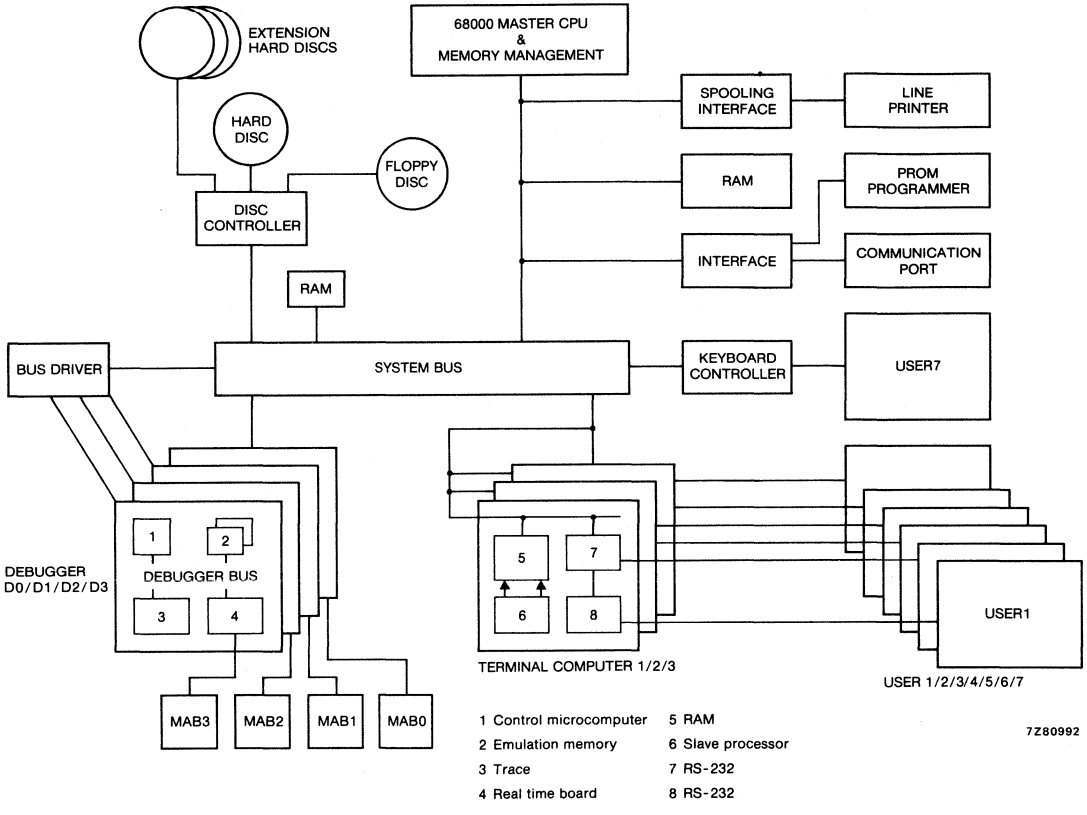


Fig. 2 PMDS 2 hardware architecture

2.2.2 Hardware options PMDS 2

PM4475 bus extender card IOTZ.
This card provides an interface to the first extention debug unit.
PM4477 M terminal computer card TCOM
CPU: 16 bit type SP16C/10.
RAM: 128 Kbytes (also accessible to DOMCOM).
Two RS 232 ports for terminals/peripherals/communication; software-settable for: speed 110-9600 baud, parity and other characteristics.
Note that one TCOM card is included in the master system as standard.
PM8400 universal debug unit
PM8405 extension debug unit
PM8410 trace card
PM8411 to PM8419 emulation memory cards
PM8418 256 Kbyte RAM card
PM8409M 1 Mbyte RAM card
These RAM cards may be used to provide additional system memory, as well as being one of the emulation memory options.
M4476M Ethernet local area network controller

2.2.3 Software specification (PMDS 2)

The PMDS 2 system software, which is supplied on diskettes, consists of a modified UNIX operating system and optional system processes such as Assemblers, linkloaders, etc (additional command support information is available from the PMDS 2 user manual Vol 2). The PMDS 2 system provides improved software development and a hierarchical file system permits logical, modular development of programs.

The UNIX operating system

The UNIX system consists of a kernel of operating system primitives, the kernel schedules tasks and manages data storage, a 'shell' around the kernel interprets user commands and invokes the primitives in various combinations. In this way the desired operation or sequence of operations are performed. Figure 3 shows the software structure of PMDS 2.

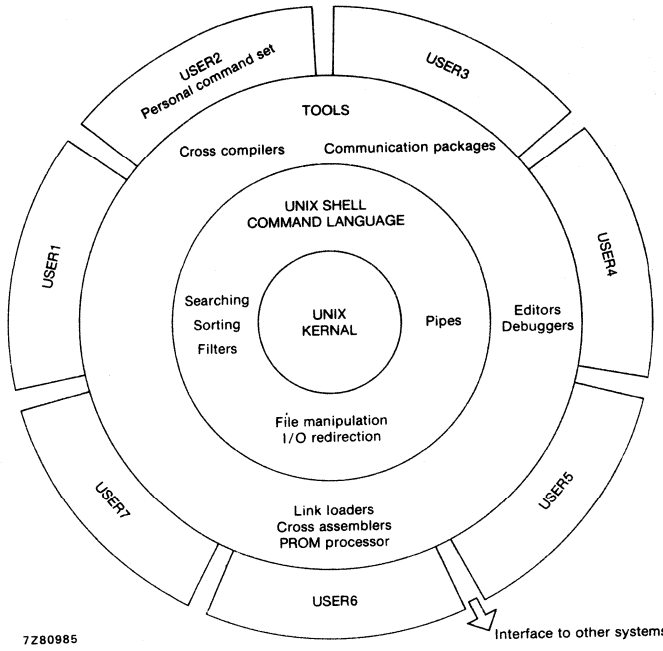


Fig. 3 PMDS 2 software structure

The UNIX-based operating system provides:

-multi-tasking for each user
-hierarchical file system
-password protection
-nine levels of file access permission:
 read/write/execute for file-owner,
 members of his group and others
 spooling of printer output
 input/output redirection
 'make' utility
-Other 'universal' software (supplied as standard):
-Linker, producing microprocessor loadfiles from object code files.
-PROM processor, Text editor (ed)

2.2.4 Software options

Utility software (option):
Cross-assemblers, cross-compilers, debug software (option):
PM8394 M screen editor
PM8396 M Source control code system (SCCS)
PM8399 M general purpose communication package.
PM8393 UUCP
PM8397 NROFF
Details of other utility packages are available on request.

2.2.5 Ordering information

Due to the many available options and the high level of flexibility offered, it is advisable to contact your local T&M representative for guidance when configuring your system.

2.3 Microcontroller development system -PMDS 3/PM4423

Features of PMDS 3 include:

- IBM PC-AT based
- XENIX-based operating system
- Support for a wide range of MPUs/MCUs
- High-level language debug
- Highly efficient
- Ensures shorter development times

PMDS 3, the latest in the range of Philips microprocessor development products, is based on the IBM PC-AT. The system combines this industry-standard PC with the Philips integration and Debug Station (PIDS), the result is a flexible and highly efficient development system.

The PIDS is connected to the PC-AT via an interface board which extends the PC's AT bus. The PIDS AT expansion box contains a Universal Debug Unit (UDU; see the separate chapter on the UDU unit) and a real-time trace board as standard. Additional emulation memory can be added as an option if required. Up to two UDUs may reside in the AT expansion box, giving the possibility of two simultaneous realtime emulation sessions. Microcomputer Adapter Boxes (MABs) are connected to the UDU for each different microprocessor type to be emulated. The MAB is the only unit that needs to be exchanged when switching from one type of target micro to the next.

PMDS 3 uses the XENIX operating system (an upgrade from UNIX) together with Cross-software packages and debugger software to provide a complete software and hardware development environment. In addition, with its friendly user interface and high-level language debug facilities, PMDS 3 represents a highly integrated and efficient microprocessor development system that will shorten development times and lower costs. Figure 4 illustrates the PMDS 3 system architecture.

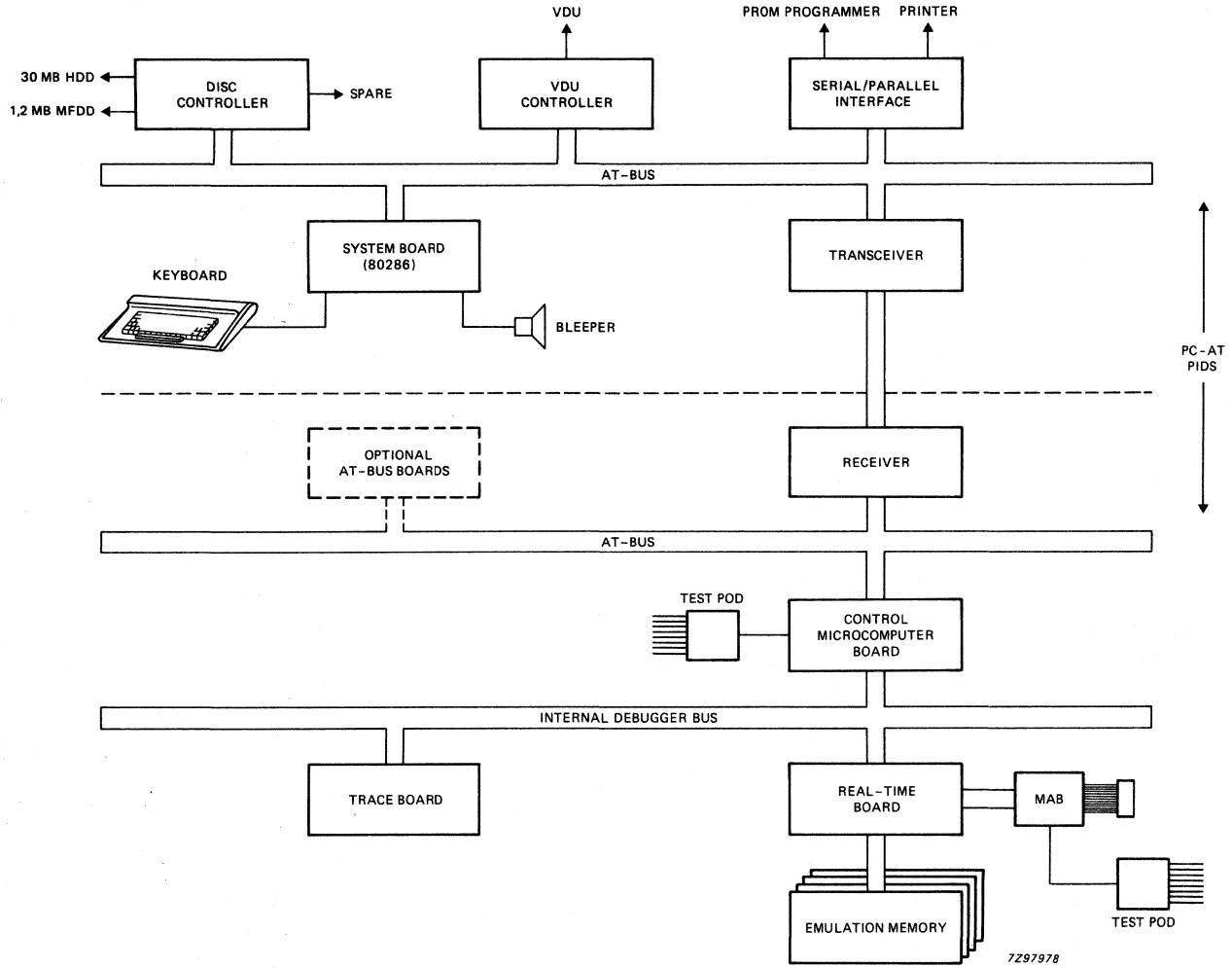


Fig. 4 PMDS 3 system architecture.

2.3.1 Philips Integration and Debug station -PIDS

The PIDS consists of:

- Transceiver card to be installed in the PC-AT (requires two free full AT-bus slots).
- Cable to interconnect PC-AT and AT extension box.
- AT extension box with 12 full AT-bus compatible slots, containing:
  - Trace board
  - receiver card
  - universal debug unit
  - 16 Kbytes fast emulation memory
  - 7 free slots
- Microprocessor specific emulator (MAB).
- Cross assembler.
- Assembly level debug processor.

2.3.2 Hardware specification -PIDS

All boards have dimensions identical to PC-AT add-in boards

- Transceiver card; installed in full AT-bus slot in PC-AT; occupies 2 locations.
- Cable to interconnect PC-AT and AT extension box; max. length 1,8 m.
- Receiver card; installed in slot 1 to AT extension box occupies 2 locations.
- AT extension box; 12 full AT-bus compatible slots; operating positions: table-top or on its side; can contain two universal debug units with options.
- Trace board.
- Universal debug unit: occupies two slots; 4 hardware breakpoints; 48 bits wide address, data, control bus and 8 to 16 external signals, to be used separately or armed (one pair), provides break or pulse.
  Two qualifiers; 48 bits wide data, address, control bus and 8 to 16 external signals, to be used for qualified data capturing and stepping.
  Memory map: 256 segments to be declared as RAM, ROM or guarded I/O map for 512 I/O ports.
  Clock source from prototype, external crystal or PIDS.
  Emulation break on (target processor dependent - can be switched off); power fail, reset active, halt-line active, not ready, trigger matches, forbidden memory or I/O access.
  Run and step function.
  8 data input lines.
- 16 Kbyte fast emulation memory.
- Microprocessor dependent emulator (MAB); only the emulator needs to be replaced for support of another microprocessor.
  Emulation and simulation modes: Trig.out, Trig.in, external clock input.
  Microprocessor dependent: support for various packages, power on/off switch, support for other family members.

2.3.3 Hardware options -PMDS 3

- PM4950 Universal Debug unit.
- PM4951 Trace board logic state analyzer (255 x 48 lines) with event counter; qualified data acquisition using the qualifiers; pre-, center-, and post-triggered data acquisition; event counter counts up, down, time and events; can be used in break conditions.
- PM4952 16 kbyte Fast emulation memory.
- PM4953 64 kbyte Fast emulation memory.
- PM4956 128 kbyte Dynamic emulation memory.
- PM4957 256 kbyte Dynamic emulation memory.
- PM4958 512 kbyte Dynamic emulation memory.
- PM4959 1 Mbyte Dynamic emulation memory.

2.3.4 Software specification -PIDS

The UNIX hierarchical file system encourages logical, modular development of programs. A progammer can produce software tools which are tailored to a particular application. UNIX utilities permit the 'pipelining' of the output of one process to the input of another.

All files are tagged with the date and time of the last modification made. Using the 'make' utility enables the most up-to-date version of a complete program to be produced, while recompiling or reassembling only those modules which have been altered, or those dependent on such changes.

PIDS is supplied with a cross-assembler, an assembly level debug processor and an efficient user interface. Figure 5 below shows PMDS 3 software structure.

- Cross-assembler based on the PCP architecture.
  Universal command and syntax structure.
  Uses mnemonics of chip manufacturer. Includes a combiner and a pre-processor with macro facilities.
- Assembly level debug processor controls universal debug unit and emulator.
  Examine and change memory, register and I/O.
  I/O simulation.
  Extensive step and run commands.
  Monitoring statements combining hardware and software breakpoints with appropriate actions.
  Procedures.
  History buffer.

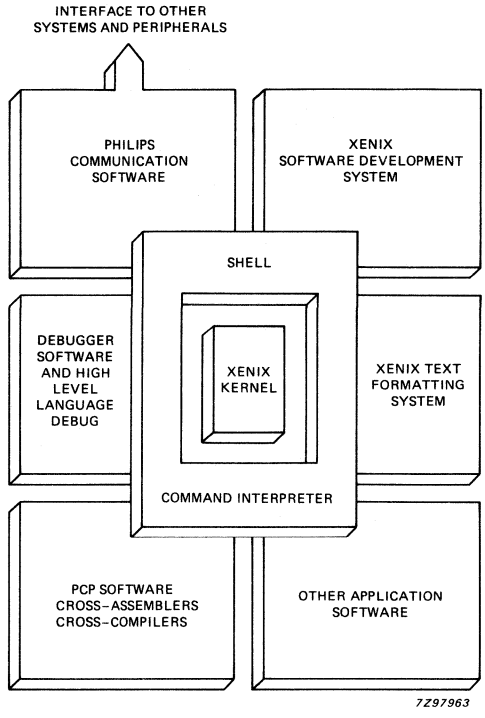


Fig. 5 PMDS 3 Software structure.

2.3.5 Software options -PMDS 3

Cross-compilers: PL/M for most microprocessors.

- Cross-assemblers: for all supported microprocessors.
- Assembly level debug processors: for all supported microprocessors.

2.3.6 Ordering information

PIDS is supplied with support for one microprocessor. New support facilities and options are introduced regularly. Please check with your local Philips representative for up-to-date information.

PM4915	PIDS for 80C51 family
PM4917/10	PIDS for MAB84X1 family
PM4917/10	PIDS for 84CXX family
PM4921	PIDS for 8048 family

3.0 SDS - THE STAND ALONE DEBUG STATION

3.1 System Architecture

Features of the SDS system include:

- Low-cost full real-time emulation
- RS232-C/V24 connection to IBM PC VDU or other hosts (e.g. VAX, Intellec)
- firmware in-line assembler
- symbolic debug software on IBM PC
- self contained architecture
- Intel based command language

The SDS-80C51 is a box presenting on one side the emulation cable and probe and on the other, two RS-232C connectors serving the users CRT interface and a host computer. The system consists of two boards; the emulation board and a control board.

Firmware required for emulation is carried on the control board which is designed around an 8031 CPU. This processor accesses external program memory ROM via an 8-bit address bus and is interfaced with two RS-232 serial ports. One serial port may be used for communication with an MDS or PC system from which assembled object code programs may be downloaded, the other port is used by the CRT terminal.

The emulation board is connected to the emulation cable and probe, this also contains an 80C51 bond-out chip (or an 84X1/84CXX bond-out chip) from which address and control data may be extracted when required.

The power supply is entirely self-contained within the device. Figure 6 shows a block diagram of the SDS architecture.

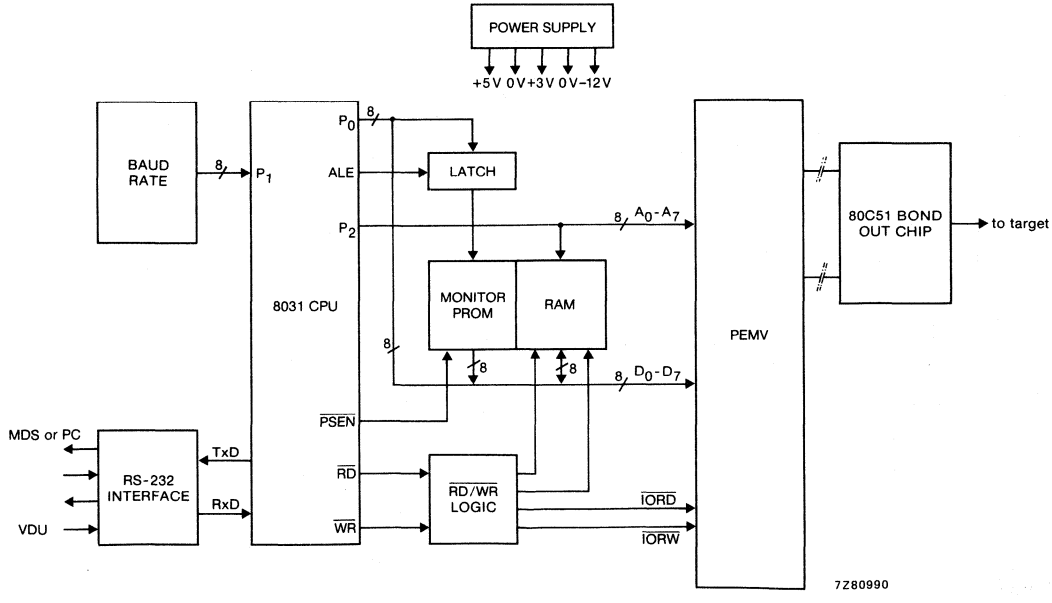


Fig. 6. Block diagram of SDS-80C51.

3.2 SDS System software and modes of operation

Monitor software in 12 K of EPROM enables the user to perform the following functions:

- To communicate with SDS using a CRT terminal, PC or other host computer
- To execute user programs in real-time or single step
- To set breakpoints on program addresses, ranges of program addresses, branches or register values
- To assemble individual 8051/84X1 assembly language instructions into memory, and disassemble memory into assembler mnemonics or equivalents line by line
- To examine and modify memory locations, registers and bits in the on-board program memory and in the microcontrroller's data memory and registers
- To upload or download object code programs from a microcomputer development system or a personal computer system.

Modes of operation

The SDS-80C51 system has five modes of operation:

- Interrogation (the default)
- Assembler mode
- Continuation mode
- Real-time emulation mode
- Single-step execution mode.

Interrogation mode is entered following power-up or any system reset. An asterisk prompt is displayed at the left margin indicating that the system is ready to accept a command. All commands are entered through the interrogation mode.

Assembler mode permits the user to enter instructions in 8051/84X1 assembly language. It then assembles the instructions directly into on-board memory. Assembler mode is initiated by the ASM command, and is terminated by pressing RETURN. The mode also permits disassembly of on-board memory into 8051 or 84X1 instruction mnemonics.

Continuation mode allows the user to enter a list of values for setting memory contents without repeating the memory type keyword.

Real Time emulation mode permits the user to run code stored in program memory. Emulation starts when the user enters a GO command while in interrogation mode. Real time emulation is controlled by breakpoints set by the user. If a breakpoint is encountered by the program, the program halts after executing the instruction that contained the breakpoint address. If breakpoints are not used, the program runs until the user terminates the program.

Single step execution mode allows the user to specify the number of steps and run the program one instruction at a time, breaking between steps. Between steps the system also displays instruction mnemonics and register values.

3.3 SDS for 80C51/8051 family

The SDS or Stand-alone Debug Station can be connected between a host computer (PC) or VDU terminal and the user's circuit under test (target). It gives the user the ability to debug his software programs, and integrate these with his target, by means of emulation of the microcontroller type Intel 80C51 on which the target is based. It is also possible to write simple programs on the SDS.

To perform these functions in a user-friendly and competitive way, the SDS is equipped with the following features:

- Emulation memory to simulate the CPU's addressable memory space (size 8 kbytes).
- Breakpoint registers to break the program flow at any convenient moment in realtime (4 address breakpoints or 1 address and 1 address-range breakpoints, 1 breakpoint on branch instructions, 1 non real time breakpoint on memory/register values).
- A trace memory to trace program flow in realtime (size 2048 lines deep, tracing all addresses, opcodes/data bytes, and all 4 8051/80C51 ports).
- Two RS232-C serial interfaces; one to be connected to a VDU or PC and the other to be connected to a host computer.
- Interface lines to external equipment indicating emulation busy, memory read, opcode fetch and emulation-break from outside.
- Software commands to run or single step the user program, to upload/download programs, set breakpoints at addresses or data values, examine memory, set memory bits in hex or with the in-line assembler.
- Symbolic Debugging by means of an additional software package, running on IBM PC/XT PC-DOS 3.1. It displays labels or enables using them in commands instead of addresses, includes - where possible - module-names, does type-checking to indicate the right label for the type of memory concerned (code or xdata), displays help information about all or single commands, adds information about error messages, performs terminal emulation of the PC to run the SDS, and gives program up/down loading from the PC disc while running the SDS.
- A manual with complete with application examples and all information regarding the operating of the SDS.

A complete development environment based on the SDS should preferably include the following products:

- The SDS itself, which includes the RS232 cable to the PC.
- The emulation probe with the bond-out chip, which is a separate item.
- An IBM PC/XT with hard disc, floppy disc, RS232/V24 interface and 256 Kb memory running PC-DOS 3.1 or MS-DOS 2.11 (or later releases); any IBM-PC-compatible PC is also suitable (e.g. P3100).
- The cross-assembler 80C51 for DOS and possibly also the PL/M-51 compiler. Any other brand of cross-software may also be used; however, the SDS accepts only an Intel HEX formatted file and the symbolic debugger will not show symbols.

The IBM PC specification is only a recommendation; a PC type IBM PC (without hard disk) will also work, but this is not ideal for a development engineer.

Specifications

RS232-C	: Baud rate selectable from 300 to 19 200 baud File format: Intel HEX Recognize Xon/Xoff
Trace	: 2048 lines

Memory		Internal/external code memory fetches; external data memory fetches; all port data Selective tracing on cycle type
Emulation memory	:	8 kbytes No wait states
Clock speed	:	Permits operation up to 12 MHz, realtime
Power down	:	Supports power down and idle mode
Signals to external equipment	:	Output from SDS: ALE: indicates valid address CLK: indicates opcode read PSENE: indicates byte read EMUL: indicates running user program Input to SDS: EXTBRK: stop emulation by external pulse
Size	:	300 x 66 x 235mm (wxhxd)
Weight	:	5 kg (approx)
Power supply	:	110/220 V AC, 50/60 Hz

3.4 SDS for 84C00/8400 families

The SDS or Stand-alone Debug Station can be connected between a host computer (PC) or VDU terminal and the user's circuit under test (target). It gives the user the ability to debug his software programs and integrate these with his target, by means of emulation of the microcontroller type Philips MAB8400/PCF 84C00 on which the target is based. It is also possible to write simple programs on the SDS. The SDS-84X1/84C00 has the same features as described for the SDS-8051/80C51, but it has no trace memory nor symbolic debug software package.

NB: It is possible to use 8048 assembler with some additional macros. Contact your local T&M supplier for more details.

Specifications

RS232-C interfaces	:	Baud rate selectable from 300 to 19 200 baud File format: Intel HEX Recognize Xon/Xoff
Emulation memory	:	8 Kbytes
Clock speed	:	No wait states
Signals to external equipment	:	Allows operation up to 9,5 MHz Output from SDS: CLK: indicates opcode read PSENE: indicates byte read EMUL: indicates running user program Input to SDS: EXTBRK: stop emulation by external pulse
Size	:	300 x 66 x 235 mm (wxhxd)
Weight	:	5 kg (approx.)
Power supply	:	110/220 V AC, 50/60 Hz

3.5 Ordering information

MAB84X1/PCF84C00 families

PM4727	:	Stand-alone debug station for the 8400 N/CMOS
PF8117F	:	Cross-assembler for the 8400 N/CMOS
PM8443	:	Probe for support of MAB8411, 8421, 8461
PM8443/10	:	Probe for support of PCF84C21, 84C41, 84C81, 3343, 3315
PM8443/40	:	PCF84C12 28 to 20 pin emulation adaptor plug

Note: A minimum configuration must include a PM4727 and either a PM8443 or PM 8443/10

MAB8051/PCB80C51 families

PM4726/10	: Stand-alone debug station for the 8051 N/CMOS family excluding probe
PM8453/40	: Adaptor for conversion of 40 pin DIL probe to 44 pin PLCC socket
PF8002F	: Symbolic debugging package for IBM PC-MSDOS
PF8112F	: Cross-assembler for 8051 N/CMOS family on IBM PC-DOS
PM8453/20	: Probe for 80C451, supports 68-pin PLCC. If DIL is required order additional option PM8453/30.
PM8002F/20	: Symbolic debugging package for IBM PC-DOS for 80C451
PM8453/30	: Adaptor for conversion of 68-pin PLCC probe to 64 pin DIL socket.

Note: Minimum configuration must include a PM4726/10 and a PM8453 or PM8453/20

4.0 ENGINEERING AND DEVELOPMENT SYSTEMS -PEDS/PM48XX

Features of PEDS include:

- Softkeys for fast learning
- UNIX-based operating system
- Up to 4 users
- 256 Kbyte to 3,2 Mbyte RAM: 22 Mbyte mass storage
- Up to 4 emulation units
- Field-expandable
- Support for wide range of MPU's/MCU's

PEDS was produced to assist in the development of small/medium sized projects and the system provides a powerful tool for single handed software and hardware production. PEDS contains all system benefits incorporated in PMDS 1.

4.1 PEDS -System architecture

PEDS has a distributed processor architecture. A 68000 master CPU runs the operating system and is assisted by an SP16C/10, a 16 bit microprocessor. Both processors have 128 kbyte memory accessible through a local bus. Using the system bus, the 68000 can access all memory which may be extended up to 1 M bytes. A memory management unit helps the 68000 to switch tasks without having to save memory on disc. These CPU's run concurrently and they communicate using mutual interrupts and messages exchanged in shared memory.

A disc controller handles both an 11 Mbyte Winchester hard disc and a 645 Kbyte 5,25 inch floppy drive (both relate to formatted capacity).
The heart of the development system is the UDU, again the UDU is not dependant upon the chosen target micro. The CPU and the UDU exchange alot of data, so the UDU connects directly to the system bus for speed reasons. PEDS carries one UDU unit but may be expanded by up to four units.

Connected to the UDU is the Microcontroller Adapter Box (MAB). As in PMDS 1/2 the MAB is the only hardware to exchange when changing the target micro. The MAB is responsible for translating the information of the micro under emulation into a format standard for all supported micros. Figure 7 shows the PEDS system architecture.

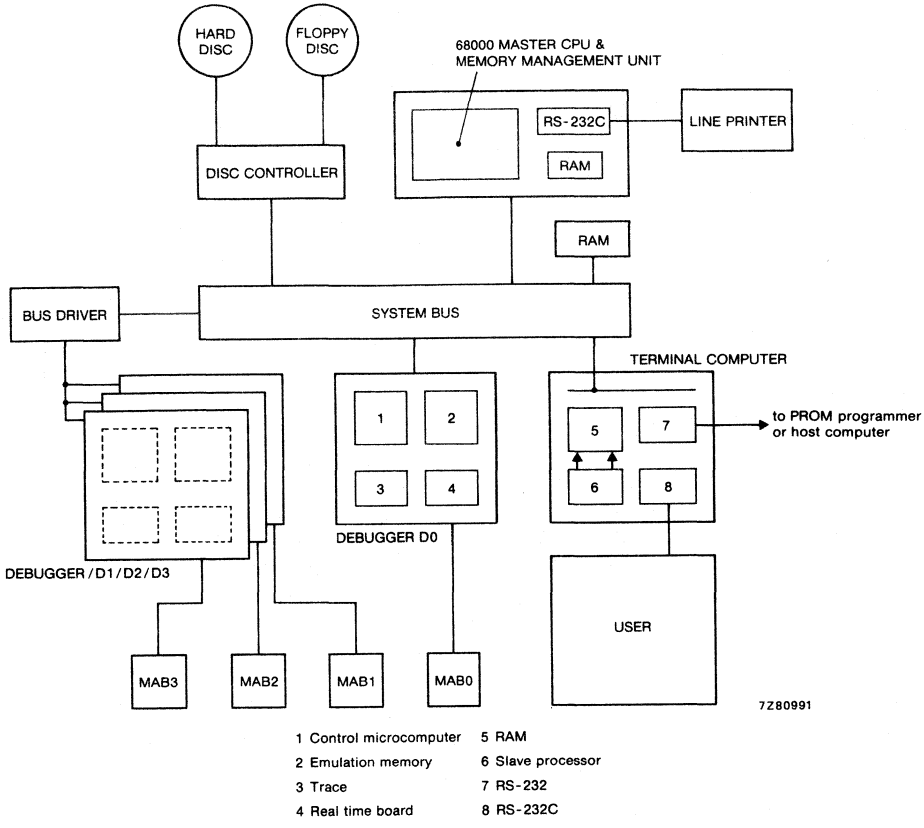


Fig. 7. PEDS system architecture

The PEDS master system consists of:

- 256 kbytes RAM and 16 Kbytes ROM
- 22,3 Mbyte hard disk
- 740 kbyte floppy disk
- 68000 CPU + Memory management unit
- two RS 232 C general interfaces
- dedicated printer interface
- softkey based multi-tasking operating system

an emulation subsystem

- universal debug unit
- 16 Kbyte fast emulation memory

and microcomputer dependent

- microcomputer adapter box
- assembler and emulation processor

For correct functioning, a terminal must be connected using standard V24 communication.

PEDS can be expanded by :

- a state trace facility with disassembly
- additional memory cards for system and emulation
- additional emulation stations
- a printer
- a PROM programmer
- utility software
- compilers
- additional support for other microcomputers
- extra VDU terminal
- extra Tcom card for 2 additional users
- hard disc unit
- formatted capacity 10,8 Mbyte (latency 8,3 ms, head setting time 15 ms, t.t.t. step time 1,5...3 ms)
- Minifloppy disk drive:
- formatted capacity 740 kbyte (average access time 185 ms)
- Interface for printer: RS 232 C subset
- Interface for terminal: RS 232 C
- Interface for peripheral: (i.e. PROM programmer or extra VDU terminal), RS 232 C
- all interfaces have software settable parameters such as baudrate and parity. 68000 CPU: clock 8 MHz with memory management unit 256 kbyte RAM
- a real time/date clock with battery back-up
- softkey control
- multi-tasking: editing during emulation run possible
- symbolic addressing
- memory instruction and state trace disassembly
- memory map of 256 segments
- Each segment can be declared RAM or ROM
- I/O map for 512 I/O ports
- clock source from prototype, external crystal or PEDS at a choice of frequencies
- 2 general trigger registers provide break, pulse or arm functions
- 2 general qualifier registers
- all registers are bit settable on address, data and/or external lines
- emulation break (target dependent; can be turned off) on conditions like power fail, reset activated, halt line, not ready. Further on trigger match, forbidden memory access or I/O, or event counter count reached.
- registers and memory can be examined and changed at will
- step function

4.2 Hardware options

PM8418 256 kbyte system memory
PM4476 M ethernet controller
PM8409 M 1 Mbyte RAM card
PM4485 M 10 minifloppies (96 tpi)
(see also, memory backup options.)
PM4477 M terminal computer (Tcom) card

Emulation

(Please refer to PM8400 data.)
- PM8410 M Trace board

This board gives logic state trace (255 x 48 lines) and event counter possibilities.
- PM8412 64 kbyte dynamic emulation memory
- PM8418 256 kbyte

- PM8414 16 kbyte fast emulation memory
- PM8419 64 kbyte

- PM8440 MAB No-break switch
- PM4475 Bus extender card this card provides an interface to the first extension debug unit.
- PM8405/20 Extention debug unit

Support for other microcomputers:

Please see relevant pages.

Peripherals

PM4499/20 M Terminal (green phosphor)
PM4492 High speed printer
PM4498 Universal PROM programmer

4.3 PEDS Software specification

We realised, that in some situations in a small development unit the operating engineer can't afford to waste time learning a new operating system, but he may require all the facilities that such an operating system offers. With this in mind PEDS was developed with a unique solution; Softkeys.

The Softkey system

As in PMDS 2, PEDS incorporates a UNIX based operating system, but allied to this is the softkey command system. The softkey system is a command layer above the UNIX shell, subsequently all features of the UNIX command language are available to the user of softkeys. Softkeys guide the novice in using the power of UNIX effectively.

By pressing the HELP function, PEDS presents the options that are available to the user. After taking an option, PEDS presents immediately the options in the next stage.

e.g. The user chooses the option 'Micro Development', he then has the choice between Edit, Assemble, Compile, Emulate and Burn PROM's. Choosing Assemble, the softkeys guide the user through the command syntax to the execution of the command. After a command is executed the user does not have to start again at the opening level but is automatically offered the level of his interest. Figure 8 illustrates PEDS software organization.

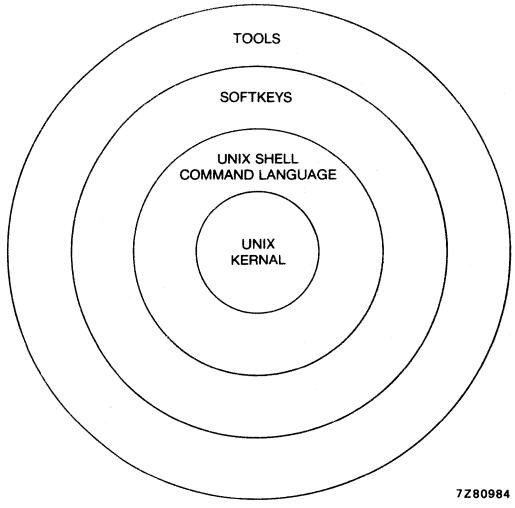


Fig. 8. PEDS software organization.

UNIX-based multi-tasking operating system provides:
- Softkey interface
- hierarchical file system
- password protection
- nine levels of file access permission read/write/execute for file owner, members of his group, and others
- spooling of printer output
- input/output redirection
- make utility
- assembler/combiner, producing load files from source code files. (See the relevant data in the support information pages.)
- emulation processor
- screen oriented PROM processor

4.4 Software options

Utility software
- PM8394 M Screen editor
- PM8395 M C on host
- PM8396 M Source coder control system
- PM8399 M General purpose communication package

Details of other utility packages are available on request.

Assembler/compilers/debug processors

Please see relevant pages.

4.5 Ordering information

Order no.	to support:
PM4816	80C51
PM4817	8400 family
PM4821	8048 family

5.0 THE 8051/80C51 PL/M COMPILER

- fully compatible with the Intel language definition
- produces highly optimized code
- automatic installation with self-test
- runs on VAX/PMDS 2/PEDS/PMDS 3 & IBM PC (DOS) hosts

As with all Philips cross-compiliers, the PL/M compiler is designed to run on various hosts including PEDS, PMDS 2 and PMDS 3. All compilers have the same user interface and are fully source and object code compatible.
The PL/M compiler converts source modules into assembly code, which is input to the PCP combiner (assembler/linker) which produces directly executable code. A special format for relocatable modules allows the PCP combiner to change not only addresses, but also instruction lengths and types to improve code density. This compiler runs on all VAX computers, including MicroVAX under the VMS or Unix operating definition.

Features of PL/M

PL/M is a programming language based on PL/1, but designed specifically for microcomputers. It is a block structured, typed language used for systems programming and control applications. Structured programming with modules, procedures and blocks, makes programs more readable, as well as being easier to maintain and debug.

PL/M supports the data types Bit, Byte and Word in single variables, arrays or structures or combinations of these. Data can be located in any of the possible memory type MAIN, IDATA, REGISTER, AUXILIARY or CONSTANT. It can be stored at fixed locations using AT, or dynamically using BASED pointer variables.

Procedures can either be typed (BIT, BYTE or WORD), in which case they return a value upon completion, or untyped, invoked with a CALL statement. Each procedure has a user-defined and compiler-checked number of parameters. Optional procedure attributes are USING and INTERRUPT (n).

Benefits

PL/M is easy to learn and use, even for novice programmers. Because PL/M is a high-level language, programmer productivity increases dramatically as compared to assembly level programming, giving lower software development costs. This is not at the expense of code size, as the compiler generates very compact code, comparable to the code resulting from direct programming in assembler. Programming in a high-level language has the advantage of increased reliability and results in programs which are easier to enhance and maintain.

Compiler package

The complete PL/M compiler package consists of the compiler, the assembler/linker and associate utility programs: cross referencer, Intel HEX-format converter, communications software (VAX) and remote Login program (VAX). Further programs supplied are a converter to debugger load file format and utility to build libraries.

A run-time support library contains routines for copying to and from different memory types.

Debug support for VAX users. For hardware and software integration, the load file generated on the VAX is transferred to a PEDS/PMDS III connected via an RS232 link, using an error-correcting protocol which guarantees error-free transmission. The code can then be debugged on the PEDS/PMDS III using the MAB.

PL/M-51 compiler. The PL/M-51 compiler is benchmarked to generate considerably less code than the Intel PL/M-51 compiler. It uses a very fast one-pass optimizing process and is fully compatible with the Intel language definition.

PL/M-51 allows full access to all the 8051 I/O and memory - features that are normally available only to the assembly language programmer.
All four levels of optimization are fully supported. The fourth level (level 3) performs overlaying of on-chip RAM variables in addition to performing all of levels 0, 1 and 2.

5.1 Ordering information

Support for the PL/M compiler and the 8051 cross assembler is shown in Table 2 below.

Table 2

Description	IBM PC		VAX/micro VAX		PMDS/PEDS
	XENIX 1.0	DOS 3.1	VMS 4.2	UNIX	UNIX
PL/M-51 compiler	PF8192 M	PF8192 F	PM5437	PM4541	PM4599 M
8051 family cross assembler	PF8112 M	PF8112 F	PM4556	PM4557	PM8362 M

6.0 LOGIC ANALYZERS

6.1 Introduction

Digital technology is replacing analogue circuitry at an increasing rate. More and more in areas ranging from customer products, such as washing machines to avionics in modern aircraft, conventional test equipment is no longer suitable. Digital technology calls for digital test equipment, and the ideal solution comes in the form of our PM3565/PM3570 and PM3632 logic analyzers. Our logic analyzers provide timing and state analysis capabilities to handle software, hardware and compound faults and lend themselves to applications at all levels.

6.2 The PM3565/PM3570 logic analyzers

Features of the PM3565/PM3570 logic analyzers include:

- Configurable logic analyzer with up to 115 channels
- 400 MHz transitional timing for infinite acquisition memory depth.
- Time correlation mode between state and timing with dual screen display
- Performance analysis
- Wide range of 8, 16 and 32 bit microprocessor disassemblers
- Multi-level fully conditional triggering with selective data acquisition.
- IEEE-488 and RS 232C remote control interfaces

Keeping ahead of today's fast-moving technological advances demands the right development environment. Including the necessary tools to monitor every stage of the process: from the development and testing of hardware and software subsystems, right through to total-system integration and performance testing.

With the all-new PM3565 and PM3570 logic analyzers, we give system engineers a powerful set of tools that streamline and speed the development process. Tools that meet the challenge of today's leading-edge technology. The PM3565 and PM3570 are in gear for 32-bit development projects, with no less than 115 possible channels and ultra-fast 400 MHz acquisition.

State analysis - modular system upgrading for up to 115 state channels.

Transitional timing analysis - with a capability extending from 32 channels at 100 MHz up to speeds of 400 MHz, alternately usable for up to 32 additional state channels.

Combi and time correlated modes - for time-correlated state and timing analysis with dual-screen display plus 32 channels simultaneously in state and timing modes without the need for dual-probing.

Performance analysis - the only effective way to check and optimize software performance in the crucial post-integration stage of hardware and software. Performance analysis is such an essential logic analysis function that it is included as standard in both PM3565 and PM3570.

Transitional timing - Captures only those events that the operator wants to see. Transitional timing ensures that even the smallest pulses are captured, but without capturing the same information several times and wasting precious memory. For example, two 5 ns pulses, separated in time by more than 20 minutes, can easily be captured in a single acquisition shot.

Operation

Menus

Upon menu selection, all parameters are entered by softkeys and hex pad. User prompting and error messages appear on the screen
The menus are:

Configuration

The organization of clocks, clock qualifiers, channels, thresholds and state labels, as well as sampling speed (timing) and polarity

Triggering

The definition of the point at which data acquisition is to stop

Selective data acquisition

The definition of which data from the data stream is to be captured

Compare

The definition of which data is to be compared in the main and reference memories

Option

The selection of a required disassembler and parameter setting of the RS232 printer interface, RS232 control interface and the IEEE control interface

Performance analysis

Built-in as standard

Divides code (target system SW) into max. 12 ranges, identifiable by start and end word in Range Selection Menu

Modes of operation

- Event frequency for occurrence of ranges; min. repetition time: 50 ns
- Range period for (averaged) execution and repetition times with minimum and maximum values plus standard deviation
- Complement period for range period of system activity outside max. four selectable ranges

Special features

Ignore	exclusion exceptional/spurious events
Continue	recognition of recursive routines

Data displays

List	listing of state/timing captured data
Time	displays a waveform diagram (timing diagram) representing the contents of the timing analyzer memories
Graph	XY-oriented display plotting memory contents against address values
Table	tabular representation of event frequency, mean execution, repetition and total time period data (performance analysis)
Histogram	XY-oriented bar graph representing event frequency, execution (mean) period, repetition (mean) period, total time and complement period data. Histogram of recursion depth (performance analysis)

Data display with compare

List	irregularities between current and reference memories are highlighted
Time	irregularities between current and reference memories are underlined

Non-volatile memory facilities

Stores up to four user settings; last setting always saved after power-down
Stores all available state/timing reference data, including performance analysis results

Built-in as standard

Universal disassembler package with four user-definable disassemblers in case no particular disassembler from the above table applies. Useful for disassembly of military CPUs, bus systems, etc.
User can specify e.g.:
* disassembler name (9 characters per disassembler name)
* PM 3565/3570 configuration incl. clocks, qualifiers
* translation table and mnemonics (max. 256 per disassembler)
Delivered as normal disaboard with one empty socket. All user-defined data to be stored in one 2764 which fits in socket. Two 2764s are supplied: one empty, another with example disassemblers (e.g. IEEE-488 bus).
Max. 4 disassemblers packs (EPROM boards) can be installed per instrument.

Personality adapters

To allow easy target clip-on connection to PM3565/70, compatible with PM8821, PM8825 and PM8826 pods. Two hard-wired types are available: PM8817/40: 5 pods to 40-pin DIL clip
PM8817/64: 8 pods to 64-pin DIL clip.
Prewired personality adapters are available for many microprocessor types. All pluggable parts of PM8817/XX can be ordered separately.
PM8816/XX personality housing for PM8817/XX, same range as PM8817/XX
PM8815/40: 40-pin DIL-clip with flat cable and connector (to PM8816/XX)

Microprocessor support

Type	Disassembler package	Personality	Front-end
Intel 8031/51		PM8817/30	40-pin DIL
	PM8858/40 (X pack)	PM8817/40	40-pin DIL
Custom disassembler	PM8859/10 (C pack)	User selectable	User selectable

PM8815/45: 40-pin DIL low-profile (12 mm height) with flat cable and connector (to PM 8816/XX)
PM8815/48: 48-pin DIL-clip with flat cable and connector (to PM8816/XX)
PM8815/64: 64-pin DIL-clip with flat cable and connector (to PM8816/XX)
PM8815/65: 64-pin DIL-clip low profile with flat cable and connector (to PM 8816/XX)
PM8815/68: 68-pin grid array front-end with flat cable and connector (to PM 8816/XX)
PM 8815/70: 68-pin grid array front-end with flat cable and connector (to PM 8816/XX)
(Intel version)
PM8815/00: live pod noses allowing connection of PM 8821/25/26 pod to PM 8816/XX
(personality housing)

6.3 The PM3632 logic analyser

Features of the PM3632 logic analyser include:

- Up to 32 channels or 100 MHz
- Disassembly
- Selective data acquisition
- Glitch catching
- ROM emulation
- Auto sequencing
- Easy to use

The PM3632 has acquisition speeds of up to 100 MHz and data widths up to 32 channels, the PM3632 personal Logic Analyzer encompasses all state/time applications in digital systems. A full range of problems in the state/time domain can thus be examined and solved.

A wide selection of pods and disassemblers permits use with virtually any of the popular microprocessors and its powerful ROM emulator option extends this versatility even further.

With the addition of plug-in Setup and Data Memory options, each user, student, field engineer or production tester can perform his task more simply, efficiently and quickly. The exceptionally high performance-to-price ratio puts the PM3632 within reach of those working in a Hi-tech field on a limited budget

The instrument offers 32 channels that can be used for state analysis at speeds of up to 12,5 MHz, suitable to handle the fastest microprocessors on the market. Switching between state analysis (external clock) and timing analysis (internal clock) is achieved simply by pressing a button.

As 12,5 MHz is generally too slow for timing analysis, the user can select a different operating mode which increases the acquisition speed to 25 MHz, 50

MHz and 100 MHz respectively. As the number of channels required for timing analysis is generally less and memory depth requirements increase with acquisition speeds, channel width is decreased and memory depth is increased up to 8 kbits per channel. The optional PM8862 allows glitches of 5 ns to be captured.

4 operating modes are available (see Table 3 below), covering applications from state analysis on 8- and 16-bit CPUs up to timing analysis with 10 ns resolution all in the same instrument.

Table 3

Channels	Speed	Memory depth
32	12,5	1K bits/channel
16	25	2K bits/channel
8	50	4K bits/channel
4	100	8K bits/channel

The capability of the instrument is represented also in its trigger capabilities. Four-level sequential, offering 19 predefined trigger conditions to cover most applications, as well as a user-defined trigger sequence, using 4 trigger words and 4 trigger levels.

The already adequate memory depth of 1K byte per channel at 32 channels width, can be virtually extended by means of powerful selective data acquisition. As well as these standard features, a wide range of options is also available for this instrument. This allows it to be tailored to the users specific application. These options include complete 8-bit microprocessor disassembly support, RS 232C serial interfacing, set-up and data non-volatile memory offering 8 data and set-up stores as well as compare and search functions, and a unique ROM-emulation offering interactive software debugging.

13. Application notes

CONTENTS – APPLICATION NOTES

		page
APPENDIX 1		13-5
1.0	SOLDERING TECHNIQUES	13-5
2.0	CONVENTIONAL CHIP PACKAGES	13-5
3.0	SURFACE MOUNT PACKAGES	13-5
3.1	Soldering – the reflow solder technique	13-5
3.1.1	Soldering	13-6
3.1.2	SO packages	13-7
3.1.3	VSO packages	13-7
3.1.4	PLCC packages	13-7
APPENDIX 2		13-8
1.0	MICROCONTROLLER DECOUPLING	13-8
2.0	ROM CODE SUBMISSION	13-9
APPENDIX 3		13-10
1.0	RECOMMENDATIONS FOR HANDLING CMOS DEVICES	13-10
1.1	Electrostatic charges	13-10
1.2	Work station	13-10
1.3	Receipt and storage	13-11
1.4	Assembly	13-11

APPENDIX 1

1.0 SOLDERING TECHNIQUES

In circuit construction, manufacturers today use a mixture of surface mount (SMD) and through-hole components. As surface mount technology gains in favour as forecast, and more and more ICs become available as SMD types, it becomes important to provide basic information upon SMD mounting together with conventional soldering techniques.

Packages for the 8-bit single chip microcontroller include amongst others:-

- 40-lead DIL plastic (SOT-121)
- 68-lead plastic leaded chip carriers (PLCC) (SOT-188A)*
- 40-lead flat pack; plastic (VSO-40; SOT-158)*
- 28-lead DIL; plastic (SOT-117D)
- 28-lead mini-pack; plastic (SO-28; SOT-136A)*
- 28-lead DIL; ceramic (CERDIP) (SOT-135A)
- 20-lead DIL; plastic (SOT-146)

* For SMD packages

2.0 CONVENTIONAL CHIP PACKAGES

For conventional chip packages e.g 28-lead DIL; plastic (SOT- 117D), rudimentary soldering tips are as follows.

1) By hand

Apply the soldering iron below the seating plane (or not more than 2 mm above). If its temperature is below 300 ^{o}C, it must not be in contact for more than 10 s; if between 300 ^{o}C and 400 ^{o}C, for not more than 5 s.

2) By dip or wave

The maximum permissable temperature of the solder is 260 ^{o}C; this temperature must not be in contact with the joint for more than 5 s. The total contact time of successive solder must again not exceed 5 s.

The device may be mounted up to the seating plane, but the temperature of the plastic body must not exceed the specific storage maximum. If the printed circuit board has been pre-heated, forced cooling may be necessary immediatly after soldering to keep the temperature within the permissible limit.

3) Repairing soldered joints

The same precautions and limits apply as in 1) above.

3.0 SURFACE MOUNT PACKAGES

3.1 Soldering -the reflow solder technique

The preferred technique for mounting miniature components on hybrid thick or thin-film circuits is reflow soldering. Solder is applied to the required areas on the substrate by dipping in a solder bath or, more usually, by screen printing a solder paste. Components are put in place and the solder is reflowed by heating.

Solder paste consists of very finely powdered solder and flux suspended in an organic liquid binder. They are available in various forms depending on the specification of the solder and the type of binder used. For hybrid circuit use, a tin-lead solder with 2 to 4% silver is recommended. The working temperature of this paste is about 220 ^{o}C to 230 ^{o}C when a mild flux is used.

For printing the paste onto the substrate a stainless steel screen with a mesh of 80 to 105 um is used for which the emulsion thickness should be about 50 um. To ensure that sufficient solder paste is applied to the substrate, the screen aperture should be slightly larger than the corresponding contact area.

The contact pins are positioned on the substrate, the slight adhesive force of the solder paste being sufficient to hold them in place. The substrate is heated up to the working solder temperature by means of a controlled hot plate. The soldering process should be kept as short as possible: 10 to 15 s is sufficient to ensure good solder joints and evaporation of the binder fluid.

After soldering, the substrate must be cleansed of any remaining flux.

3.1.1 Soldering

1) Soldering iron or pulse heated solder tool

Apply the heating tool to the flat part of the pin only. Limit the contact time to a maximum of 10 s up to 300 ^{o}C, or 5 s up to a maximum of 400 ^{o}C. When using the correct tools, all pins may be soldered in one operation within 2 to 5 s between a temperature of 270 ^{o}C to 320 ^{o}C.

2) By dip or wave

The maximum permissable temperature of the solder is 260 ^{o}C. The total permissable time of immersing the whole package in the bath is 10 s, if it is allowed to cool down to less than 150 ^{o}C within 6 s.

3) Repairing soldered joints

The same precautions and limits apply as in 1) above. If the vertical part of the pin needs heating, reduce the soldering iron temperature to 260 ^{o}C.

4) Joint assessment

Although the criteria for the assessment of an SMD joint are basically the same as those for through-hole components; good wetting, the right amount of solder, and a sound, smooth surface, an additional factor to be considered is the mis-alignment of the component on the solder lands.

Surface mount packages for the 8-bit microcontroller range are presently manufactured in three distinct sizes;

- (PLCC) Plastic leaded chip carrier
- (SO) Small Outline package
- (VSO) Very Small Outline package.

3.1.2 SO Packages

When placing surface mounted packages such as SO ICs, a projection of the lead over the pad of more than half the width of the lead constitutes a major defect. A projection smaller than half the width of the lead is a minor defect. Shifting the lead lengthways is not a problem, as long as the whole foot of the lead is on the solder pad.

For the best joint, the space between the heel and the solder land should be filled with solder with a meniscus height equal to the thickness of the lead. The solder fillets on the sides of the foot should also be this height. In the case of projection of the leads beyond the solder pads, this requirement is only valid for the side of the lead situated on the solder land. A meniscus height of less than half the lead is unacceptable.

3.1.3 VSO Packages

The criteria for assessing a good joint on (VSO) packages, are to an extent the same as for the SO with regard to the miniscus height and degree of wetting. However, at minimum joint, the foot of the lead should be secured over at least three quarters of its length and to a height equal to half the thickness of the lead.

3.1.4 PLCC Packages

For ICs in plastic leaded chip carriers (PLCCs), part of the lead and its associated solder fillet will be hidden from view beneath the component. It is necessary therefore, to assess the quality of the joint from the quantity of solder and the appearance of the fillet between the outside bend of the lead and the solder land.

With the ideal joint, the sides of the lead should be wetted and the area between the outside bend and the solder land should be filled with solder to a height equal to half the thickness of the lead. A meniscus extending to a height equal to half the thickness of the lead is the acceptable minimum. In both cases the solder must wet the entire solder land.

APPENDIX 2

1.0 MICROCONTROLLER DECOUPLING

Decoupling and surge voltage protection

Surge voltages in power lines are often caused by the switching of inductive loads, and direct or indirect lightning strikes. Other sources of spurious spikes may arise from resonance by thyristor equipment and power system faults. An ideal protection scheme for microcontroller systems consists of a gas discharge tube connected in parallel across the supply with either a silicon avalanche diode or a metal oxide varistor. With this arrangement all transient energy is diverted through the glass tube to ground. When the current level drops to a few mA, the gas tube returns to a high impedance state ready to protect against another surge (see Fig. 1 below).

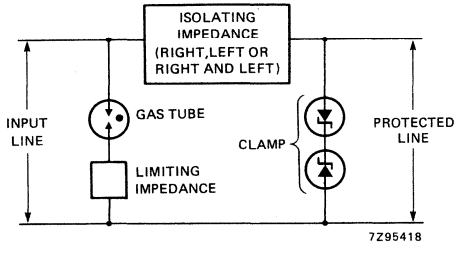


Fig. 1. A gas tube with series limiter, isolating impedance and clamp are the components necessary for surge protection. Gas discharge tubes with a wide range of breakdown voltages are available from manufacturers.

With the MAB84X1, it is advisable to decouple the power supply at each package. Using a ceramic disc capacitor of approximately 100 nF between V_{CC} and ground prevents damage to ICs from voltage spikes. The capacitor should be placed as close to the V_{CC} pin as possible, with the connections as short as possible.

2.0 ROM CODE SUBMISSION

Object code programs for insertion into ROM, prior to fabrication, should be submitted using either of the following machine readable media.

- 8,0 inch floppy disc;
  Single sided, single or double density, written on an int$_e$l MDS 2 system. (format INTELLEC .HEX)

- 5,25 inch floppy disc;
  Double sided, double density, written on a PHILIPS PMDS (1,2 or 3) system in int$_e$l format.

Suitable EPROMS include; 2716, 2732 (A), 2764, 8748, 8749, 8751.

Note: When submitting code to the manufacturers, use the official 'order entry' forms, examples of these are shown in section 15.

APPENDIX 3

1.0 RECOMMENDATIONS FOR HANDLING CMOS DEVICES

1.1 Electrostatic charges

Electrostatic charges are found in many things; commonplace examples are persons wearing man-made fibre clothing, moving machinery, objects with air blowing across them, plastic storage bins, sheets of paper stored in plastic envelopes, paper from electrostatic copying machines, and human movement. The charge is caused by friction to movement between two surfaces, at least one of which is non-conductive. The magnitude and polarity of electrostatic charge depends on the different affinities for electrons of the two materials rubbing together, the friction force and the humidity of the air.

Electrostatic discharge is a transfer of electrostatic charges between bodies at different potentials and occurs with direct contact or when induced by an electrostatic field. It is the electrostatic discharge that causes damage to sensitive electronic components. All of our CMOS integrated circuits are internally protected against electrostatic discharge, but they can be damaged if proper precautions are not taken.

The following paragraphs give an outline of the precautions that can be taken to avoid damage by electrostatic discharge to sensitive devices.

1.2 Work station

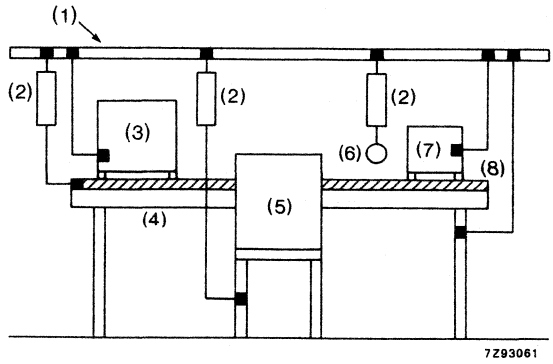
The work station (Fig. 1) is a working area constructed for the safe handling of electrostatic sensitive devices. It has a work bench with a conductive surface or with the surface covered by an antistatic sheet. A typical resistance value for the bench surface is 1 kΩ to 0.5 MΩ per cm^2. The floor covering should also be of antistatic material.

Persons working at the bench should be connected to ground potential via a wrist strap and a resistor.

All electrical equipment should be connected to the mains supply via an earth-leakage switch and the casings connected directly to ground.

Relative humidity should be kept within the range 50 to 65%.

An ionizer should be employed to neutralize objects with immobile static charges.



(1) Grounding rail
(2) Resistor (1 MΩ ± 10%, 0.5 W)
(3) Ionizer
(4) Work bench
(5) Chair
(6) Wrist strap
(7) Electrical equipment
(8) Conductive surface/ antistatic sheet

Fig. 1 Protected work station for handling electrostatic sensitive devices.

1.3 Receipt and storage

Our electrostatic sensitive devices are packed for despatch in antistatic/conductive boxes, rails or blister tape. The fact that the devices are sensitive to electrostatic discharge is shown by warning labels on both primary and secondary packing.

Sensitive devices should be kept in their original packing whilst in storage. If a bulk container is to be partially unpacked, this work should be performed at the protected work station. The removed devices and devices requiring temporary storage should be packed in conductive or antistatic packing or carriers.

1.4 Assembly

Production/assembly documents should state that the product contains electrostatic sensitive devices and that special precautions need to be taken.

During assembly, ensure that the electrostatic sensitive devices are the last of the components to be mounted and that this is done at the protected work station (Fig. 1).

Devices are to be removed from their protective packing with a grounded component-pincer or short-circuit clip. Short-circuit clips are to be fitted to all leads of each device and remain there during mounting, soldering and cleansing/drying processes. Do not take more components from the storage packing than are needed at any one time.

All tools used during assembly, including soldering tools and solder baths, are to be grounded. All hand tools should be of conductive or antistatic material and, where possible, not be insulated.

Measuring and testing after assembly should be performed at the protected work station. Place the soldered side of the circuit board on conductive or antistatic foam and remove the short-circuit clips. Remove the circuit board from the foam, holding the board only at the edges. Carry out measuring and testing making sure that the circuit board does not touch the conductive surface of the work bench. After testing, replace the circuit board on the conductive foam to await packing.
Assembled printed circuit boards containing electrostatic sensitive devices are to be handled as for the sensitive devices themselves. They should carry suitable warning labels and be packed in conductive or antistatic packing.

14. Integrated circuits supporting the I^2C-Bus

INTEGRATED CIRCUITS SUPPORTING THE I²C-BUS

MICROCONTROLLERS

NMOS

	RAM	ROM	
MAB8411	64	1K	plus 8-bit LED driver
MAF8411	64	1K	plus 8-bit LED driver; extended temperature
MAF84A11	64	1K	plus 8-bit LED driver; automotive temperature; reduced frequency
MAB8421	64	2K	plus 8-bit LED driver
MAF8421	64	2K	plus 8-bit LED driver; extended temperature
MAF84A21	64	2K	plus 8-bit LED driver; automotive temperature; reduced frequency
MAB8422	64	2K	plus 8-bit LED driver
MAF8422	64	2K	plus 8-bit LED driver; extended temperature
MAF84A22	64	2K	plus 8-bit LED driver; automotive temperature; reduced frequency
MAB8401	128	-	bond-out version for MAB84X1 family plus 8-bit LED-driver
MAB8441	128	4K	plus 8-bit LED driver
MAF8441	128	4K	plus 8-bit LED driver; extended temperature
MAF84A41	128	4K	plus 8-bit LED driver; automotive temperature; reduced frequency
MAB8442	128	4K	plus 8-bit LED driver
MAF8442	128	4K	plus 8-bit LED driver; extended temperature
MAF84A42	128	4K	plus 8-bit LED driver; automotive temperature; reduced frequency
MAB8461	128	6K	plus 8-bit LED driver
MAF8461	128	6K	plus 8-bit LED driver; extended temperature
MAF84A61	128	6K	plus 8-bit LED driver; automotive temperature; reduced frequency

CMOS

	RAM	ROM	
PCF84C21	64	2K	plus 8-bit LED driver; extended temperature
PCF84C41	128	4K	plus 8-bit LED driver; extended temperature
PCF84C43	128	4K	plus LCD driver
PCB80C552	256	-	ROM-less version of PCB83C552
PCB80C652	256	-	ROM-less version of PCB83C652
PCF84C00	256	-	bond-out version PCF84CXX family
PCF84C64	256	6K	plus DOS; 4 x 6-bit DAC; 14-bit DAC and 3-bit ADC
PCB83C552	256	8K	PCB80C51 plus additional functions
PCB83C652	256	8K	PCB80C51 plus additional functions
PCF84C81	256	8K	plus 8-bit LED driver; extended temperature
PCF84C85	256	8K	32 I/O; extended temperature

GENERAL PURPOSE ICs

PCB8582	256 x 8-bit static CMOS EEPROM
PCD8572	128 x 8-bit static CMOS EEPROM
PCF8200	voice synthesizer
PCF8566	universal LCD driver for low multiplex rates (1:1 to 1:4); max. 96 segments
PCF8570	256 x 8-bit static RAM
PCF8571	128 x 8-bit static RAM
PCF8573	clock calendar with alarm control
PCF8574	remote 8-bit I/O expander/LED driver
PCF8574A	remote 8-bit I/O expander/LED driver; different slave address
PCF8576	universal LCD driver for low multiplex rates (1:1 to 1:4); max. 160 segments
PCF8577	LCD direct driver (32 segments) or duplex driver (64 segments)
PCF8577A	LCD direct driver (32 segments) or duplex driver (64 segments); different slave address
PCF8578	LCD row/column driver for dot matrix graphics displays

PCF8579	LCD column driver for dot matrix graphics displays
PCF8582	256 x 8-bit static CMOS EEPROM (extended temperature range)
PCF8583	Clock calendar with 256 x 8-bit static RAM
PCF8591	8-bit A/D and D/A converter
SAA1300	tuner switching circuit; or 5-bit high current output
SAA3028	infrared remote control transcoder (RC-5)
TDA8442	I^2C-bus interface for colour decoders; or quad 6-bit DAC with 3 outputs
TDA8444	octuple 6-bit general purpose DAC

TELEVISION CIRCUITS

SAA1064	2-digit static/4 digit dynamic LED driver
SAA3028	infrared remote control transcoder (RC-5)
SAA5240A	computer Controlled Teletext circuit (CCT) 625-line system (English, German, Swedish)
SAA5240B	computer Controlled Teletext circuit (CCT) 625-line system (Italian, German, French)
SAA9020	field memory controller
SAA9050	digital PAL/NTSC colour decoder
SAA9055	digital SECAM decoder
SAA9061	digital deflection processor ($1f_H$; $1f_V$)
SAA9062	digital deflection processor ($2f_H$; $1f_V$)
SAA9063	digital deflection processor ($2f_H$; $2f_V$)
SAA9068	picture-in-picture controller
SAB3035	computer Interface for Tuning and Control (CITAC); 8 DACs
SAB3036	computer Interface for Tuning and Control (CITAC); without DACs
SAB3037	computer Interface for Tuning and Control (CITAC); 4 DACs
TDA8400	computer interfaced prescaler-synthesizer
TDA8405	stereo/dual sound processor; West German TV system

TDA8420	loudspeaker and headphone audio output processor
TDA8421	loudspeaker and headphone audio output processor
TDA8425	loudspeaker audio output processor
TDA8432	analogue deflection processor and sync control
TDA8440	peri-tv plug switch for CTV receivers
TDA8442	I^2C bus interface for colour decoders; or quad 6-bit DAC with 3 outputs
TDA8443	YUV/RGB matrixing and switching for computer controlled television levels
TDA8443A	YUV/RGB matrixing and switching for standard decoder levels b
TDA8461	PAL/NTSC colour decoder and RGB processor

CAR RADIO/RADIO CIRCUITS

SAA1300	tuner switching circuit; or 5-bit high current output
SAA3028	infrared remote control transcoder (RC-5)
SAA7250	digital audio signal processor
TEA6000	FM/IF system and microcomputer-based tuning interface
TEA6100	FM/IF system and microcomputer based quad detect tuning interface
TEA6300	Sound Fader Control Circuit (SOFAC) pre-amplifier with source selector
TEA6310T	Sound Fader Control Circuit (SOFAC) with tone and volume control
TSA6057	radio tuning PLL frequency synthesizer

TELEPHONY CIRCUITS

PCD3311	DTMF/modem/musical tone generator
PCD3312	DTMF/modem/musical tone generator
PCD3341	pulse repertory dialler/telephone set controller
PCD3343	microcontroller

VCR CIRCUITS

SAA3028	infrared remote control transcoder (RC-5)
SAA4700;T	data line 16 decoder
SAB3036	Computer Interface for Tuning and Control (CITAC); without DACs
SAF1134P	data line 16 decoder
SAF1135P	data line 16 decoder
TDA8400	computer interfaced prescaler-synthesizer

COMPACT DISC CIRCUIT

SAA1136	PCM-audio ident-word interface (IDI)

15. Order entry forms

8041A - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON

CUSTOMER'S NAME AND ADDRESS

CUSTOMER'S PROJECT NAME AND APPLICATION

FOR INTERNAL USAGE

TYPE NO. ASSIGNED

ORDER QUANTITY :
REQUESTED DELIVERY SCHEDULE:
.....SAMPLES WK 2QTY WK
1. QTY WK 3QTY WK

PACKAGE TYPE ☐ P-DIL 40

MIN QTY ACC/REJ.

ACC/REJ.

☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS

☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS

1st LINE RESERVED
2nd LINE RESERVED
3rd LINE RESERVED
4th LINE FOR SPECIAL MARK *

* PLEASE ENTER SPECIAL MARKING IF REQUESTED

ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8041A	0 ... +70	125	1.0	6.0

WORK IN PROGRESS LIABILITY OPTION

WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 4,5 K - UP TO

INPUT MEDIA PROVIDED:
1.
2.

X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTELLEC PAPER TAPE

B PROGRAMMED 8741/8748/8749 8751/2716/2732

C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 000D TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)

YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2094/01

8048H - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON		
CUSTOMER'S NAME AND ADDRESS		
CUSTOMER'S PROJECT NAME AND APPLICATION		FOR INTERNAL USAGE TYPE NO. ASSIGNED
ORDER QUANTITY : REQUESTED DELIVERY SCHEDULE: .....SAMPLES WK 1. QTY WK 2QTY WK 3QTY WK	PACKAGE TYPE ☐ P-DIL 40 ☐ VSO 40	MIN QTY ACC/REJ. ACC/REJ.
☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS	☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS - FOR P-DIL ONLY 1st LINE RESERVED 2nd LINE RESERVED 3rd LINE RESERVED 4th LINE FOR SPECIAL MARK * * PLEASE ENTER SPECIAL MARKING IF REQUESTED	ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.	REMARKS
☐	MA8 8048H	0 ... +70	80	1.0	11.0	
☐	MAF 8048H	-40 ... +85	90	1.0	11.0	NOT VSO 40
☐	MAF 80A48H	-40 ...+110	90	1.0	10.0	NOT VSO 40

WORK IN PROGRESS LIABILITY OPTION	WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 5 K - UP TO	
INPUT MEDIA PROVIDED:	1. 2.	X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTELLEC PAPER TAPE

B PROGRAMMED 8741/8748/8749 8751/2716/2732

C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 0000 TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE) YES ☐	WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN: DATE: SIGNATURE:

HD 2095/01

8049H - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON		
CUSTOMER'S NAME AND ADDRESS		
CUSTOMER'S PROJECT NAME AND APPLICATION		FOR INTERNAL USAGE TYPE NO. ASSIGNED
ORDER QUANTITY : REQUESTED DELIVERY SCHEDULE: .....SAMPLES WK 1. QTY WK 2QTY WK 3QTY WK	PACKAGE TYPE ☐ P-DIL 40	MIN QTY ACC/REJ. ACC/REJ.
☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS	☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS 1st LINE RESERVED 2nd LINE RESERVED 3rd LINE RESERVED 4th LINE FOR SPECIAL MARK * * PLEASE ENTER SPECIAL MARKING IF REQUESTED	ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8049H	0 ... +70	90	1.0	11.0
☐	MAF 8049H	-40 ... +85	100	1.0	11.0
☐	MAF 80A49H	-40 ...+110	100	1.0	10.0

WORK IN PROGRESS LIABILITY OPTION	WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 4 K - UP TO	
INPUT MEDIA PROVIDED:	1. 2.	X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

- A INTELLEC PAPER TAPE
- B PROGRAMMED 8741/8748/8749 8751/2716/2732
- C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

\+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 0000 TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)

YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2096/01

80C49 - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON		
CUSTOMER'S NAME AND ADDRESS		
CUSTOMER'S PROJECT NAME AND APPLICATION		FOR INTERNAL USAGE TYPE NO. ASSIGNED
ORDER QUANTITY : REQUESTED DELIVERY SCHEDULE: .....SAMPLES WK 2QTY WK 1. QTY WK 3QTY WK	PACKAGE TYPE ☐ P-DIL 40	MIN QTY ACC/REJ. ACC/REJ.
☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS	☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS 1st LINE RESERVED 2nd LINE RESERVED 3rd LINE RESERVED 4th LINE FOR SPECIAL MARK * * PLEASE ENTER SPECIAL MARKING IF REQUESTED	ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	PCB 80C49	0 ... +70	15	1.0	11.0
☐	PCF 80C49	-40 ... +85	15	1.0	11.0

WORK IN PROGRESS LIABILITY OPTION	WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 1,5 K - UP TO	
INPUT MEDIA PROVIDED:	1. 2.	X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

- A INTELLEC PAPER TAPE
- B PROGRAMMED 8741/8748/8749 8751/2716/2732
- C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

\+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 000D TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)
YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2100/01

8421 - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON

CUSTOMER'S NAME AND ADDRESS

CUSTOMER'S PROJECT NAME AND APPLICATION

FOR INTERNAL USAGE

TYPE NO. ASSIGNED

ORDER QUANTITY :
REQUESTED DELIVERY SCHEDULE:
.....SAMPLES WK 2QTY WK
1. QTY WK 3QTY WK

PACKAGE TYPE ☐ P-DIL 28

MIN QTY ACC/REJ.

ACC/REJ.

☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS

☐ STANDARD - MARKING + SPECIAL MARK UP TO 10 DIGITS

1st LINE RESERVED
2nd LINE RESERVED
3rd LINE RESERVED
4th LINE | X | X | X | X | X | X | X | X | | | | | | | | | |

RESERVED SPECIAL MARKING

ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8421	0 ... +70	85	1,0	6,0
☐	MAF 8421	-40 ... +85	100	1,0	6,0
☐	MAF 84A21	-40 ...+110	100	1,0	5,12

I/O MASK OPTIONS: SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS 0 THROUGH 2 AND FOR THE T1 INPUT BY MARKING ONLY 1 BOX FOR EACH PIN:

PIN	OPTION 1	2	3	PIN	OPTION 1	2	3	PIN	OPTION 1	2	3
P00 4				P07 11				P16 24			
P01 5				P10 18				P17 25			
P02 6				P11 19				P20 26			
P03 7				P12 20				P21 27			
P04 8				P13 21				P22 1			
P05 9				P14 22				P23 2			
P06 10				P15 23				T1 13			

PARTS:
OPTION 1: OPEN DRAIN I/O WITHOUT PULL UP
OPTION 2: OPEN DRAIN I/O WITH PULL UP
OPTION 3: PUSH PULL I/O WITH PULL UP

T1: OPTION 1: DELETES THE PULLUP RESISTOR FOR USE AS A ZERO CROSS DETECTION INPUT

OPTION 2: INCLUDES THE PULLUP RESISTOR FOR USE WITH AN EXTERNAL SWITCH OR STANDARD TTL

WORK IN PROGRESS LIABILITY OPTION

WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 4,5 K - UP TO

INPUT MEDIA PROVIDED:

1.
2.

X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTELLEC PAPER TAPE

B PROGRAMMED 8741/8748/8749 8751/2716/2732

C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 0000 TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT

YES ☐ TO BE TESTED FROM .....TO .....

NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)

YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2092/01

8441 - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON

CUSTOMER'S NAME AND ADDRESS

CUSTOMER'S PROJECT NAME AND APPLICATION

FOR INTERNAL USAGE

TYPE NO. ASSIGNED

ORDER QUANTITY :
REQUESTED DELIVERY SCHEDULE:
.....SAMPLES WK
1. QTY . WK
2QTY WK
3QTY WK

PACKAGE TYPE ☐ P-DIL 28

MIN QTY ACC/REJ.

ACC/REJ.

☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS

☐ STANDARD - MARKING + SPECIAL MARK UP TO 10 DIGITS

1st LINE RESERVED
2nd LINE RESERVED
3rd LINE RESERVED
4th LINE | X | X | X | X | X | X | X | X | | | | | | | | | | |

RESERVED SPECIAL MARKING

ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8441	0 ... +70	85	1,0	6,0
☐	MAF 8441	-40 ... +85	100	1,0	6,0
☐	MAF 84A41	-40 ...+110	100	1,0	5,12

I/O MASK OPTIONS: SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS 0 THROUGH 2 AND FOR THE T1 INPUT BY MARKING ONLY 1 BOX FOR EACH PIN:

PIN	OPTION 1	2	3	PIN	OPTION 1	2	3	PIN	OPTION 1	2	3
P00 4				P07 11				P16 24			
P01 5				P10 18				P17 25			
P02 6				P11 19				P20 26			
P03 7				P12 20				P21 27			
P04 8				P13 21				P22 1			
P05 9				P14 22				P23 2			
P06 10				P15 23				T1 13			

PARTS:
OPTION 1: OPEN DRAIN I/O WITHOUT PULL UP
OPTION 2: OPEN DRAIN I/O WITH PULL UP
OPTION 3: PUSH PULL I/O WITH PULL UP

T1: OPTION 1: DELETES THE PULLUP RESISTOR FOR USE AS A ZERO CROSS DETECTION INPUT

OPTION 2: INCLUDES THE PULLUP RESISTOR FOR USE WITH AN EXTERNAL SWITCH OR STANDARD TTL

WORK IN PROGRESS LIABILITY OPTION

WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 3 K - UP TO

INPUT MEDIA PROVIDED:
1.
2.

X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTELLEC PAPER TAPE

B PROGRAMMED 8741/8748/8749 8751/2716/2732

C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 000D TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)

YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2093/01

PCF 84C12 ORDER ENTRY FORM FOR NEW CODES

MARKETING DEPT.: FASELEC AG

RAEFFELSTRASSE 29, 8045 ZÜRICH

ALL INITIAL 84CXX-ORDERS MUST BE SUBMITTED ON THIS FORM (4-87)

PCF84C12	ROM CODE
PCF84C42	
	DATE:

SALES ORGANIZATION (MSO)	
CONTACT PERSON IN MSO	
CUSTOMER'S NAME AND ADDRESS	
CUSTOMER CONTACT PERSON	
CUSTOMER PROJECT NAME AND APPLICATION	

REQUESTED SAMPLE QUANTITY + PACKAGE

NO SAMPLES REQUIRED LESS THAN 100 PCS ☐

1. ☐ 5 CERDIL COSTS INCLUDED IN MASK CHARGE
2. ☐ ...CERDIL (LESS 30 PCS) ADDITIONAL TO MASK CHARGE 30 SFR/EACH
3. ☐ ...SO-20 (LESS 100 PCS) + 1 WK ML TIME
4. ☐ ...DIL-20 (LESS 100 PCS) + 4 WK ML TIME

ORDER QUANTITY:
REQUESTED DELIVERY SCHEDULE FOR PRODUCTION

1.QTY..... WK..... 2.QTY..... WK.....

START FIRST PRODUCTION (MORE THAN 100 PCS) WITHOUT SAMPLE VERIFICATION

☐ YES ☐ NO

PACKAGE TYPE FOR PRODUCTION ☐ PLASTIK T-PACK ☐ PLASTIC P-PACK

SPECIAL MARKING

☐ YES

☐ NO

SO-20 (10 CHARACTERS)

DIL-20 (17 CHARACTERS)

PLEASE NOTE:

REQUESTED MARKING WILL SUBSTITUTE TYPNUMBER

MARKING SHOULD INCLUDE PHILIPS LOGO ☐

I/O MASK OPTIONS SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS

PORT	PIN	OPTION	PORT	PIN	OPTION	PORT	PIN	OPTION
P00	1		P04	5		P10	15	
P01	2		P05	6		P11	16	
P02	3		P06	7		P12	17	
P03	4		P07	8		P13	18	
						P14	19	

PORT: OPTION 1: STANDARD I/O
OPTION 2: OPEN DRAIN I/O
OPTION 3: PUSH-PULL OUTPUT

I REALIZE THAT PORT OPTION NUMBERS OF 84CXX ARE DIFFERENT FROM 8400 NMOS OPTION NUMBERS. DATE: SIGNATURE:

ROM-CODE HAS TO BE SUBMITTED IN TWO EXAMPLES OF THE FOLLOWING MEDIA:
A) PROGRAMMED 2732, 2764
B) FILE TRANSMISSION VIA SERIENET ADDRESS: DNZ:FASZUS1

WE CONFIRM, THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN

DATE: SIGNATURE

REMARKS:

PCF 84C21/41/81 ORDER ENTRY FORM FOR NEW CODES

MARKETING DEPT.: FASELEC AG

RAEFFELSTRASSE 29, 8045 ZÜRICH

ALL INITIAL 84CXX-ORDERS MUST BE SUBMITTED ON THIS FORM (4-87)

PCF84C21	ROM CODE
PCF84C41	
PCF84C81	DATE:

SALES ORGANIZATION (MSO)	
CONTACT PERSON IN MSO	
CUSTOMER'S NAME AND ADDRESS	
CUSTOMER CONTACT PERSON	
CUSTOMER PROJECT NAME AND APPLICATION	

REQUESTED SAMPLE QUANTITY + PACKAGE

NO SAMPLES REQUIRED LESS THAN 100 PCS ☐

1. ☐ 5 CERDIL COSTS INCLUDED IN MASK CHARGE
2. ☐ ...CERDIL (LESS 30 PCS) ADDITIONAL TO MASK CHARGE 30 SFR/EACH
3. ☐ ...SO-28 (LESS 100 PCS) + 1 WK ML TIME
4. ☐ ...DIL-28 (LESS 100 PCS) + 4 WK ML TIME

ORDER QUANTITY:
REQUESTED DELIVERY SCHEDULE:

1.QTY..... WK..... 2.QTY..... WK.....

START FIRST PRODUCTION (MORE THAN 100 PCS) WITHOUT SAMPLE VERIFICATION

☐ YES ☐ NO

PACKAGE TYPE FOR PRODUCTION ☐ PLASTIK T-PACK ☐ PLASTIC P-PACK

SPECIAL MARKING

☐ YES

☐ NO

SO-28 (14 CHARACTERS)

DIL-28 (19 CHARACTERS)

PLEASE NOTE:

REQUESTED MARKING WILL SUBSTITUTE TYPNUMBER

MARKING SHOULD INCLUDE PHILIPS LOGO ☐

I/O MASK OPTIONS SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS

PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION
P00 4		P04 8		P10 18		P14 22		P20 26	
P01 5		P05 9		P11 19		P15 23		P21 27	
P02 6		P06 10		P12 20		P16 24		P22 1	
P03 7		P07 11		P13 21		P17 25			

NOTE: P23 IS ALWAYS OPEN DRAIN

PORTS: OPTION 1: STANDARD I/O
OPTION 2: OPEN DRAIN I/O
OPTION 3: PUSH-PULL OUTPUT

I REALIZE THAT PORT OPTION NUMBERS OF 84CXX ARE DIFFERENT FROM 8400 NMOS OPTION NUMBERS: DATE: SIGNATURE:

ROM-CODE HAS TO BE SUBMITTED IN TWO EXAMPLES OF THE FOLLOWING MEDIA:
A) PROGRAMMED 2732, 2764
B) FILE TRANSMISSION VIA SERIENET ADDRESS: DNZ:FASZUS1

WE CONFIRM, THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN

DATE: SIGNATURE:

REMARKS:

8051AH - STANDARD ORDER ENTRY FORM FOR NEW CODES

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ALL MICRO CONTROLLER - ORDERS MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON	
CUSTOMER'S NAME AND ADDRESS	
CUSTOMER'S PROJECT NAME AND APPLICATION	FOR INTERNAL USAGE TYPE NO. ASSIGNED

ORDER QUANTITY :
REQUESTED DELIVERY SCHEDULE:
.....SAMPLES WK 2QTY WK
1. QTY WK 3QTY WK

PACKAGE TYPE ☐ P-DIL 40

MIN QTY ACC/REJ.

ACC/REJ.

☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS

☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS

1st LINE RESERVED
2nd LINE RESERVED
3rd LINE RESERVED
4th LINE FOR SPECIAL MARK *

* PLEASE ENTER SPECIAL MARKING IF REQUESTED

ACC/ REJ.

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8051AH	0 ... +70	160	3,5	12.0
☐	MAF 8051AH	-40 ... +85	175	3,5	12.0
☐	MAF 80A51AH	-40 ...+100	175	3,5	10.0

WORK IN PROGRESS LIABILITY OPTION

WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 1,5 K - UP TO

INPUT MEDIA PROVIDED:
1.
2.

X-CHECK ACC/REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTELLEC PAPER TAPE
B PROGRAMMED 8741/8748/8749 8751/2716/2732
C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 0000 TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT
YES ☐ TO BE TESTED FROM .....TO .....
NO ☐ FULLY TESTED

AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

BURN IN (IF POSSIBLE)
YES ☐

WE CONFIRM,THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2099/01

PCF 84C43H ORDER ENTRY FORM FOR NEW CODES

MARKETING DEPT.: FASELEC AG

RAEFFELSTRASSE 29, 8045 ZÜRICH

ALL INITIAL 84CXX-ORDERS MUST BE SUBMITTED ON THIS FORM (7-87)

PCF84C43H

ROM CODE

DATE:

SALES ORGANIZATION (MSO)	
CONTACT PERSON IN MSO	
CUSTOMER'S NAME AND ADDRESS	
CUSTOMER CONTACT PERSON	
CUSTOMER PROJECT NAME AND APPLICATION	
REQUESTED SAMPLE QUANTITY + PACKAGE	1. ☐ ...QFP64 (LESS 100 PCS) + 1 WK ML TIME 5 SAMPLES INCLUDED IN MASK CHARGE NO SAMPLES REQUIRED LESS THAN 100 PCS ☐

ORDER QUANTITY:
REQUESTED DELIVERY SCHEDULE:

1.QTY..... WK..... 2.QTY..... WK.....

START FIRST PRODUCTION (MORE THAN 100 PCS) WITHOUT SAMPLE VERIFICATION

☐ YES ☐ NO

SPECIAL MARKING

☐ YES

☐ NO

☐☐☐☐☐☐☐☐☐☐ (10 CHARACTERS)

MARKING SHOULD INCLUDE PHILIPS LOGO ☐

PLEASE NOTE:

REQUESTED MARKING WILL SUBSTITUTE TYPENUMBER

I/O MASK OPTIONS SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS

PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION
P00 23		P10 32		DP30 11	
P01 24		P11 33		DP31 12	
P02 25		P12 34		DP32 13	
P03 27		P13 35		DP33 14	
P04 28		P14 36		DP34 17	
P05 29		P15 37		DP35 18	
P06 30		P16 38		DP36 19	
P07 31		P17 39		DP37 20	

NOTE: P23 IS ALWAYS OPEN DRAIN

PORTS: OPTION 1: STANDARD I/O
OPTION 2: OPEN DRAIN I/O
OPTION 3: PUSH-PULL OUTPUT

I REALIZE THAT PORT OPTION NUMBERS OF 84CXX ARE DIFFERENT FROM 8400 NMOS OPTION NUMBERS: DATE: SIGNATURE:

ROM-CODE HAS TO BE SUBMITTED IN TWO EXAMPLES OF THE FOLLOWING MEDIA:
A) PROGRAMMED 2764
B) FILE TRANSMISSION VIA SERIENET ADDRESS: DNZ:FASZUS1

WE CONFIRM, THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN

DATE: SIGNATURE:

REMARKS:

PCF 84C85 ORDER ENTRY FORM FOR NEW CODES

MARKETING DEPT.: FASELEC AG

RAEFFELSTRASSE 29, 8045 ZÜRICH

ALL INITIAL 84CXX-ORDERS MUST BE SUBMITTED ON THIS FORM (7-87)

PCF85C85

ROM CODE

DATE:

SALES ORGANIZATION (MSO)	
CONTACT PERSON IN MSO	
CUSTOMER'S NAME AND ADDRESS	
CUSTOMER CONTACT PERSON	
CUSTOMER PROJECT NAME AND APPLICATION	

REQUESTED SAMPLE QUANTITY + PACKAGE

NO SAMPLES REQUIRED LESS THAN 100 PCS ☐

1. ☐ 5 CERDIL COSTS INCLUDED IN MASK CHARGE

2. ☐ ...CERDIL (LESS 30 PCS) ADDITIONAL TO MASK CHARGE 30 SFR/EACH

3. ☐ ...SO-28 (LESS 100 PCS) + 1 WK ML TIME

4. ☐ ...DIL-28 (LESS 100 PCS) + 4 WK ML TIME

ORDER QUANTITY:
REQUESTED DELIVERY SCHEDULE:

1.QTY..... WK..... 2.QTY..... WK.....

START FIRST PRODUCTION (MORE THAN 100 PCS) WITHOUT SAMPLE VERIFICATION

☐ YES ☐ NO

PACKAGE TYPE FOR PRODUCTION ☐ PLASTIK T-PACK ☐ PLASTIC P-PACK

SPECIAL MARKING

☐ YES

☐ NO

SO-28 (14 CHARACTERS)

DIL-28 (19 CHARACTERS)

PLEASE NOTE:

REQUESTED MARKING WILL SUBSTITUTE TYPENUMBER

MARKING SHOULD INCLUDE PHILIPS LOGO ☐

I/O MASK OPTIONS SPECIFY THE DESIRED CONNECTION FOR EACH I/O LINE ON PORTS

PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION	PORT PIN	OPTION
P00 2		P07 9		P16 16		DP05 27		DP14 34	
P01 3		P10 10		P17 17		DP06 28		DP15 35	
P02 4		P11 11		DP00 22		DP07 29		DP16 36	
P03 5		P12 12		DP01 23		DP10 30			
P04 6		P13 13		DP02 24		DP11 31			
P05 7		P14 14		DP03 25		DP12 32			
P06 8		P15 15		DP04 26		DP13 33			

NOTE: P23 IS ALWAYS OPEN DRAIN

PORTS: OPTION 1: STANDARD I/O
OPTION 2: OPEN DRAIN I/O
OPTION 3: PUSH-PULL OUTPUT

I REALIZE THAT PORT OPTION NUMBERS OF 84CXX ARE DIFFERENT FROM 8400 NMOS OPTION NUMBERS: DATE: SIGNATURE:

ROM-CODE HAS TO BE SUBMITTED IN TWO EXAMPLES OF THE FOLLOWING MEDIA:
A) PROGRAMMED 2764
B) FILE TRANSMISSION VIA SERIENET ADDRESS: DNZ:FASZUS1

WE CONFIRM, THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN

DATE: SIGNATURE:

REMARKS:

8052AH - DEVELOPMENT SAMPLES ORDER ENTRY FORM

TO: MIC - LOGISTIC AND M.I.S.D.- DEPARTMENT, VALVO RHW DER PHILIPS GMBH
STRESEMANNALLEE 101, 2000 HAMBURG 54
ORDERS FOR DEVELOPMENT SAMPLES MUST BE SUBMITTED ON THIS FORM

MSO/ CONTACT PERSON

CUSTOMER'S NAME AND ADDRESS

CUSTOMER'S PROJECT NAME AND APPLICATION

FOR INTERNAL USAGE

TYPE NO. ASSIGNED

ORDER QUANTITY :
REQUESTED DELIVERY SCHEDULE:

.....SAMPLES WK ..... 2.QTY ..... WK .....

1.QTY ..... WK ..... 3.QTY ..... WK .....

PACKAGE TYPE ☐ P - DIL 40

MIN QTY ACC/ REJ.

ACC/ REJ.

☐ STANDARD - MARKING ACC.TO PHILIPS - MARK. INSTRUCTIONS

☐ STANDARD - MARKING + SPECIAL MARK UP TO 19 DIGITS (P - DIL 40 ONLY)

1st LINE RESERVED
2nd LINE RESERVED
3rd LINE RESERVED
4th LINE FOR SPECIAL MARK *

* PLEASE ENTER SPECIAL MARKING IF REQUESTED

ACC/ REJ.

☐ BURN IN (IF POSSIBLE)

YES

LIST OF STANDARD SPECIFICATIONS

	TYPE	TEMP.(°C)	MAX.SUPPLY CURRENT (mA)	FREQUENCY (MHZ) MIN.	FREQUENCY (MHZ) MAX.	REMARKS VCC MIN.	REMARKS VCC MAX.
☐	MAB 8052AH	0...70 °	175	3.5	12.0	4.5	5.5

WORK IN PROGRESS LIABILITY OPTION

WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. - UP TO

INPUT MEDIA PROVIDED:

1.
2.

X-CHECK ACC/ REJ.

ROM-CODE HAS TO BE SUBMITTED IN TWO OF THE FOLLOWING MEDIA FOR QUICKEST TURN/AROUND:

A INTEL COMPATIBLE EPROMS 27XX, 87XX, 80XX FAMILY

B 8" INTELLEC FORMATED FLOPPY DISK SD AND DD

C 5,25" PMDS I/II FORMATED FLOPPY DISK

D S E R I - NETWORK IF POSSIBLE

+ HARDCOPY (LISTING)

THE CUSTOMER AGREES, THAT ROM'S, THAT ARE ONLY PARTIALLY FILLED BY THE CUSTOMER, GET ONLY TESTED IN THESE PARTS. THAT IS, IF A CUSTOMER DELIVERS A CODE THAT FILLS FOR INSTANCE ONLY 1/4 K BYTES (ADDRESSES 000D TO 255D = 000H TO 0FFH), THEN VALVO IS ALLOWED TO TEST THE FILLED PART OF THE ROM.

AGREEMENT

YES ☐ TO BE TESTED FROM .....TO .....

NO ☐ FULLY TESTED

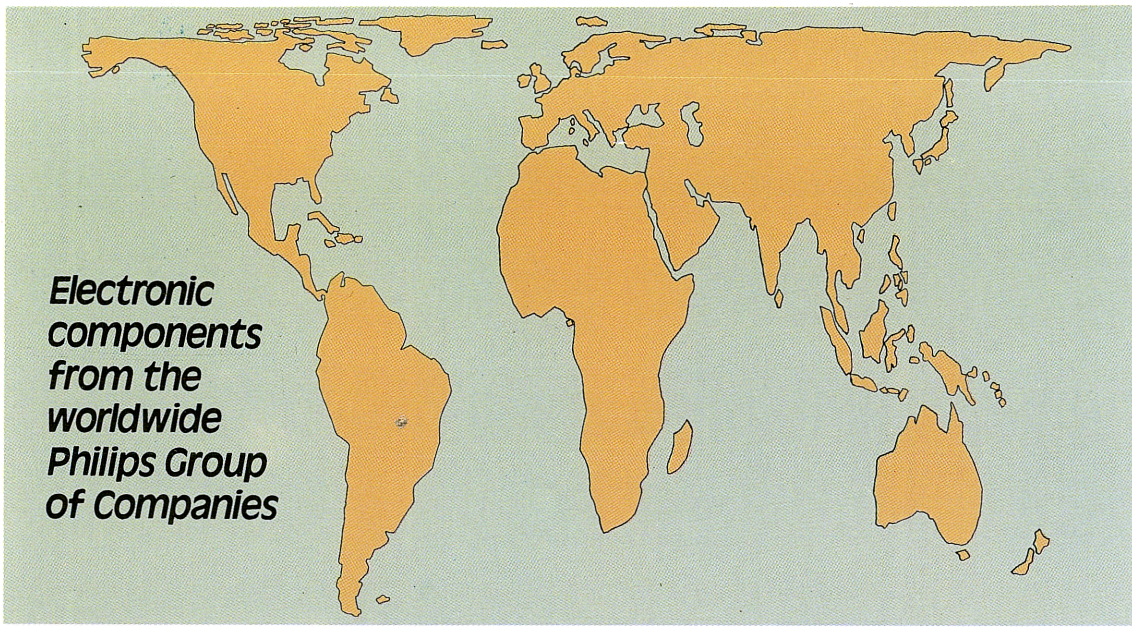
AFTER THE ROM-CODE IS VERIFIED, FACTORY WILL REPROGRAM AND PROVIDE MSO WITH THE CUSTOMER CODE TO CONFIRM CORRECT PROCESSING (SAME EPROM TYPE AS SUBMITTED BY THE CUSTOMER).

REMARKS

WE CONFIRM, THAT ORFO-ORDER WILL BE PLACED WITH QTIES MENTIONED ABOVE AS SOON AS TYPE NO. AND 12NC ARE KNOWN:

DATE: SIGNATURE:

HD 2116/1



Argentina: PHILIPS ARGENTINA S.A., Div. Elcoma, Vedia 3892, 1430 BUENOS AIRES, Tel. (01) 541 - 7141 to 7747.
Australia: PHILIPS INDUSTRIES LTD., Elcoma Division, 11 Waltham Street, ARTARMON, N.S.W. 2064, Tel. (02) 439 3322.
Austria: ÖSTERREICHISCHE PHILIPS INDUSTRIE G.m.b.H., UB Bauelemente, Triester Str. 64, 1101 WIEN, Tel. (0222) 60101-820.
Belgium: N.V. PHILIPS PROF. SYSTEMS - Elcoma Div., 80 Rue Des Deux Gares, B-1070 BRUXELLES, Tel. (02) 525 61 11.
Brazil: CONSTANTA-IBRAPE; (Active Devices): Av. Brigadeiro Faria Lima, 1735-SAO PAULO-SP, Tel. (011) 211-2600,
(Passive Devices & Materials): Av. Francisco Monteiro, 702 - RIBEIRAO PIRES-SP, Tel. (011) 459-8211.
Canada: PHILIPS ELECTRONICS LTD., Elcoma Division, 601 Milner Ave., SCARBOROUGH, Ontario, M1B 1M8, Tel. (416) 292-5161.
Chile: PHILIPS CHILENA S.A., Av. Santa Maria 0760, SANTIAGO, Tel. (02) 77 38 16.
Colombia: IND. PHILIPS DE COLOMBIA S.A., c/o IPRELENSO LTD., Cra. 21, No. 56-17, BOGOTA, D.E., Tel. (01) 2 49 76 24.
Denmark: MINIWATT A/S, Strandlodsvej 2, P.O. Box 1919, DK 2300 COPENHAGEN S, Tel. (01) 54 11 33.
Finland: OY PHILIPS AB, Elcoma Division, Kaivokatu 8, SF-00100 HELSINKI 10, Tel. (90) 1 72 71.
France: RTC-COMPELEC, 117 Quai du Président Roosevelt, 92134 ISSY-LES-MOULINEAUX Cedex, Tel. (01) 40 93 80 00.
Germany (Fed. Republic): VALVO, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, D-2 HAMBURG 1, Tel. (040) 3296-0.
Greece: PHILIPS HELLENIQUE S.A., Elcoma Division, No. 15, 25th March Street, GR 17778 TAVROS, Tel. (01) 48 94 339/48 94 911.
Hong Kong: PHILIPS HONG KONG LTD., Elcoma Div., 15/F Philips Ind. Bldg., 24-28 Kung Yip St., KWAI CHUNG, Tel. (0)-24 51 21.
India: PEICO ELECTRONICS & ELECTRICALS LTD., Elcoma Dept., Band Box Building, 254-D Dr. Annie Besant Rd., BOMBAY - 400 025, Tel. (022) 4930311/4930590.
Indonesia: P.T. PHILIPS-RALIN ELECTRONICS, Elcoma Div., Setiabudi II Building, 6th Fl., Jalan H.R. Rasuna Said (P.O. Box 223/KBY) Kuningan, JAKARTA 12910, Tel. (021) 51 79 95.
Ireland: PHILIPS ELECTRICAL (IRELAND) LTD., Elcoma Division, Newstead, Clonskeagh, DUBLIN 14, Tel. (01) 69 33 55.
Italy: PHILIPS S.p.A., Div. Componenti Elcoma, Piazza IV Novembre 3, I-20124 MILANO, Tel. (02) 6752.1.
Japan: NIHON PHILIPS CORP., Shuwa Shinagawa Bldg., 26-33 Takanawa 3-chome, Minato-ku, TOKYO (108), Tel. (03) 448-5611.
(IC Products) SIGNETICS JAPAN LTD., 8-7 Sanbancho Chiyoda-ku, TOKYO 102, Tel. (03) 230-1521.
Korea (Republic of): PHILIPS ELECTRONICS (KOREA) LTD., Elcoma Div., Philips House, 260-199 Itaewon-dong, Yongsan-ku, SEOUL, Tel. (02) 794-5011.
Malaysia: PHILIPS MALAYSIA SDN BHD, Elcoma Div., 345 Jalan Gelugor, 11700 PULAU PINANG, Tel. (04) 87 00 44.
Netherlands: PHILIPS NEDERLAND, Marktgroep Elonco, Postbus 90050, 5600 PB EINDHOVEN, Tel. (040) 78 37 49.
New Zealand: PHILIPS NEW ZEALAND LTD., Elcoma Division, 110 Mt. Eden Road, C.P.O. Box 1041, AUCKLAND, Tel. (09) 605-914.
Norway: NORSK A/S PHILIPS, Electronica Dept., Sandstuveien 70, OSLO 6, Tel. (02) 68 02 00.
Pakistan: PHILIPS ELECTRICAL CO. OF PAKISTAN LTD., Philips Markaz, M.A. Jinnah Rd., KARACHI-3, Tel. (021) 72 57 72.
Peru: CADESA, Av. Pardo y Aliaga No. 695, 6th Floor, San Isidro, LIMA 27, Tel. (014) 707080.
Philippines: PHILIPS INDUSTRIAL DEV. INC., 2246 Pasong Tamo, P.O. Box 911, Makati Comm. Centre, MAKATI-RIZAL 3116, Tel. (02) 86 89 51 to 59.
Portugal: PHILIPS PORTUGUESA S.A.R.L., Av. Eng. Duarte Pacheco 6, 1009 LISBOA Codex, Tel. (019) 68 31 21.
Singapore: PHILIPS PROJECT DEV. (Singapore) PTE LTD., Elcoma Div., Lorong 1, Toa Payoh, SINGAPORE 1231, Tel. 35 02 000.
South Africa: S.A. PHILIPS (Pty) LTD., EDAC Div., 3rd Floor Rainer House, Upper Railway Rd. & Ove St., New Doornfontein, JOHANNESBURG 2001, Tel. (011) 402-4600/07.
Spain: MINIWATT S.A., Balmes 22, 08007 BARCELONA, Tel. (03) 301 63 12.
Sweden: PHILIPS KOMPONENTER A.B., Lidingövägen 50, S-11584 STOCKHOLM, Tel. (08) 7821000.
Switzerland: PHILIPS A.G., Elcoma Dept., Allmendstrasse 140-142, CH-8027 ZÜRICH, Tel. (01) 488 22 11.
Taiwan: PHILIPS TAIWAN LTD., 150 Tun Hua North Road, P.O. Box 22978, TAIPEI, Taiwan, Tel. (02) 7120500.
Thailand: PHILIPS ELECTRICAL CO. OF THAILAND LTD., 283 Silom Road, P.O. Box 961, BANGKOK, Tel. (02) 233-6330-9.
Turkey: TÜRK PHILIPS TICARET A.S., Elcoma Department, Inönü Cad., No. 78-80, 80090 Ayazpasa ISTANBUL, Tel. (01) 143 59 10.
United Kingdom: MULLARD LTD., Mullard House, Torrington Place, LONDON WC1E 7HD, Tel. (01) 580 6633.
United States: (Active Devices & Materials) AMPEREX SALES CORP., Providence Pike, SLATERSVILLE, R.I. 02876, Tel. (401) 762-9000.
(Passive & Electromech. Dev.) MEPCO/CENTRALAB, INC., 2001 West Blue Heron Blvd, RIVIERA BEACH, Florida 33404, Tel. (305) 881-3200.
(IC Products) SIGNETICS CORPORATION, 811 East Arques Avenue, SUNNYVALE, CA 94088-3409, Tel. (408) 991-2000.
(Colour picture tubes - Monochrome & Colour display tubes) PHILIPS ECG INC., 50 Johnston St., SENECA FALLS, N.Y. 13148, Tel. (315)568-5881.
Urugay: LUZILECTRON S.A., Avda Uruguay 1287, P.O. Box 907, MONTEVIDEO, Tel. (02) 98 53 95.
Venuzuela: IND. VENEZOLANAS PHILIPS S.A., c/o MAGNETICA S.A., Calle 6, Ed. Las Tres Jotas, App. Post. 78117, CARACAS, Tel. (02) 239 39 31.

For all other countries apply to: Philips Electronic Components Division, International Business Relations, P.O. Box 218, 5600 MD EINDHOVEN, The Netherlands, Telex 35000 phtcnl

AB56

Printed in The Netherlands

9398 638 20011