SINGLE-CHIP 8-BIT MICROCONTROLLERS

USER MANUAL 1986

(continued on next page)

(continued on next page)

1. Introduction

CONTENTS – INTRODUCTION

AN INTRODUCTION TO MICROCONTROLLERS

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

1.0 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, coffee perculators, washing machines and televisions etc. It is in the realm of industrial and domestic control applications that the microcontroller/microprocessor's role has expanded. The data processing capability of these devices merits only a small production market when compared with their control applications.

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

Technical Parameters

- What will the system control?
- What are the data input and output requirements?
- What are the software requirements?
- What operating frequency the microcontroller will have?
- Which microcontroller is best for the application?

Cost

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

After these questions have been resolved and trade-offs weighed up, 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 the device, which combines versatility with simplicity, can be used by those with limited knowledge of electronics but who wish to design or improve existing control applications.

2.0 MICROPROCESSORS AND MICROCONTROLLERS

To avoid confusion between microcontrollers and microprocessors, the following definition may prove helpful, our microcontroller model is the MAB8400.

A microcontroller is a single LSI (Large Scale Integration) chip capable of implementing arithmetic and control functions. The microprocessor is only one of the functional units forming 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 communication with the outside-world, while data and program memory permit the storage of information and instructions. All units are connected together by buses, simply paths for signals having a common function.

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

3.0 THE BASIC MICROCONTROLLER SYSTEM

A typical microcontroller 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 operation are stored in Program Memory. Data, 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. Figure 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 exist, ready for use, before the microcontroller is switched on.

This type of memory is called Read Only Memory (ROM) - implying that stored information cannot be changed by write operations.

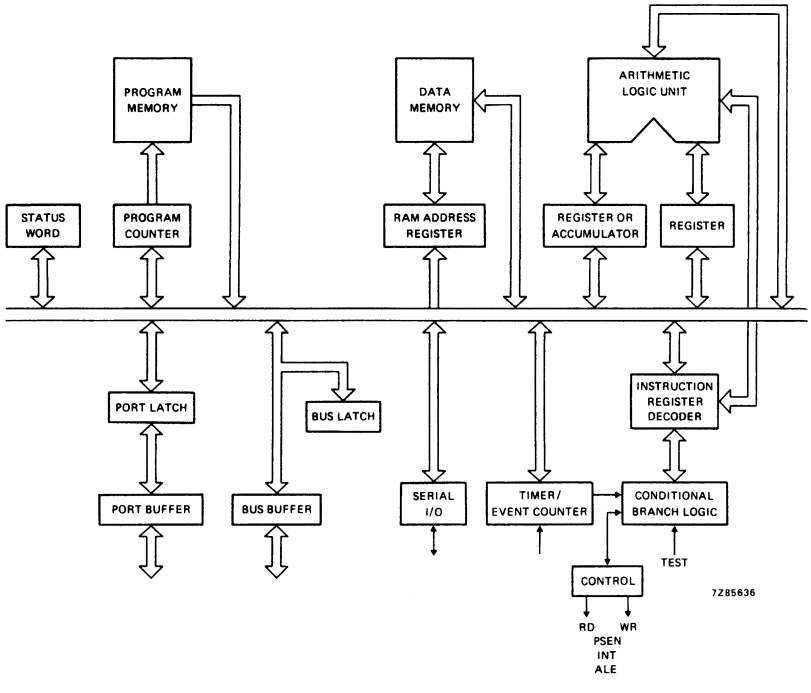


Fig. 1 Block diagram of a microcontroller.

There are two 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) that are selectively 'blown' to form a binary pattern

Erasable programmable ROM - EPROM eg Intel 8716.

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 memory 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 and control external events. Output Ports are used to transmit results to devices beyond the system, such as LEDs, relays and stepper motors.

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

6.0 CENTRAL PROCESSING UNIT (CPU)

The CPU is the nerve centre of any digital computer system, since it co-ordinates and controls the activities of the other units and performs all arithmetic and logic processes. 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 8-bit storage cells 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 (ALU), as its name implies, performs arithmetic and logic 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. However, in practice 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' to other program locations that 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 start processing operations.

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

Interrupts 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 serviceing peripheral equipment is by Polling. Polling involves testing the input ports or/and the interrupt request pin in rotation, for the presence of a signal. When a signal is detected the program will hand over control to a subroutine or interrupt service routine. This service routine will be dedicated to a particular device or operation. Polling is generally used in multi-device control applications.

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 serviced first.

7.0 REGISTERS

During arithmetric and/or logic operations, each register has specific properties,the following are the most important:-

7.1 Accumulator

The accumulator is a register in which arithmetic results are created, it contains the operand on which an operation takes place and in some cases 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 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 from several instructions. These memory locations can be indirectly addressed through the RAM Pointer Register within the register array. Since the working registers are more easily addressed, they are used to store frequently-accessed 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 memory location 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 break from sequence.

A jump instruction is a departure from normal program execution. The next instruction in the sequence is 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 is 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 unnecessary repetition of this set of instructions in the main program -a subroutine may be considered as a 'program within a program.'

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 of the Program Counter in order to resume execution of the program at the pointer immediately following the 'RET' 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 the Program Counter contents are stored in the stack, the Stack Pointer is incremented and points 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 from 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 returned 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, it contains the following information:

- 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. The addressed byte is then returned to the controller 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 interpreted as 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 when dealing with 8-bit binary words, 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 RAM and ROM exist which are compatible and able to replace the original chip used. Memory can also be expanded by adding more RAM or ROM external to the chip, up to a limiting value dependent on the microcontroller 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 output onto the address bus. ALE (Address latch enable) 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 data 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, sends the proper 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 the advantage of high speed, with the complete byte being transferred at the same time.

Special quasi-bidirectional input/output devices are available for handling parallel data, 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 reduces the number of connections required, leading to simpler circuit layouts and economic 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 data bus, the serial I/O detects, receives and converts the data stream to parallel without interrupting the execution of the current program. Some standard serial bus sytems commonly used include; I^2C, D^2B and 9-bit protocol.

10.0 COMPUTER TIMING

The synchronisation of functions in a computer is fundamental to it's correct operation. Instructions must be fetched and executed continously until the program is complete. A precise timing reference is therefore required. A free running clock provides just such a reference for all processing actions. This clock is generated by an on-chip oscillator with an external XTAL or LC combination providing the feedback and phase-shift required for operation.

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 MAB84XX instruction cycle is shown as consisting of 2 machine cycles each of 5 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. Memory is located by this address, and the instruction therein is placed in the instruction register via the data bus . The program counter is then incremented and will point to the next memory 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.

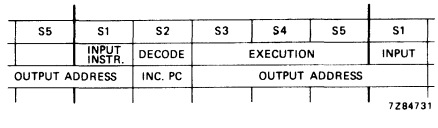


Fig. 2 An 8048 instruction cycle

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 controller. In both modes, counter operation is the same, the only difference is the source to the counter.

11.0 MICROCONTROLLER 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 controller understands. For most 8-bit microcontrollers, instructions are made up of eight binary 1's and 0's. 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 1's and 0's is to use the hexadecimal number base. Hexadecimal is a means of writing numbers to base 16, instead of, in the case of binary, base 2. This 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 slightly daunting to convert. But, when it is then split 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 write or understand. The use of Assembly Languages overcomes this disadvantage by allowing the programmer to use alphanumeric symbols to represent the 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 operations 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 the first byte is the instruction or opcode, and the second the data. In today's microcontroller the address is very often 16-bits wide, hence it is not uncommon to find instructions that are 3 bytes long.

The first instruction, MOV R1,#0, can be explained as: Clear register 1 using the value 0. MOV (Move)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 seperate software examples section of this manual.

Elsewhere in this manual is an expanded instruction set which explains the mnemonics and the 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 Language, the problem exists of how to translate it into machine code. The answer is to use ASSEMBLERS. Assemblers are computer programs that take typed instructions and data and convert them into binary codes.

Although programs may be correctly assembled, contain no syntax error and are 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 minicomputer, 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. With the benefits offered by such a system, access is 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 another section of this manual.

2. The MAB8048/80C49 microcontroller family

CONTENTS – THE MAB8048/80C49 MICROCONTROLLER FAMILY

(continued on next page)

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:

MAB8021 - single-chip microcontroller
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 5). 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 @fosc = 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

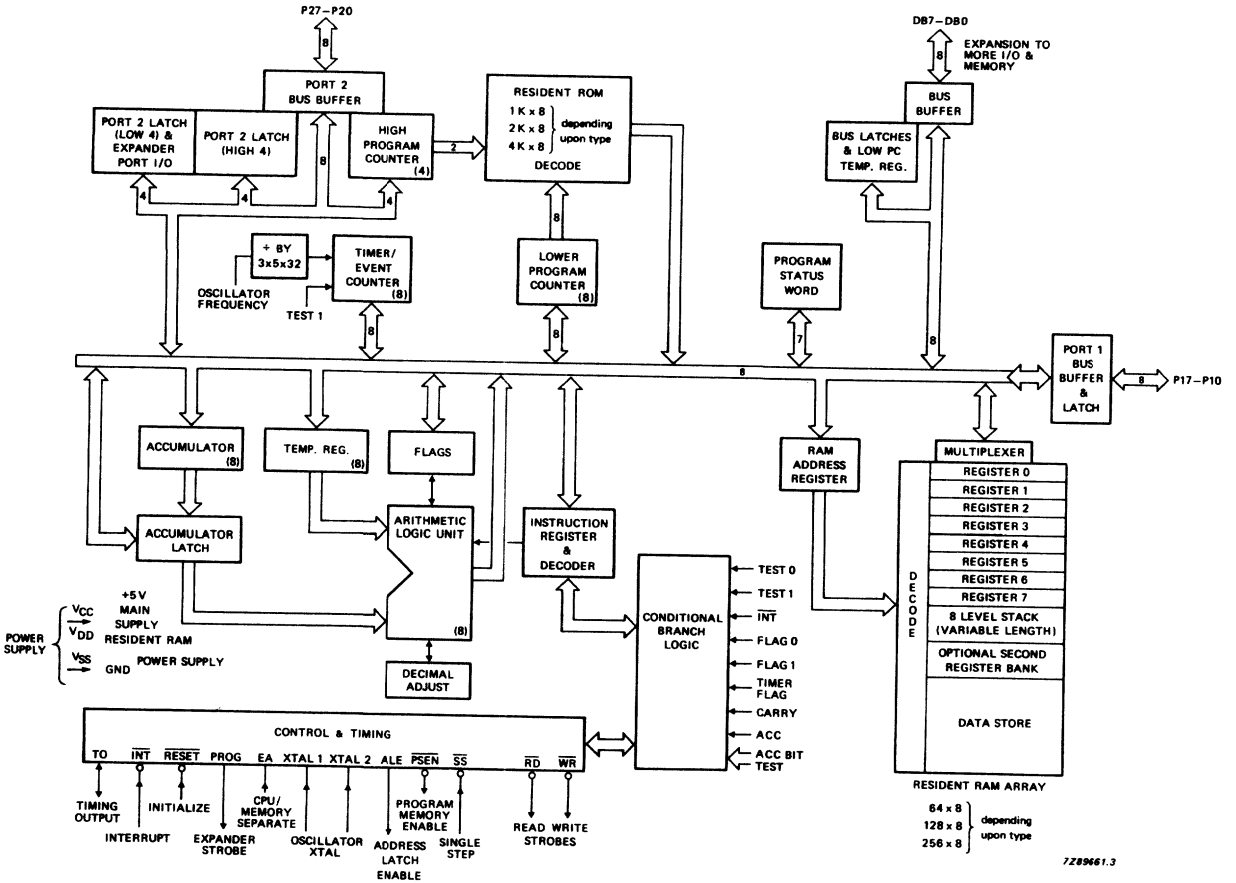


Fig. 2.1 Block diagram of the 8048/C49 family

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.

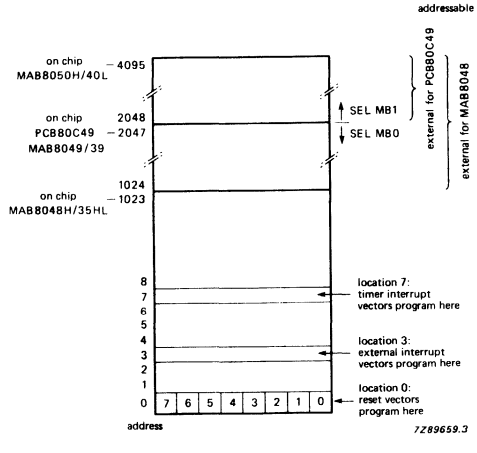


Fig. 3.1 Program memory map.

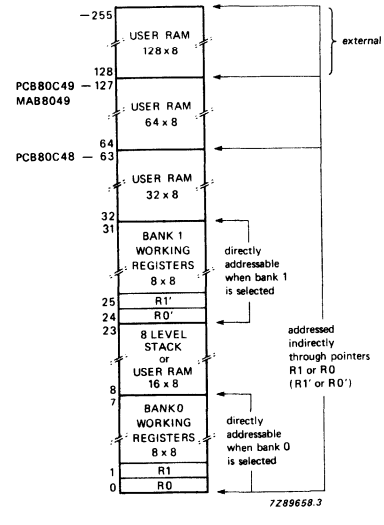


Fig. 3.2 Data memory map.

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 respectivly. 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 seperate 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.

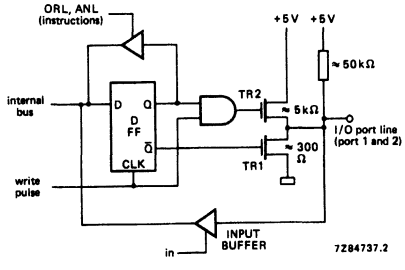


Fig. 3.3 Quasi-bidirectional port structure of MAB8048 family

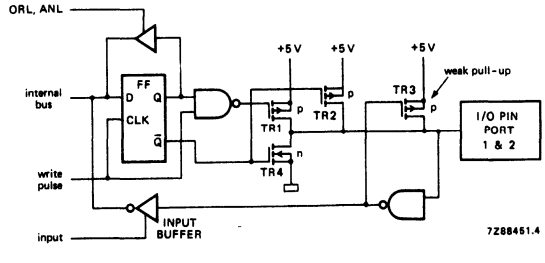


Fig. 3.4 Quasi-bidirectional port structure of PCB80C49 family

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 (300Ω) overcomes the light pull-up 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 2V 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 5V. 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, overcomes this light pull-up 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. $\overline{\text{RESET}}$ 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.

* 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.

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 $\overline{\text{RD}}$ and $\overline{\text{WR}}$ 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 $\overline{\text{WR}}$ output line and output data is valid at the trailing edge of $\overline{\text{WR}}$. A read of the port generates a pulse on the $\overline{\text{RD}}$ output line and input data must be valid at the trailing edge of $\overline{\text{RD}}$. 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 ($\overline{\text{INT}}$, 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.

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

1. WHEN INTERRUPT IN PROGRESS FLIP-FLOP IS SET ALL FURTHER INTERRUPTS ARE LOCKED OUT INDEPENDNET OF STATE OF EITHER INTERRUPT ENABLE FLIP-FLOP.
2. WHILE TIMER INTERRUPTS ARE DISABLED TIMER OVERFLOW FF WILL NOT STORE ANY OVERFLOW THAT OCCURS. TIMERFLAG WILL BE SET, HOWEVER.

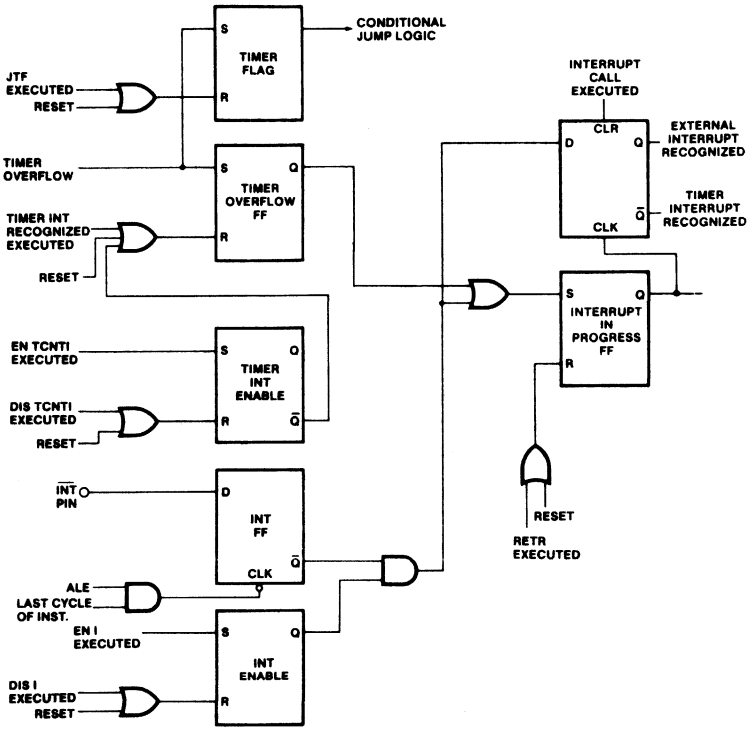


Fig. 3.5 The interrupt logic

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.

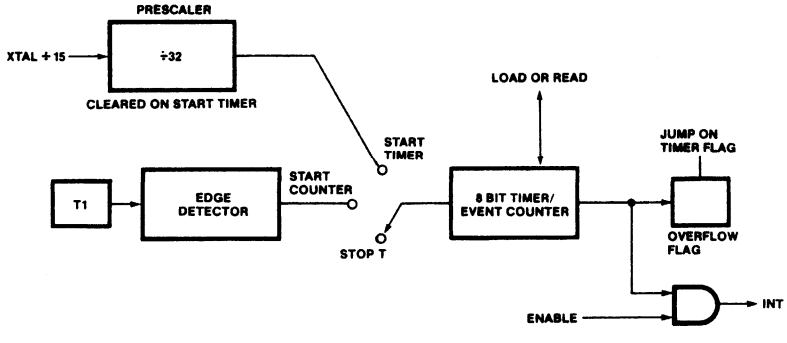


Fig. 3.6 Timer/event counter

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.

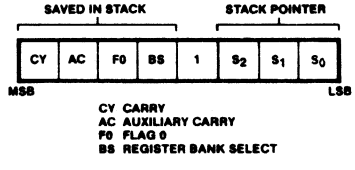


Fig. 3.7 Program status word (PSW).

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.

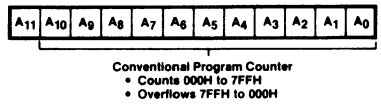


Fig. 3.8 Program counter

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.

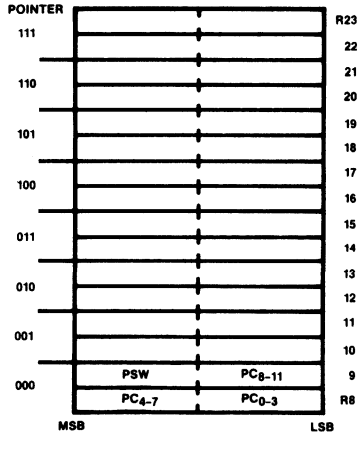


Fig. 3.9 Program counter stack.

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 5.1 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.

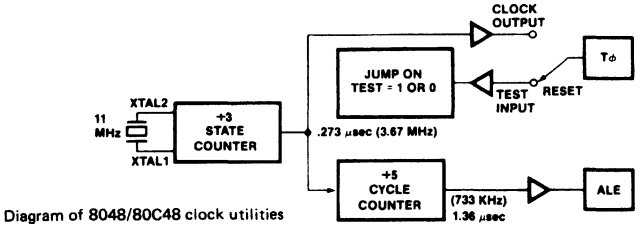
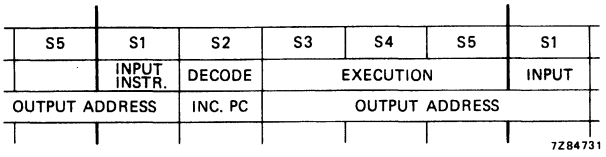
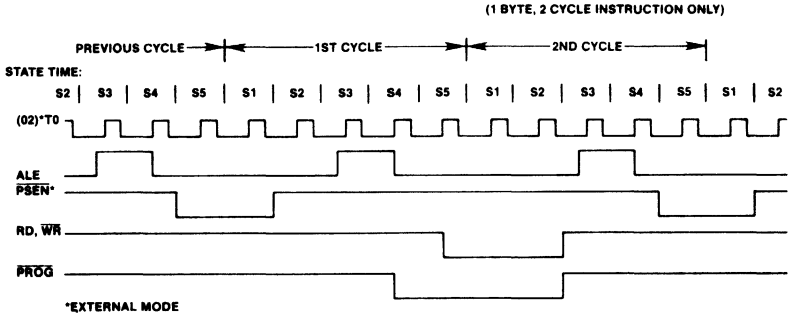


Diagram of 8048/80C48 clock utilities



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 $\overline{RESET}$ pin must be held at ground (0,45 V) for at least 10ms 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.

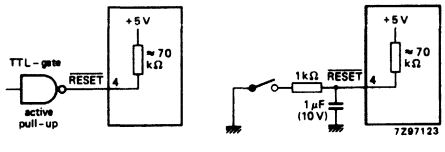


Fig. 3.11 Reset circuitry.

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 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.

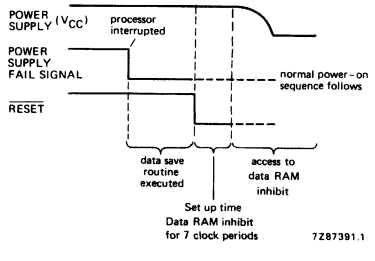


Fig. 3.12 Power-down sequence

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.

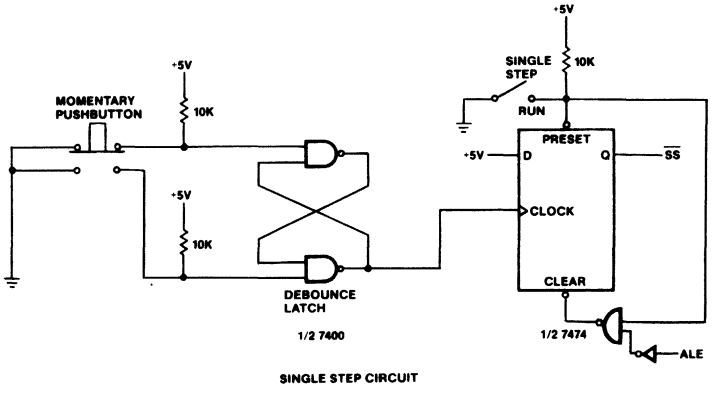


Fig. 3.13 Single step circuit

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 $\overline{SS}$.

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. $\overline{SS}$ 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 $\overline{SS}$ must be brought LOW again soon after ALE goes LOW. If $\overline{SS}$ 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 $\overline{SS}$. In the run mode $\overline{SS}$ 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 $\overline{SS}$ 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 $\overline{SS}$ unless ALE is HIGH removing clear from the flip-flop. In response to $\overline{SS}$ going HIGH the processor begins an instruction fetch bringing ALE LOW, resetting $\overline{SS}$ 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 8048/80C48 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.

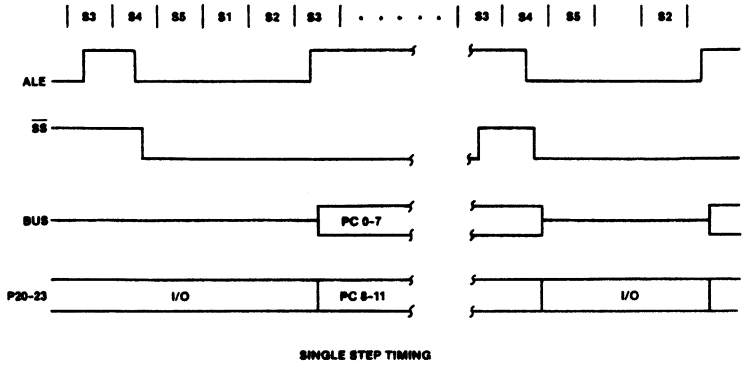


Fig. 3.14 Single step timing

INSTRUCTION SET SUMMARY

Table 1

Mnemonic	Description	Bytes	Cycle
Accumulator			
ADD A,R	Add register to A	1	1
ADD A,@R	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,R	Add register with carry	1	1
ADDC A,@R	Add data memory with carry	1	1
ADDC A,#data	Add immediate with carry	2	2
ANL A,R	And register to A	1	1
ANL A,@R	And data memory to A	1	1
ANL A,#data	And immediate to A	2	2
ORL A,R	Or register to A	1	1
ORL A,@R	Or data memory to A	1	1
ORL A,#data	Or immediate to A	2	2
XRL A,R	Exclusive Or register to A	1	1
XRL A,@R	Exclusive Or data memory to A	1	1
XRL A,#data	Exclusive Or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A,P	Input port to A	1	1
OUTL P,A	Output A to port	1	2
ANL P,#data	And immediate to port	2	2
ORL P,#data	Or immediate to port	2	2
INS A,BUS	Input BUS to A	1	2
OUTL BUS,A	Output A to BUS	1	2
ANL BUS,#data	And immediate to BUS	2	2
ORL BUS,#data	Or immediate to BUS	2	2
MOVD A,P	Input Expander port to A	1	2
MOVD P,A	Output A to Expander port	1	2
ANLD P,A	And A to Expander port	1	2
ORLD P,A	Or A to Expander port	1	2
Registers			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC 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 MAB8021 FUNCTIONAL DESCRIPTION

The following is a functional description of the major elements of the MAB8021.

4.1 Program memory (ROM)

The MAB8021 contains 1 K x 8 bytes of mask programmable ROM. No external ROM expansion capability is provided.

4.2 Data memory (RAM)

A 64 x 8 dynamic RAM is located on-chip for data storage. All locations are indirectly addressable with eight locations designated as directly addressable working registers. Also included in the memory is the stack, addressed by a 3-bit stack pointer.
Memory is organized as show in figure 4.1. The lowest 8 addresses 0-7, are directly addressable by any of the direct register instructions. These locations are readily accessible for a variety of operations, using only single byte instructions.

Register 0 and 1 have another function; they can be used to indirectly address all locations in memory using the indirect register instructions. These indirect RAM address registers, IRAR's, are especially useful for repetitive operations on adjacent memory locations. The indirect register instruction specifies which IRAR to use, and the contents of the IRAR are used to address a location in RAM. The contents of the addressed location are used during the execution of the instruction and may be modified. A value larger than 63 should not be present in the IRAR when selected by an indirect register instruction. IRARs may point to addresses 0-7, if desired.

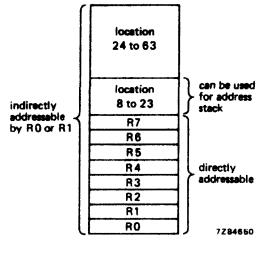


Fig. 4.1 Internal RAM organization

Locations 8-23 may be used as the stack. The stack enables the processor to keep track of the return addresses generated from CALL instructions. A 3-bit pointer (SP) supplies the address of the locations to be loaded with the next return address generated. The SP to this pushdown stack is incremented by one after a return address is stored, and decremented by one before an address is fetched during a RET. The unaffected program counter address is stored in the address stack. Before loading the program counter during a return from the subroutine (RET), the stack contents are incremented. A total of 8 levels of nesting is possible. The SP is initialized to location 8 upon $\overline{\text{RESET}}$. Since each address is 10-bits long, two bytes must be used to store a single address. The SP is incremented and decremented by one, but each increment or decrement moves the address pointed to by two. Therefore, only even numbered addresses are pointed to. If a particular application does not require 8 levels of nesting, the unused portion of the stack may be used as any other indirectly addressable RAM location. For example, if only 3 levels of subroutine nesting are used, then only locations 8 - 13 need be reserved for the stack, and locations 14 - 63 can be used for data storage.

4.3 Input/Output

The MAB8021 I/O configurations are highly flexible. A number of different configurations are possible, tailoring an MAB8021 to a given task. Other than the power supply and dedicated pins, all other pins can be used for input, output, or both, depending on the configuration.

All ports are quasi-bidirectional to facilitate stand-alone use. A simplified schematic of the quasi-bidirectional interface is shown in Figure 4.2. This configuration allows buffered outputs, and also allows external input. 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. When writing a '0' to these ports, the low impedance pull-down device sinks an external TTL load. When writing a '1', a large current is supplied through the low impedance pull-up device to allow a fast data transfer. After a short time (less than one instruction cycle), the low-impedance device is shut off and the high-impedance pull-up maintains the HIGH level indefinitely. However, in this situation, an input device capable of overriding the small amount of sustaining current supplied by the pull-up device can be read. (Alternatively, the data written can be read). So, by writing a '1' to any particular pin, that pin can serve either as a true HIGH-level latched output pin, or as just a pull-up resistor on an input. This allows maximum user flexibility in selecting input or latched output pins, with a minimum of external components.

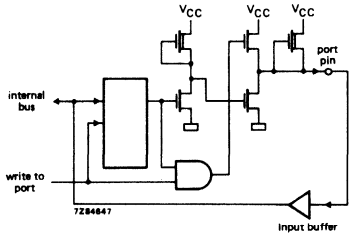


Fig. 4.2 Quasi-bidirectional port structure.

Port 00-07 is also quasi-bidirectional, except there is no low-impedance pull-up device. As outputs, this port is essentially open drain plus the high impedance pull-up.

As a mask option the high-impedance pull-up devices on P00-P07 may be deleted on any pin providing a true open drain output. This is useful in driving analogue circuits and certain loads, such as keyboards.

4.4 Timer/event counter

The MAB8021 has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit has an 8-bit binary up-counter that is presettable and readable with two MOV instructions. These instructions transfer the contents of the accumulator to the counter and vice versa. The counter is cleared by $\overline{\text{RESET}}$ and can be set by the MOV T,A instruction. The counter is stopped by a $\overline{\text{RESET}}$ or STOP TCNT instruction and remains stopped until started as a timer by a STRT T instruction or as an event counter by a STRT CNT instruction. Once started, the counter increments to its maximum count (FF), and overflows to zero. The count continues until stopped by a STOP TCNT instruction or $\overline{\text{RESET}}$. The increment from maximum count to zero (overflow) sets an overflow flag. The state of the overflow flag is testable with the conditional jump instructions JTF. The flag is reset by JTF, not by executing a $\overline{\text{RESET}}$ like the MAB8048H.

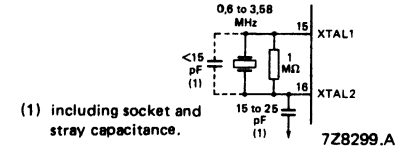
At the STRT T command, the internal timer/counter prescaler is zeroed and thereafter increments after every 30 oscillator periods (once each single cycle instruction, twice each double cycle instruction). As the prescaler divides by 32, a (11111) to (00000) transition increments the timer/counter. The timer/counter is an 8-bit register and after an overflow from FFH to 00H, the timer flag is set. A conditional branch instruction (JTF) is available for testing this flag, the flag being reset each test. Total count capacity for the timer is $2^8 \times 2^5 = 8192$ or 81,9 ms at a 10 µs cycle time. Contents of the timer can be moved to the accumulator by the MOV A,T instruction without disturbing the counting process.

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the leading edge of state 3. Subsequent HIGH to LOW transitions on T1 will cause the counter to increment. T1 must be held LOW for at least 1 machine cycle to ensure it will not be missed. The maximum rate at which the counter may be incremented is once per three machine cycles (every 30 µs for a 3 MHz oscillator) - there is no minimum rate. T1 input must remain stable for at least 1/5 machine cycle after each transition.

4.5 Oscillator and clock

The MAB8021 contains its own internal oscillator and clock driver. The frequency is determined by a single crystal, or inductor. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. The particular control element is connected between pins 15 (XTAL1) and 16 (XTAL2).

An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. Therefore, to obtain a 10 µs instruction cycle, a 3,3 MHz crystal should be used.



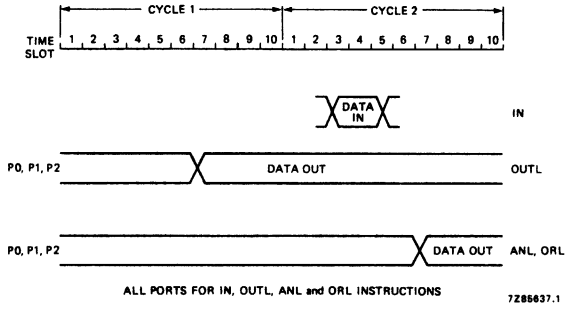


Fig. 4.3 MAB8021 timing diagram.

The MAB8021 uses dynamic RAM and certain other types of dynamic logic. Due to the clocking required with dynamic circuits, the oscillator frequency must be equal to or greater than 600 kHz, or inadequate charge refreshing of these circuits will result.

4.6 Central processing unit

The MAB8021 CPU has arithmetic and logical capability. A variety of arithmetic and logic instructions may be exercised that affect the contents of the accumulator and/or direct or indirect scratchpad locations. Provisions have been made for elementary BCD arithmetic using the DAA, SWAP A and XCHD instructions. In addition, MOV A,@A allows table look-up for display formating and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the user's program. Programmers should use the conditional jump instructions with the tests listed below to effect changes in program execution sequence.

Table 3 Conditional branches

Test	Jump Condition	Jump Instructions
Accumulator	A = 0 A = 0	JZ JNZ
Carry flag	0 1	JC
Timer overflow flag	- 1	JTF
test input T1	0 1	JNT1,JT1

4.7 T1 input

The MAB8021 T1 input line can be used as an input for the following functions:

- Event Counter (external input)
- Test input for branch instructions
- Zero voltage crossing detection

The operation of T1 as an input to the Event Counter is described in the Timer/ Event Counter section. When used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels, respectively.

The T1 pin can also be used to detect the zero-crossing of slowly moving AC signals (60 Hz). The self baising circuit shown in figure 4.4 permits the Test 1 input to detect when the input voltage crosses zero within + 5%: the voltage is then coupled through a 1,0 µF capacitor. Maximum input voltage is 3 V peak-to-peak. The zero-cross detection is especially useful in SCR control of 50 Hz power and in developing time-of-day and other timing routines. As a RAM mask option there is a pull-up device that is useful for switch contact input or standard TTL.

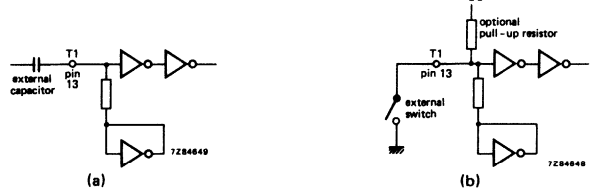


Fig. 4.4 Self-biasing circuit

4.8 High current outputs

High current drive is desirable for minimizing external parts required to do high power control. P10 and P11, MAB8021 only, have been designated high drive outputs.

4.9 Expanded I/O

The MAB8021 may be used with the 8243 I/O expander chip, which provides additional I/O capability with a limited number of overhead pins. This chip has 4 directly addressable 4-bit ports. The I/O expander chip connects to the $\overline{\text{PROG}}$ pin, which provides a clock, and pins P20-P23, which hold addresses and data. These ports can be written with the instructions:

```
MOVD P,A
ANLD P,A
ORLD P,A
```

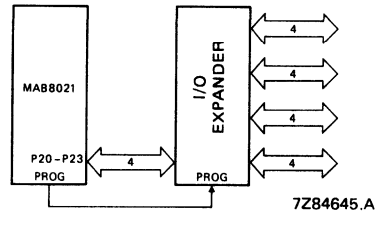


Fig. 4.5 I/O expander interface.

A HIGH to LOW transition on $\overline{\text{PROG}}$ signifies that address and control data are available on P20-P23. Any data on P20-P23 prior to an output expander instruction is lost. Therefore, P20-P23 are not useful for general input/output when an output expander is used. Reading is via the MOVD A,P. This circuit configuration is shown in Figure 4.5. The timing diagram is shown in Figure 4.6. Details of the operation of an I/O expander are given in Table 4.

Table 4 Expander operation

Parameter	Symbol	Min.	typ.	Max.	Unit	Conditions
Port control setup before falling edge of PROG	t_{CP}	0,3	-		µs	
Port control hold afer falling edge of PROG	t_{PC}	0,8	-		µs	
PROG to time P ± input must be valid	t_{PR}	2,0	-	4,0	µs	
Output data setup time	t_{DP}	1,0	-	-	µs	
Input data hold time	t_{PF}	0	-	0,15	µs	
PROG pulse width	t_{pp}	6,0	-	-	µs	

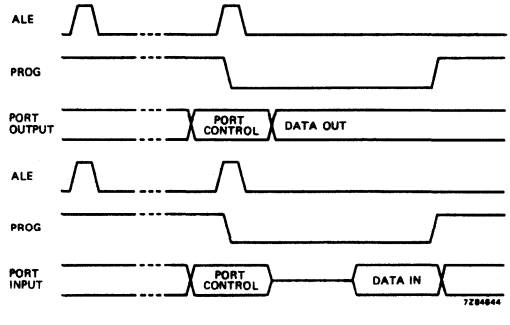


Fig. 4.6 Expanded I/O timing

4.10 Reset

The RESET input provides a means for initializing the processor. This Schmitt-trigger input has an internal pull-down device which in combination with an external 1 µF capacitor, provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in figure 4.7. If the reset pulse is generated externally, the RESET pin must be held above 3,8 V for at least 10 ms after the power supply has stabilised. Only 3 machine cycles (2,5 µs at 3,6 MHz) are required if power is already on and the oscillator has stabilized.

Reset performs the following functions:

1. Sets program counter to zero
2. Sets stack pointer to zero
3. Sets Ports to input mode
4. Disables interrupts (timer and external)
5. Stops timer
6. Clears timer register

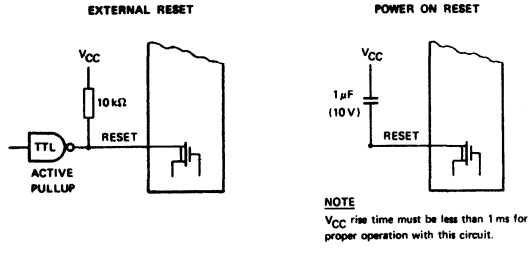


Fig. 4.7 Reset circuitry

4.11 Test and debug

To facilitate testing and debug, certain test modes may be activated in the MA88021 by raising combinations of RESET, TEST 1 and PROG to 15 volts. Internal ROM is dumped out sequentially for verification. External memory operation is used for CPU checkout (see Table 5).

Table 5 Activating test modes

Reset	Prog	Test 1	Case	Fuction
5 V	X	X		Power On Clear
0 V	X	X		Normal Operation
15 V	15 V	15 V	Mode 1a	On each cycle internal ROM is dumped to port 0 - Sequentially after ALE leading edge
15 V	15 V	_I¯I_	Mode 1b	On every TEST 1 falling edge the program counter increments, dumps internal ROM to Port 0
0 V	15 V	X	Mode 2	Chip will operate from external memory (one page) via Port 0. ALE strobes Address Out, memory in
15 V	X	X	Mode 3	Chip accepts op codes into Port 1. Allows Port 0 and I/O expander chip testing

NOTE: X = Normal mode - between 0 V and V_{CC}
Test 1 in Mode 1B should be limited to V_{CC}
Mode 3: I/O Expander chip testing for MAB8021

4.12 Differences between MAB8021 and MAB8048H

Although the MAB8021 is basically an electrical and fuctional decendant of the MAB8048H, there are some differences:

1. Pin Out - As the MAB8021 is a 28-pin DIP, some form of interface must be used with the MAB8048H to simulate the MAB8021,

2. Instruction Time - The MAB8021 instruction cycle is 30 clock periods long, the MAB8048H instruction cycle is 15 clock periods long. Where exact timing is important, the MAB development breadboard should be operated at half the MAB8021 clock rate.

3. Test 1 - To facilitate developing time-of-day routines from 50 Hz, and for SCR control, the Test 1 pin without the pull-up resistor option will detect zero-crossing of a capacitively coupled AC input.

4. Quasi-bidirectional Ports - All MAB8021 ports are quasi-bidirectional to facilitate stand-alone use. Port 0 has open drain outputs and by mask option it may or may not have pull-up devices.

5. Oscillator - The MAB8021 has an on-chip oscillator that is optimized for the single inductor mode. External components will differ from the MAB8048H.

6. Dynamic RAM and Logic - The MAB8021 utilizes dynamic RAM and some dynamic logic. Input clocking must be maintained above the minimum rate or improper operation may result.

7. High Current Outputs - High current drive is desirable for minimizing external parts required for high power control. P10 and P11 have been designated high drive outputs capable of sinking 7 mA at V_{SS} + 2,5 volts. (For clarity, this is 7 mA to V_{SS} with a 2,5 volt drop across the buffer). These pins may, of course, be paralleled for 14 mA drive if the output logic states are always the same.

8. Timer/Counter -

   1. If prescaler overflow occurs during a 'STRT T' or 'STOP TCNT' instruction, the MAB8048H will increment the timer, whereas the MAB8021 will not.

   2. The MAB8021 sets the timer flag in the same cycle as the overflow. The MAB8048H waits one cycle. Therefore, the MAB8021 can do a JTF instruction one cycle earlier (prescaler = '0') than the MAB8048H (prescaler = '1').

9. RESET - Reset has been modified on the MAB8021 to active HIGH; the MAB8048 is active LOW.

10. Instruction Set - The instructions below, which are found in the MAB8048H have been deleted from the MAB8021 instruction set.

4.13 Instructions in MAB8048H instruction set not found in MAB8021

Table 6 Instruction set differences

DATA MOVES		REGISTERS		BRANCH		SUBROUTINE		CONTROL		INPUT/OUTPUT	
MOV	A,PSW	DEC	R	JT0	addr	RETR		EN	I	ANL	P,#data
MOV	PSW,A	FLAGS		JNT0	addr	TIMER		DIS	I	ORL	P,#data
MOVX	A,@R			JF0	addr			SEL	RB0	INS	A,BUS*
MOVX	@R,A	CLR	F0	JF1	addr	EN	TCNTI	SEL	RB1	OUTL	BUS,A*
MOVP3	A,@A	CPL	F0	JNI	addr	DIS	TCNTI	SEL	MB0	ANL	BUS,#data
		CLR	F1	JBb	addr			SEL	MB1	ORL	BUS,#data
		CPL	F1					ENT0	CLK		

* These instructions have been replaced in the MAB8021 by IN A,P0 and OUTL P0,A respectively. OUTL P0,A has the same opcode as MOVX @R,A.

5.0 EXPANDED 8048/80C48 FAMILY SYSTEM (NOT MAB8021)

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 - On 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.

5.1 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.

5.1.1 Instruction fetch cycle (external)

This is not included on the MAB8050H because of it's 4 K byte program memory.

As shown in Figure 5.1, 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 ($\overline{\text{PSEN}}$) 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.

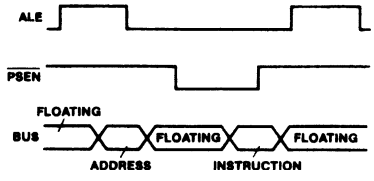


Fig. 5.1 Instruction fetch from external program memory.

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.

5.1.2 Extended program memory addressing (beyond 2K 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.

5.1.2.1 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.

5.1.2.2 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.

5.1.3 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 besampled or latched at this time.

5.1.4 Expansion example

Figure 5.2 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.

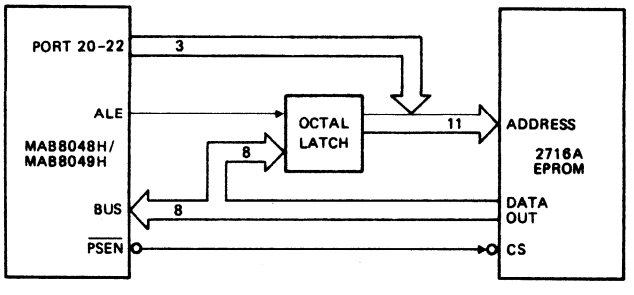


Fig. 5.2 Expanding program memory using standard memory components.

5.2 Expansion of data memory

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

5.2.1 Read/write cycle

All address and data is transferred over the 8 lines of BUS. As shown in figure 5.3, 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.

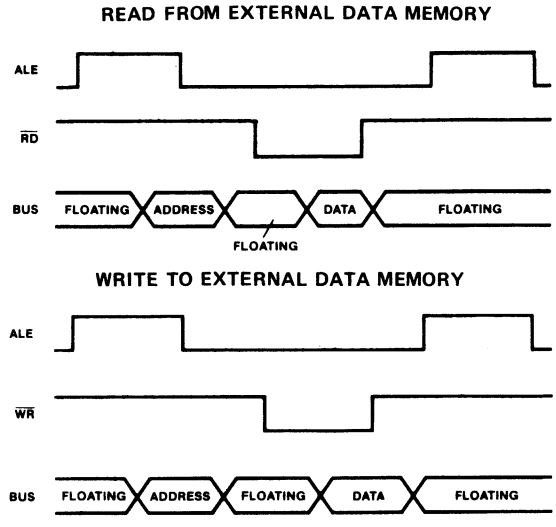


Fig. 5.3 External data memory timing.

5.2.2 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.

5.2.3 Data memory expansion example

Figure 5.4 shows how the MAB8048H can be expanded using a 256 x 8 standard memory.

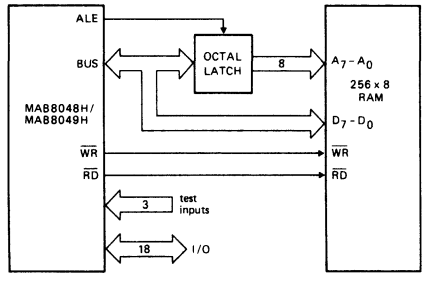


Fig. 5.4 MAB8048H to 256 x 8 standard memories.

5.3 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.

5.3.1 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. 5.5). 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

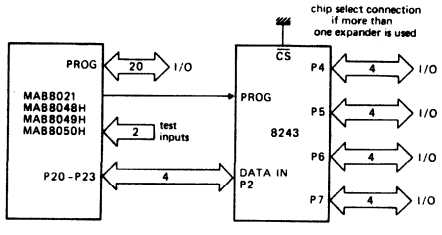
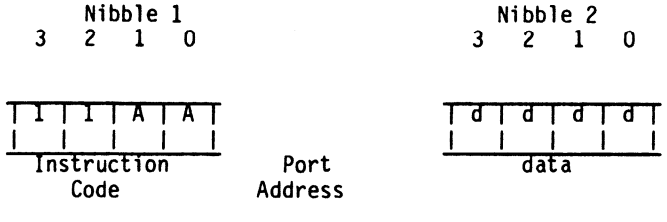


Fig. 5.5 Expander I/O interface.

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 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.

5.4 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.

5.5 Control signal summary

Table 7 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 7 8048/80C49 control signals

Control Signal	When active
$\overline{RD}$	During MOVX A,@R or INS Bus
$\overline{WR}$	During MOVX @R,A or OUTL Bus
ALE	Every machine cycle
$\overline{PSEN}$	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 microcontroller family

CONTENTS – THE MAB8051/C51 MICROCONTROLLER FAMILY

(continued on next page)

1.0 DESCRIPTION

The 8051/80C51 family of 8-bit microcomputers is manufactured in NMOS N500 and CMOS C500 process, respectively. The 8051/80C51 family consists of:

MAB8051 - single-chip microcomputer
MAB8031 - ROMless version of MAB8051

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

Throughout this section of the Microcomputer User Manual, 8051 is used as the generic term for the 8051/80C51 family of microcomputers. 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 microcomputer 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/PCB80C51 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 the device can begin receiving a second byte before a previously received byte has been read from the receive register and it can transmit and receive simultaneously.

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

2.0 FEATURES

- 8-bit CPU
- 4K bytes of ROM on MAB8051 and PCB80C51
- 128 bytes of RAM
- 21 special function registers
- 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
- Five interrupt sources with a two priority-level structure
- A fully duplex serial I/O
- Bit addressability for Boolean processing
- Reduced power consumption with CMOS versions

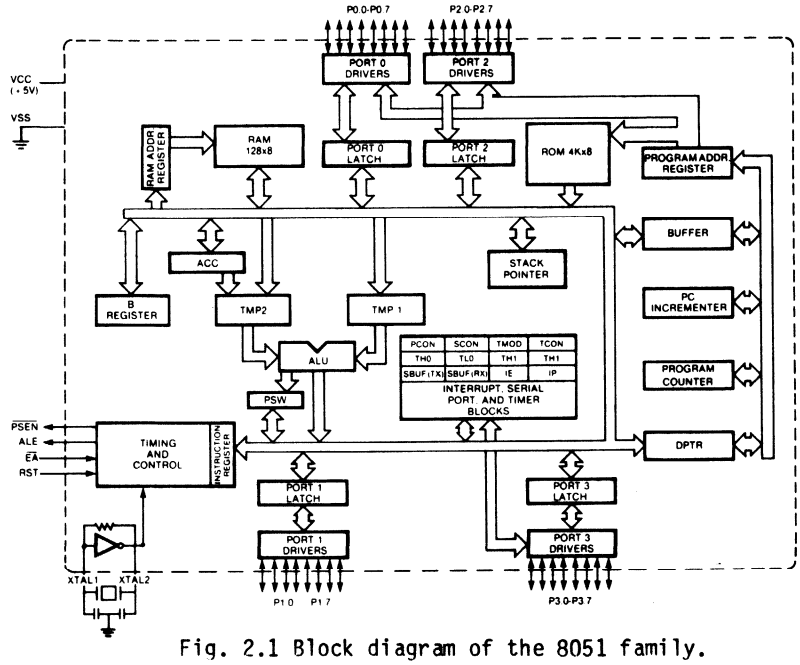
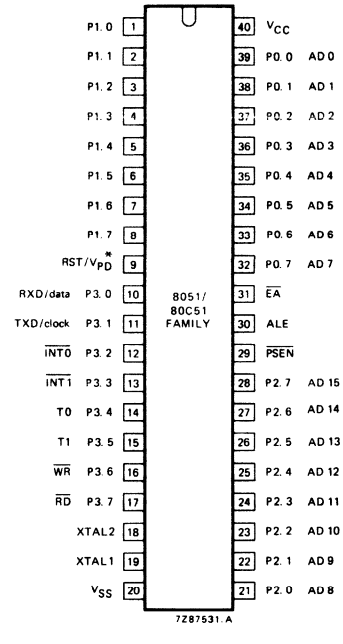


Fig. 2.1 Block diagram of the 8051 family.



* V_{PD} applicable to NMOS versions only.

Fig. 2.2 Pinning diagram for the 8051 family.

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.

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 though 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.)

$\overline{PSEN}$: 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. (However, 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 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). When $\overline{EA}$ is held LOW the CPU executes only out of external program memory. In the MAB8031 and PCB80C31 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 4K-bytes of this memory are in the on-chip ROM. The program memory 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 OFFFH. Fetches from locations 1000H up to FFFFH 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 and PCB80C31 which have no internal program memory, $\overline{EA}$ must be externally wired LOW to enable the proccessor to fetch the lowest 4K-bytes from the external program memory. The program memory map is shown in Fig. 3.1.

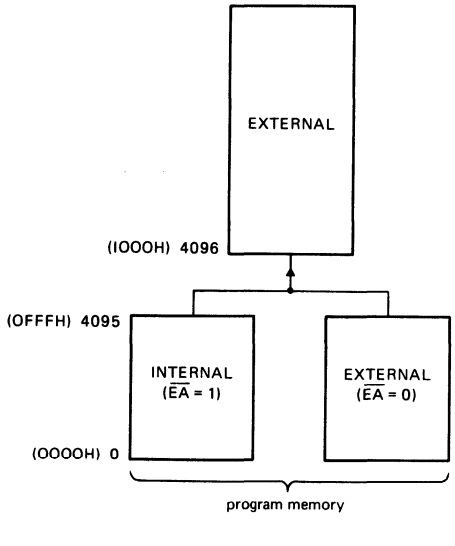


Fig. 3.1 Program memory map.

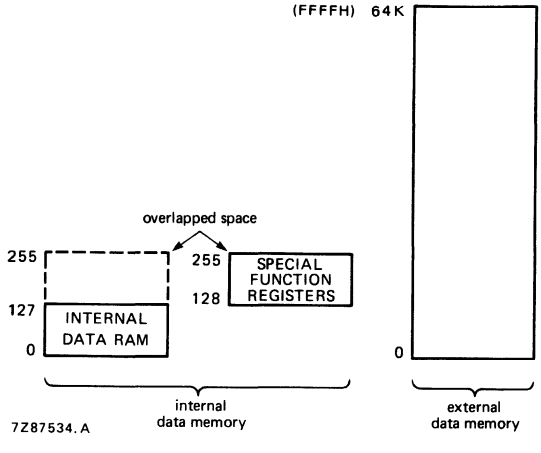


Fig. 3.2 Data memory map.

3.2 Data memory

The data memory comprises 128-bytes of on-chip RAM, 21 special function registers and up to 64K-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 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 scratch 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 outwith 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.

* 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 multifunctional, 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)

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

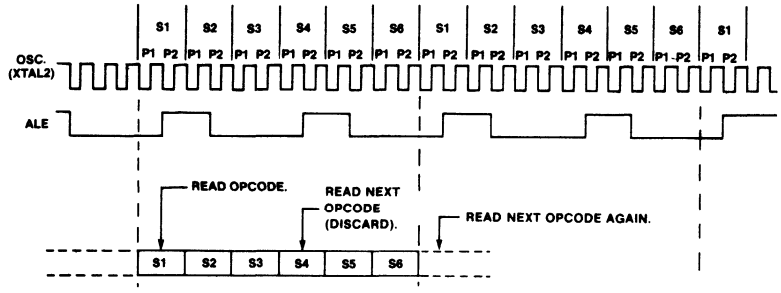
A machine cycle consists of 6 states (12 oscillator periods) and each state is divided into two phases corresponding to the two phases of the clock signal. Normally arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place in Phase 2.

The diagrams of Fig. 3.3 show the fetch/execute timing referred to the internal states and phases. Since these internal clock signals are not externally observable, 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.

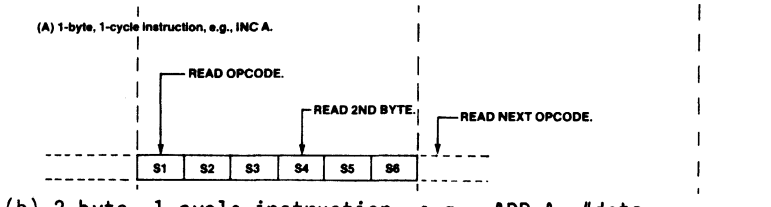
Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the instruction register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one byte instruction, there is still a read at S4, but the byte read (which would be 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.

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.

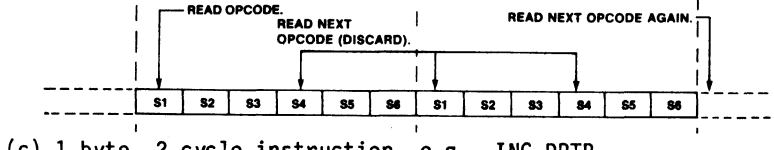
Normally, two code bytes are fetched from program memory during every machine cycle. The only exception to that is when a MOVX instruction is executed. MOVX is a 1-byte, 2-cycle instruction that accesses the external data memory, during which two fetches are skipped while the external data memory is 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.



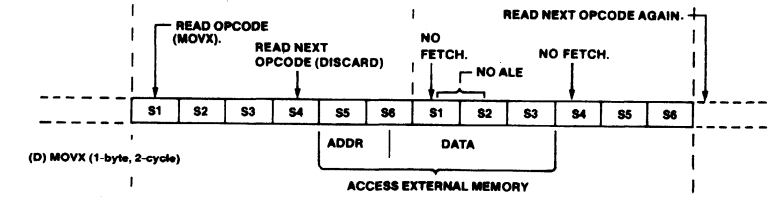
(a) 1-byte, 1-cycle instruction, e.g., INC A.



(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.

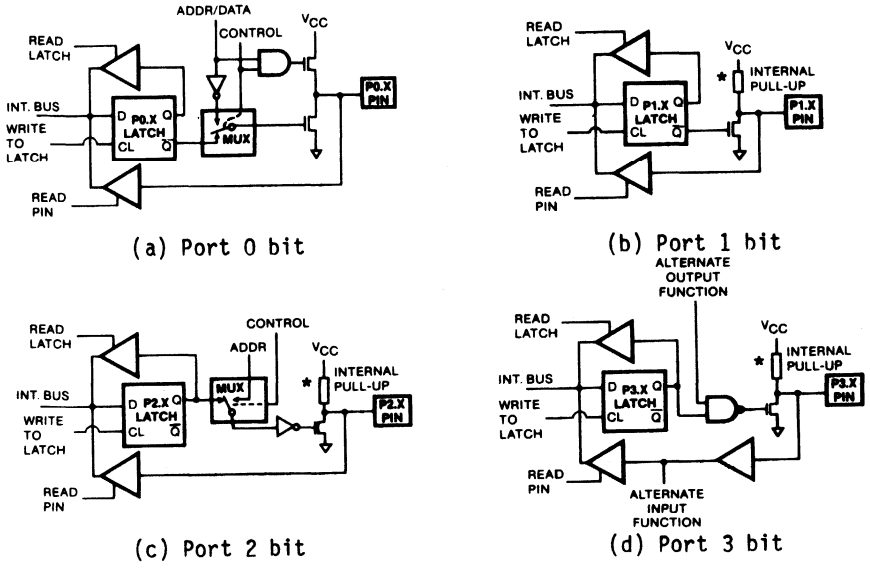


Fig. 3.4 Input/Output port latches and buffers.

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 shorted to ground, this transistor will allow about 0,25mA (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 shorted to ground, this extra transistor will allow an additional 30mA (typical) to exit the pin.

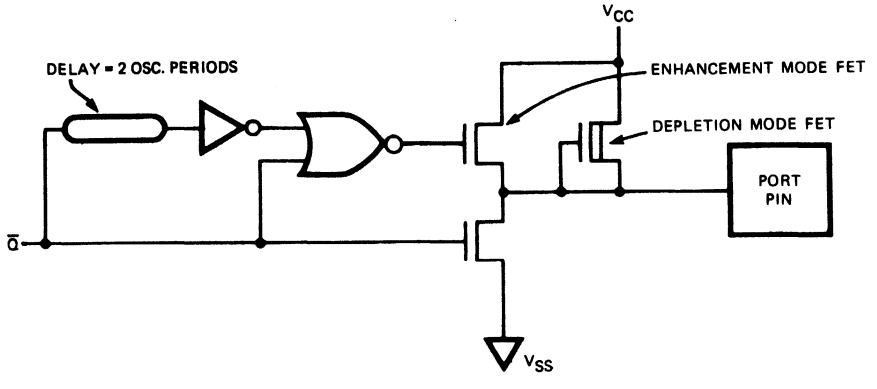


Fig. 3.5 The arrangement of the internal pull-ups for the MAB8051/8031

3.6.3 Port loading

The output buffers of ports 1, 2, and 3 can drive three 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).

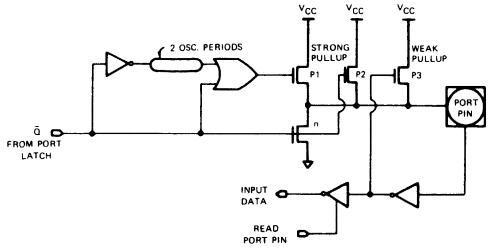


Fig. 3.6 I/O buffers in the PCB80C51/80C31 (ports 1, 2, and 3).

When the port latch of port 1, 2 or 3 contains a 0 ($\bar{Q}$ = 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.

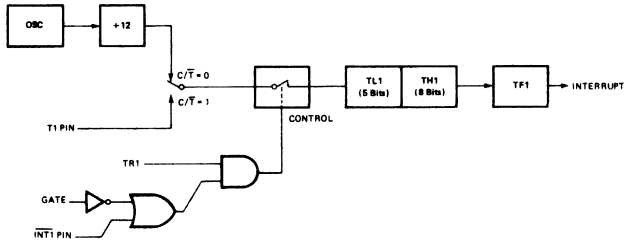
3.7 Timer/event counter

The 8051 has two 16-bit registers, Timer 0 and Timer 1, which can be used as timers or event counters. 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'.

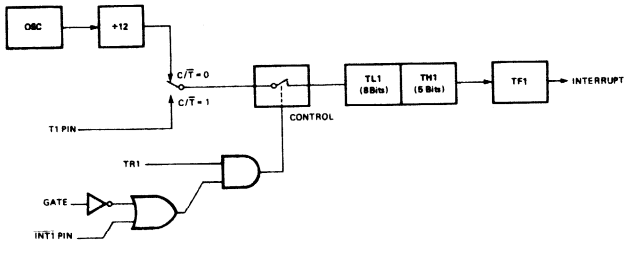
In the 'timer' function the register is incremented every machine cycle and therefore it can be thought of as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

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.

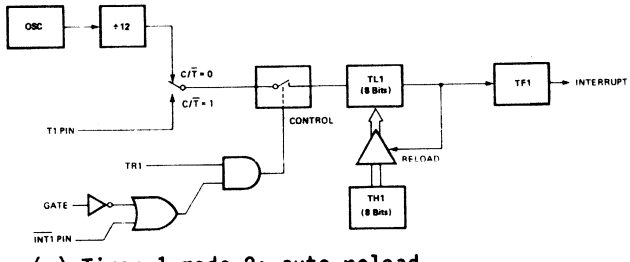
In addition to the 'timer' or 'counter' selection, each timer/event counter has four operating modes from which to select. These modes are functionally illustrated in Fig. 3.7. Operating modes 0, 1, and 2 are the same in both timer/event counters, but in mode 3 they differ.



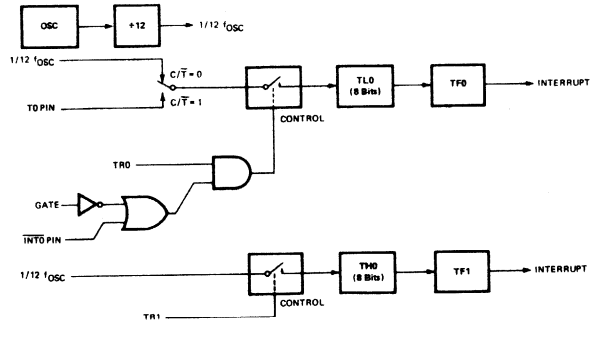
(a) Timer 1 mode 0: 13-bit counter.



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



(c) Timer 1 mode 2: auto reload.



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

Fig. 3.7 Timer/counter modes.

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. 3.11a 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.

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 $\overline{INT1}$ 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 $\overline{INT1}$ rather than a 1 in IE1.

IE0: is the external interrupt 0 edge flag. If IT0 = 1, this bit is set by hardware when $\overline{INT0}$ 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 $\overline{INT0}$ rather than a 1 in IE0.

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.

3.8 Serial 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 still has 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. 3.8. 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. 3.8 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 in this case.

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 should be cleared for operation in Mode 0 or 1.

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.

Baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate, as shown below:

Baud rate = (Timer 1 overflow rate)/n

The value of the 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 $C/\overline{T}$ = 0, the count rate is 1/12 of the oscillator frequency. If the timer is running with $C/\overline{T}$ = 1, the count rate is the external input frequency, whose maximum practical value is 1/24 of the oscillator frequency.

Table 3.1 lists the various commonly used baud rates and how they can be achieved with the 8051.

BAUD RATE	f_{osc}	SMOD	TIMER 1		
			$C/\overline{T}$	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 3.1 Baud rates in common use.

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 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: 8 data bits LSB first. The baud rate is fixed at 1/12 of the oscillator frequency.

Fig. 3.9 shows a simplified functional diagram of the serial port in mode 0, and the associated timing diagrams.

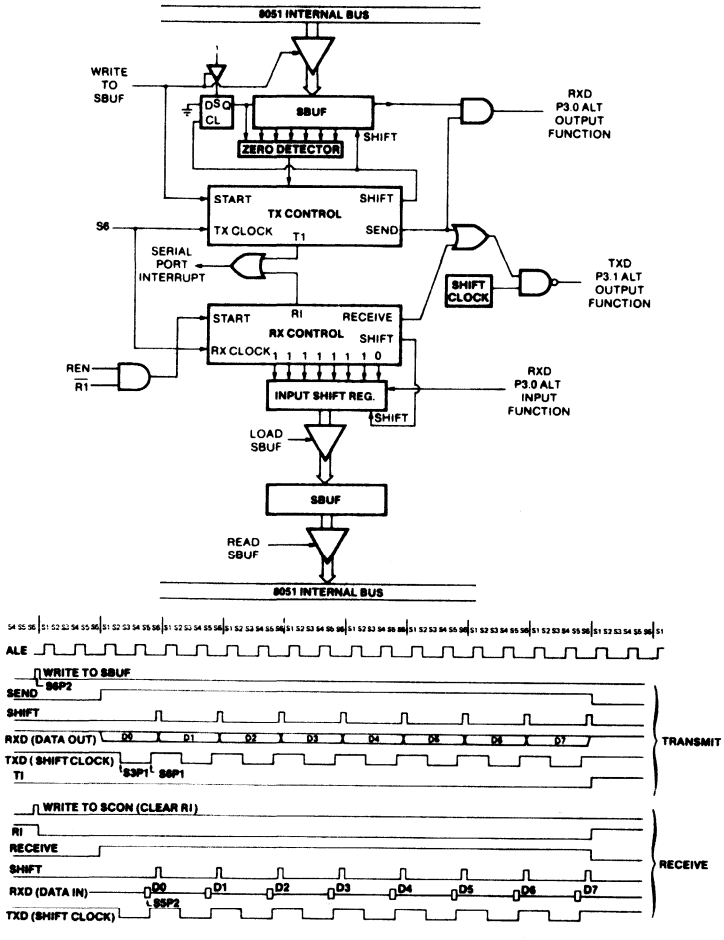


Fig. 3.9 The serial port in mode 0.

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 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. 3.10.

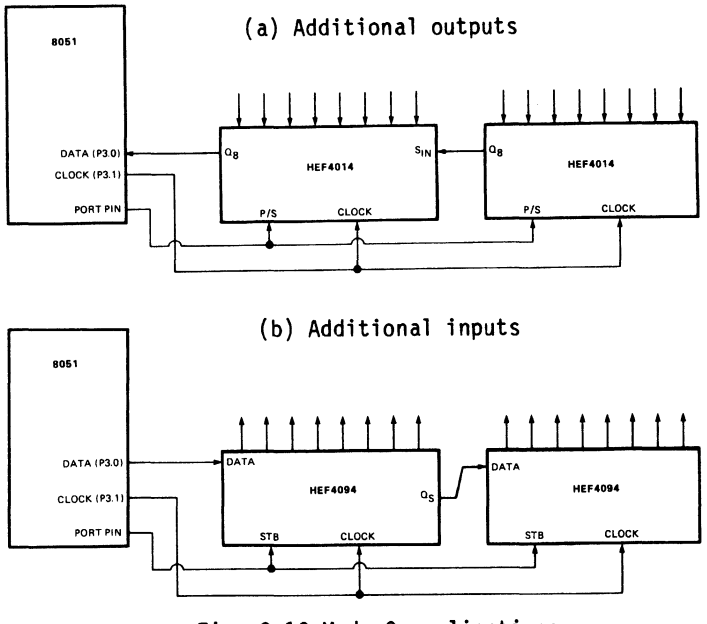


Fig. 3.10 Mode 0 applications.

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 is SCON. The baud rate is determined by the Timer 1 overflow rate.

Fig. 3.11 shows a simplified functional diagram of the serial port in mode 1 and the associated timing diagrams for transmit and receive.

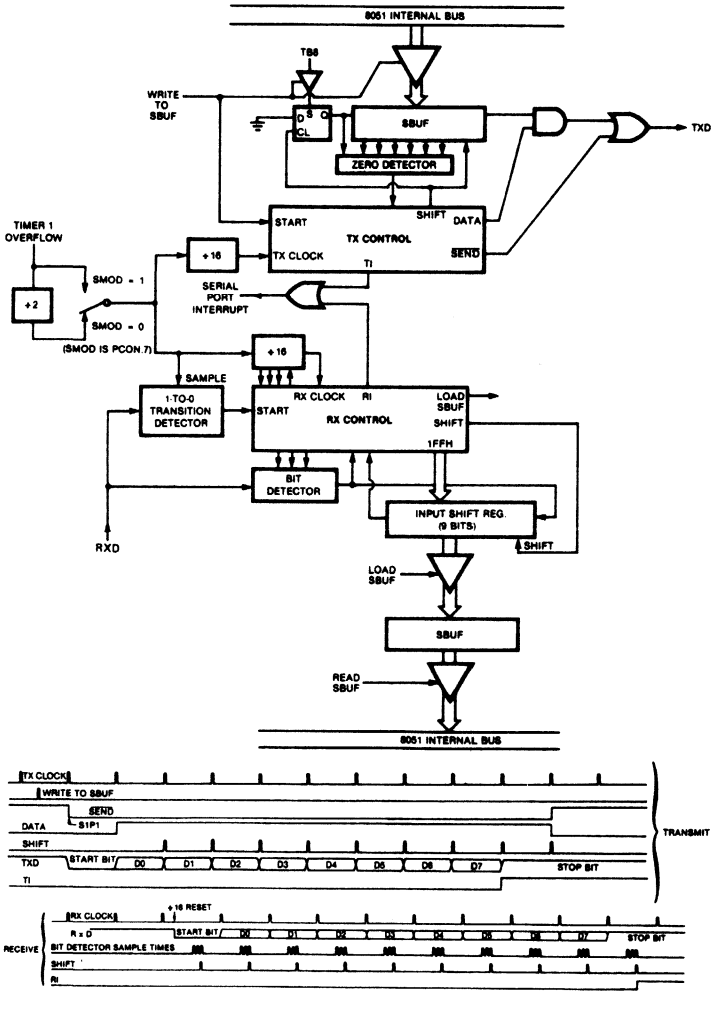


Fig. 3.11 The serial port in mode 1.

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.

Fig. 3.12 shows a simplified functional diagram of the serial port in modes 2 and 3. 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.

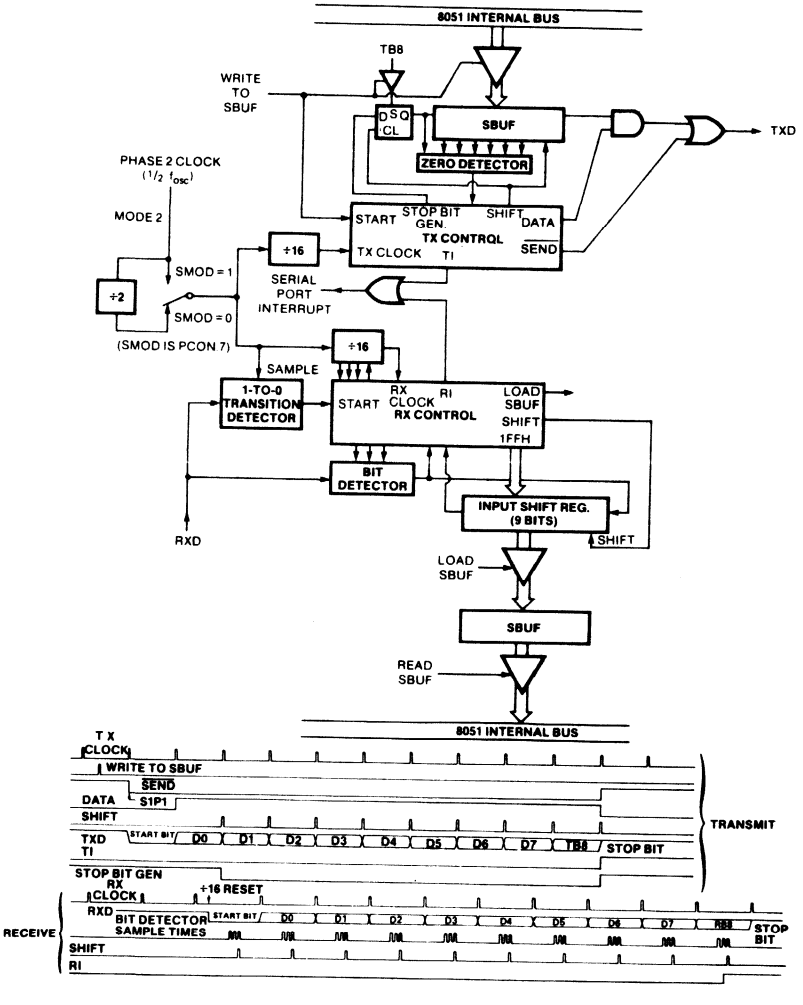


Fig. 3.12 The serial port in modes 2 and 3.

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 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.)

The transmission begins with the activation of $\overline{\text{SEND}}$, 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 $\overline{\text{SEND}}$, 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.

3.9 Accessing external memory

Accesses to external memory are 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

Accesses 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.

Whatever the 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;

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 4K 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, $\overline{PSEN}$ 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. 3.13 for comparison.

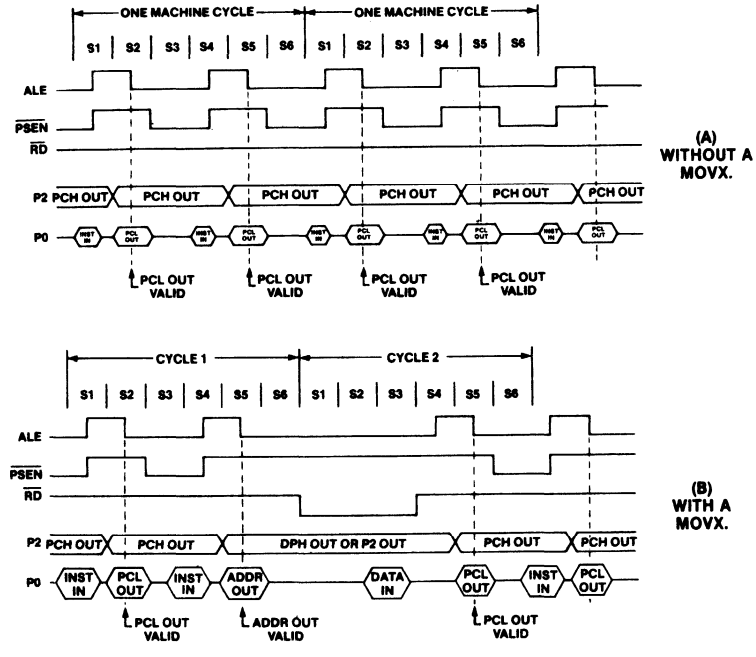


Fig. 3.13 Executing out of the external program memory.

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. 3.13). 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, each of which can be programmed to one of two priority levels. The five interrupt sources are listed below:

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.

Each source can be individually enabled or disabled by setting or clearing a bit in SFR IE. All interrupt sources can also be globally eabled or disabled.

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. 3.14.

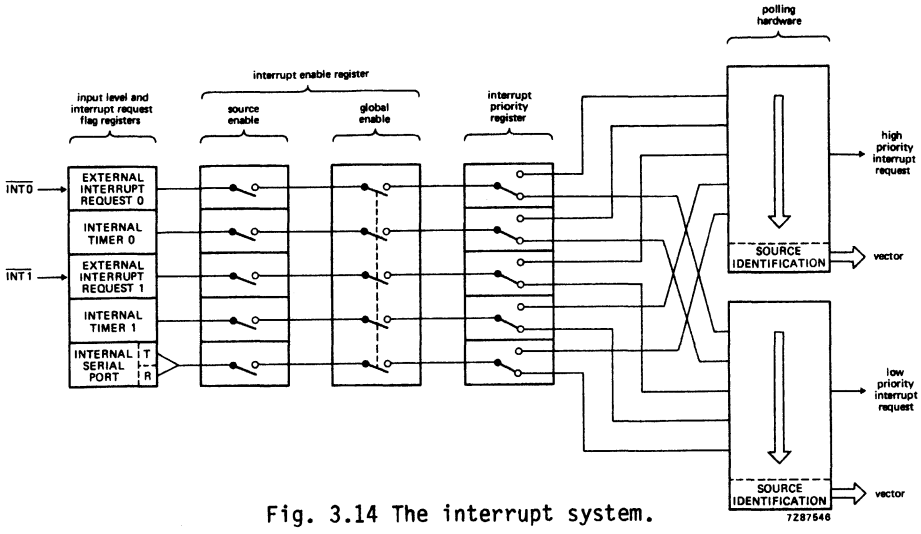


Fig. 3.14 The interrupt system.

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)

All the interrupt sources are examined sequentially during each cycle, such that by S6 of any cycle all active interrupt requests have been found and prioritized. Response to the active request of highest priority will commence with state 1 of the next machine cycle, provided the response 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 very next machine cycle.

The processor acknowledges a request by first setting the appropriate 'priority level active' flip-flop. It then executes a hardware subroutine call to the servicing routine and it also clears the flag that requested the interrupt (with the exceptions of $\overline{INT0}$ and $\overline{INT1}$, over whose sources the processor has no control, and TI and RI). The hardware subroutine call 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 request, as shown below:

Source	Address
External interrupt 0	0003H
Timer 0 overflow	000BH
External interrupt 1	0013H
Timer 1 overflow	001BH
Serial port	0023H

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 $\overline{INTx}$ pin and if ITx = 1, external interrupt x is edge-triggered. In the second case, if successive samples of the $\overline{INTx}$ 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. 3.15 shows the interrupt response timings.

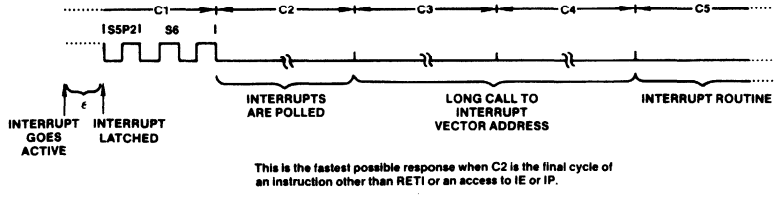


Fig. 3.15 The interrupt response timings.

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

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	X	ES	ET1	EX1	ET0	EX0

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.

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	X	PS	PT1	PX1	PT0	PX0

where

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 $\overline{\text{INT0}}$, 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 right 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, go back 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

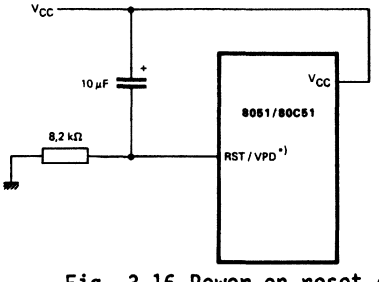
Reset is achieved by holding the RST pin HIGH for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset and it also configures the ALE and $\overline{\text{PSEN}}$ pins as inputs. (They are quasi-bidirectional.) The internal 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
IE	(0XX00000)	PCON	(0XXXXXXX)*

* (0XXX0000) for 80C51

The internal RAM is not affected by reset. When V_{CC} is turned on, the RAM content is indeterminate (see 3.11.43).

An automatic reset can be obtained when V_{CC} is turned on by connecting the RST pin to V_{CC} through a 10uF capacitor and V_{SS} through an 8.2kΩ resistor, provided that the V_{CC} risetime does not exceed one millisecond or so and that the oscillator start-up does not exceed 10 milliseconds. A power-on reset circuit is shown in Fig. 3.16. 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 is, 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 1ms, a 10µF capacitor will provide a reliable power-on reset.



* pin termed RST in the PCB80C51/80C31.

Fig. 3.16 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. 3.17. 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.

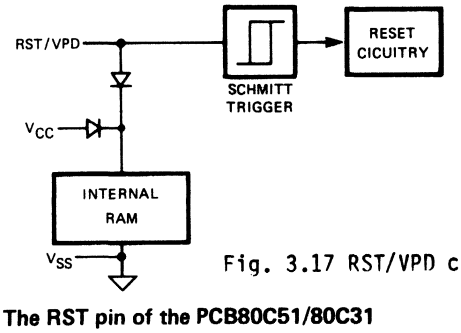


Fig. 3.17 RST/VPD circuitry

3.11.2 The RST pin of the PCB80C51/80C31

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

The nature of the internal RST pulldown 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.

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. 3.17, 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 INT0 or 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, V_{PD} 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. 3.18 suggests one possible implementation of the power down feature. Assuming imminent power failure detection interrupts the processor via INT0, the extenal 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 R, C, 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.

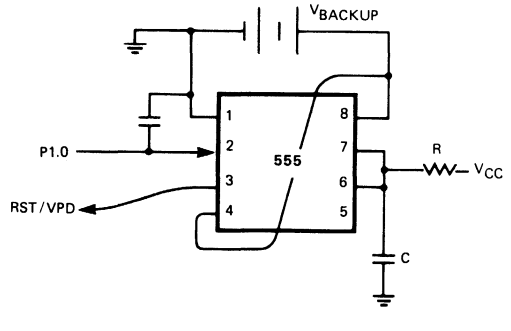


Fig. 3.18 Power down circuit.

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

Fig. 3.19 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.

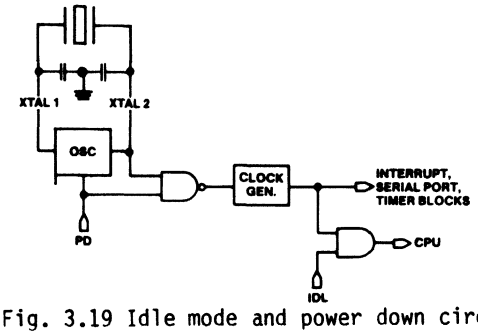


Fig. 3.19 Idle mode and power down circuitry.

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 in whatever mode the serial port is operating in. When SMOD is set the baud rate is doubled.

-: are 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 is turned off to the CPU, 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.

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 bits. On exit from the idle mode via an interrupt, the interrupt service routine can examine the flag bits.

The other way to exit 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 10ms).

3.13 Program verification

Fig. 3.20 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 2,5V 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., 10kΩ) are required on port 0 during this operation.

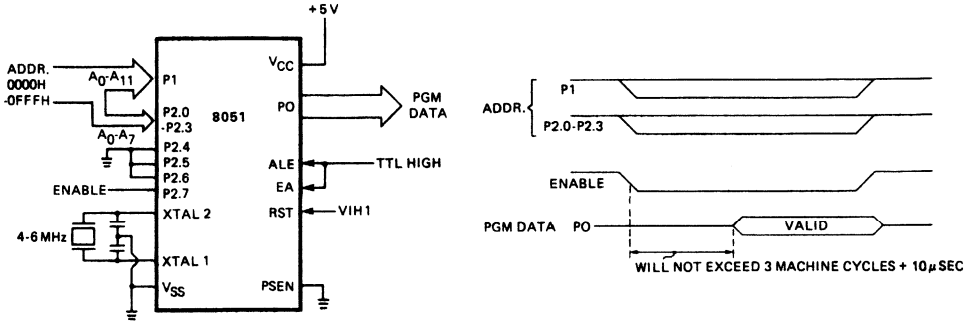


Fig. 3.20 Program verification.

4.0 MEMORY, ORGANIZATION, ADDRESSING MODES AND DATA MANIPULATION

4.1 Memory organization

In the 8051 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Fig. 3.1 and Fig. 3.2 are the:

- 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 64K addressing capabilities. The program counter allows 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 4K of the 64K program memory address space is filled by internal ROM. By tying the $\overline{EA}$ pin HIGH, the processor can be forced to fetch from the internal ROM for program memory addresses 0 to 4095. Bus expansion for accessing program memory from 4K upwards is automatic since external instruction fetches occur automatically when the program counter exceeds 4095. 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 are reserved for the five interrupt-request service routines. Each resource which can request an interrupt requires that its service routine is stored in its reserved location.

The 64K-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. In addition, 128 bit locations of the internal data RAM are accessible through direct addressing. These bits reside in the internal data RAM at byte 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, thanks 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.

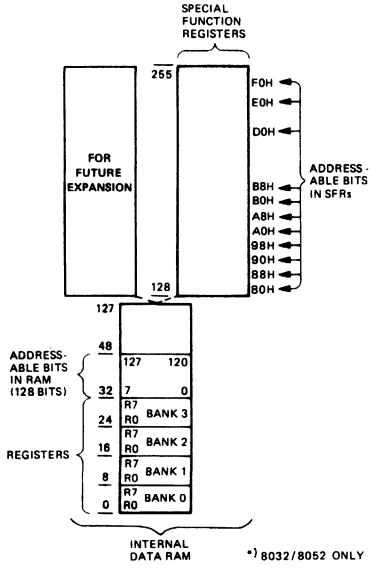


Fig. 4.1 Internal data memory address space.

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. In addition, 128 bit locations within the SFR address space can be accessed using direct addressing. These bits reside in the SFR byte addresses divisible by eight. The twenty SFRs are listed in Fig. 4.2 and their mapping in the SFR address space is shown in Fig. 4.3 and Fig. 4.4.

ARITHMETIC REGISTERS:
ACCumulator*, B register*,
Program Status Word*
POINTERS:
Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
Interrupt Priority Control*,
Interrupt Enable Control*
TIMERS:
Timer MODe, Timer CONtrol*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL I/O PORT:
Serial CONtrol*, Serial data BUFfer, PCON
***Bits in these registers are 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 is 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.

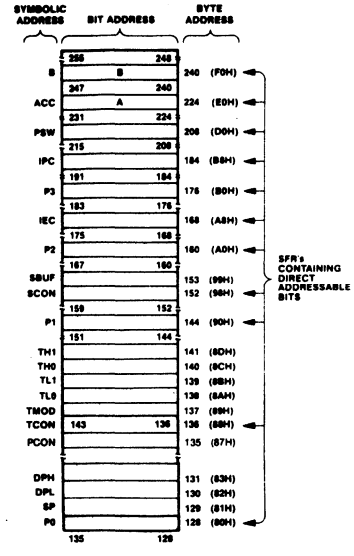


Fig. 4.3 Mapping of special function registers.

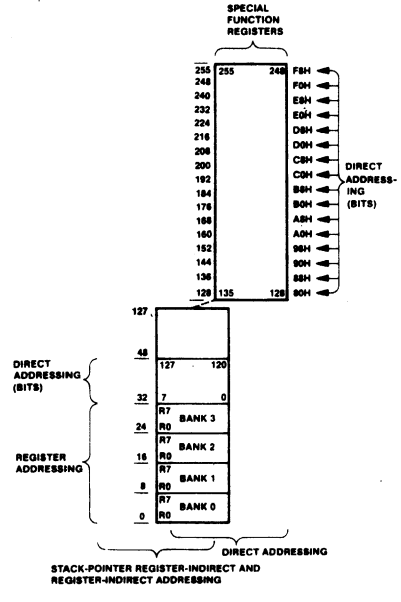


Fig. 4.4 Special function register bit address.

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.

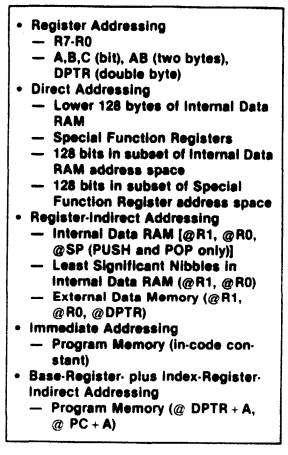


Fig. 4.5 Operand addressing modes.

4.2.1 Register addressing

Register addressing permits access to the eight registers (R_7-R_0) 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.

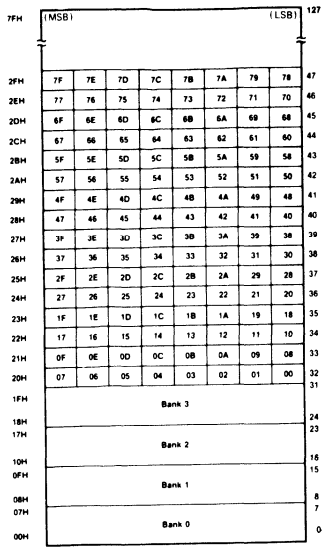


Fig. 4.6 RAM bit addresses.

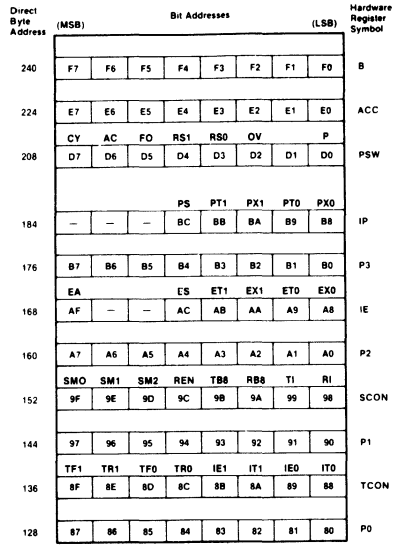


Fig. 4.7 Addressing operands in internal data memory.

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 additions to the internal data RAM from address 128 to 255 can only be addressed indirectly. This address area lies parrallel to the SFR range of addresses which are only accessed directly.

4.2.3 Register-indirect addressing

Register-indirect addressing using the content of R1 or R0 in the selected register bank, or using the content of the stack pointer (PUSH and POP only), addresses the internal data RAM. 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 64K external address space.

4.2.4 Immediate addressing

Immediate addressing allows 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 accessing 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 microcomputer is efficient both as a controller and as 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 & 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 periferals. 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.

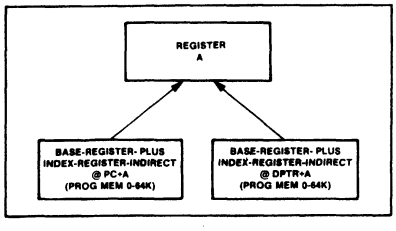


Fig. 4.8 Program memory move operations.

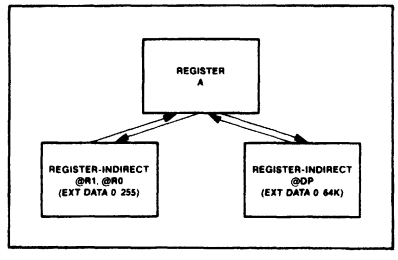


Fig. 4.9 External data memory move operations.

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 64K 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.

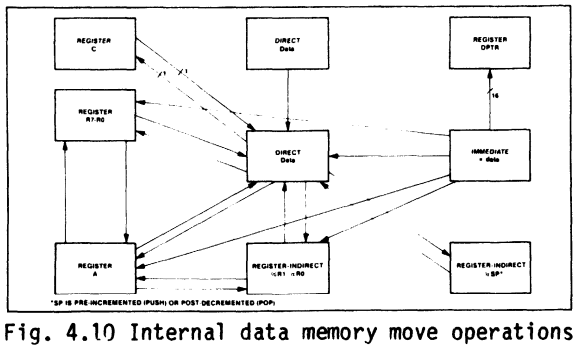


Fig. 4.10 Internal data memory move operations.

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.

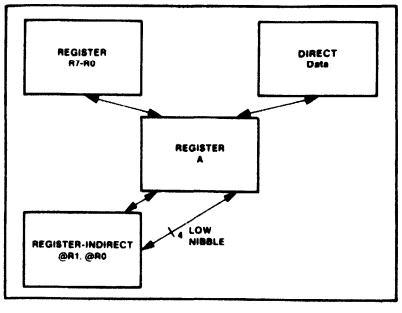


Fig. 4.11 Internal data memory exchange operations.

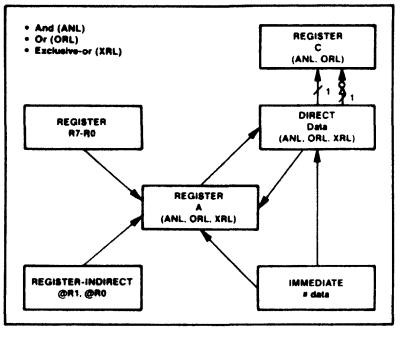


Fig. 4.12 Internal data memory logic operations.

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

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.

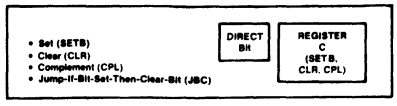


Fig. 4.13 Internal data memory logic operations (Register A specific).

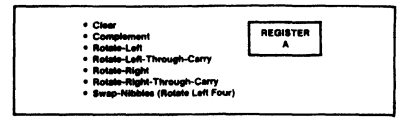


Fig. 4.14 Internal data memory logic operations (Bit-specific).

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.

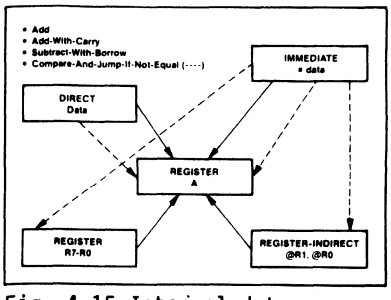


Fig. 4.15 Internal data memory arithmetic operations.

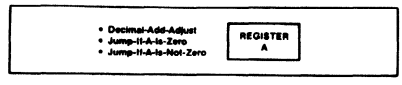


Fig. 4.16 Internal data memory arithmetic operations (Register A specific).

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 (MSB in B, LSB 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.

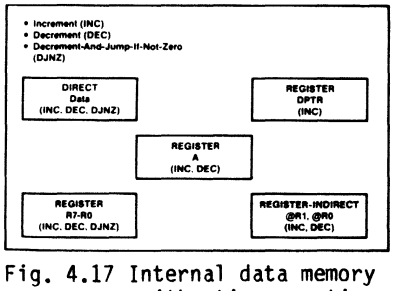


Fig. 4.17 Internal data memory arithmetic operations.

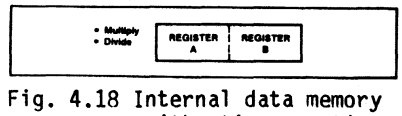


Fig. 4.18 Internal data memory arithmetic operations (Register A with B specific).

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 64K 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 2K program module. Unlike the 8048, the 8051's call operations do not push the PSW into 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.

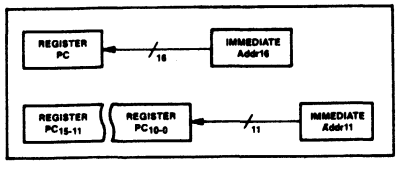


Fig. 4.19 Unconditional branch operations.

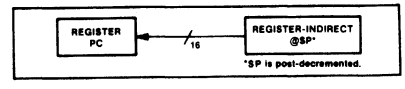


Fig. 4.20 Return operations.

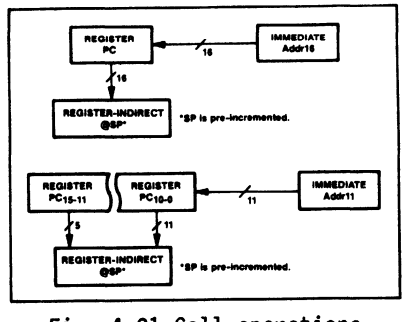


Fig. 4.21 Call operations.

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.

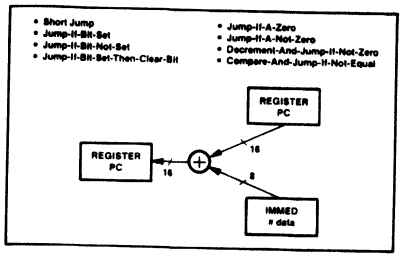


Fig. 4.22 Unconditional short branch and conditional branch operations.

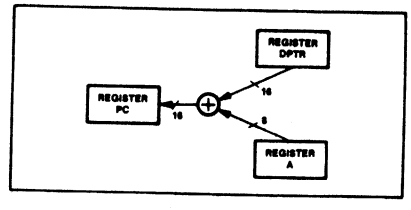


Fig. 4.23 Unconditional branch (indirect) operation.

5.0 INSTRUCTION SET OF THE 8051

The 8051 assembly language needs only forty-two mnemonics to specify the 8051's thirty-three 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 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 periferals 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 instructions execute in 12 oscillator periods, 45 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.

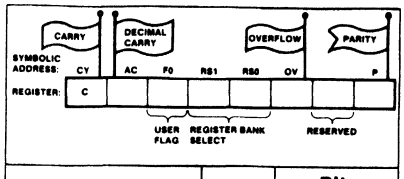
There are three two-operand operations provided:

ANL	performs the bitwise 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 bitwise logical inclusive-OR of two source operands (for both bit and byte operands) and returns to the location of the first operand.
XRL	performs the bitwise 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 (odd parity); 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.

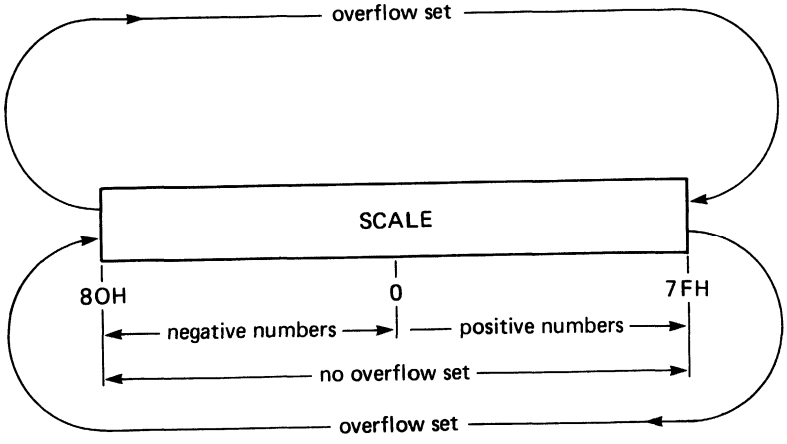


Fig. 5.2 The overflow bit.

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 operation 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 8048/8049/3050

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 addr11

Function: Absolute call

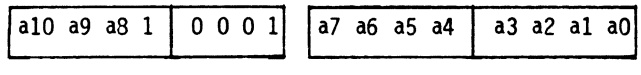
Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination 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 2K 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



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, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

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.

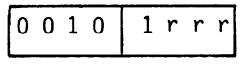
Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

```
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

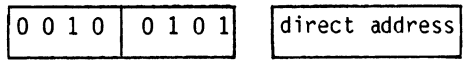
Bytes: 1
Cycles: 1



Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

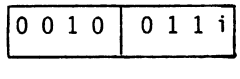
Bytes: 2
Cycles: 1



Operation: ADD
(A) ← (A) + (direct)

ADD A,@Ri

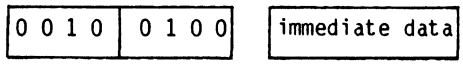
Bytes: 1
Cycles: 1



Operation: ADD
(A) ← (A) + ((Ri))

ADD A,#data

Bytes: 2
Cycles: 1



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, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

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.

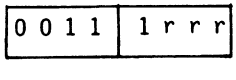
Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

```
ADDC  A,R0
```

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

ADDC A,Rn

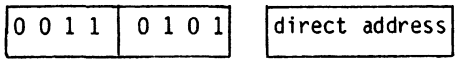
Bytes: 1
Cycles: 1



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

ADDC A,direct

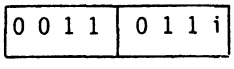
Bytes: 2
Cycles: 1



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

ADDC A,@Ri

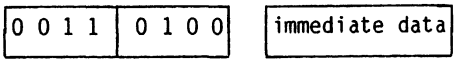
Bytes: 1
Cycles: 1



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

ADDC A,#data

Bytes: 2
Cycles: 1



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

AJMP addr11

Function: Absolute Jump

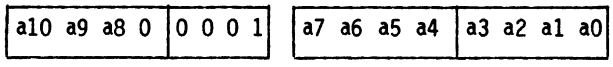
Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (AFTER incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block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



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 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,

```
ANL   A,R0
```

will leave 41H (01000001B) in the accumulator.

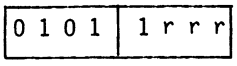
When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time.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

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

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

ANL A,Rn

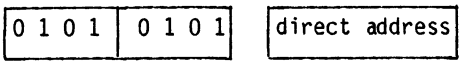
Bytes: 1
Cycles: 1



Operation: ANL
(A) ← (A) ∧ (Rn)

ANL A,direct

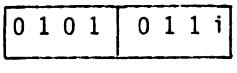
Bytes: 2
Cycles: 1



Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

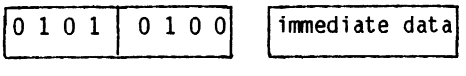
Bytes: 1
Cycles: 1



Operation: ANL
(A) ← (A) ∧ ((Ri))

ANL A,#data

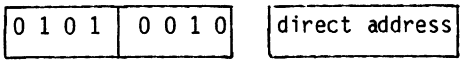
Bytes: 2
Cycles: 1



Operation: ANL
(A) ← (A) ∧ #data

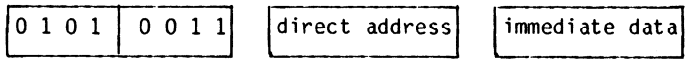
ANL direct,A

Bytes: 2
Cycles: 1



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

ANL direct,#data
Bytes: 3
Cycles: 2



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

ANL C,/src-bit>

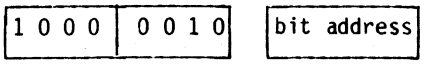
Function: Logical-AND for bit variables

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
```

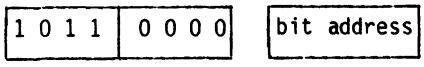
ANL C,bit
Bytes: 2
Cycles: 2



Operation: ANL
(C) ← (C) ^(bit)

ANL C,/bit

Bytes: 2
Cycles: 2



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 the [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
```

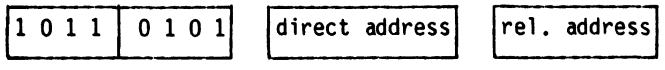
sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

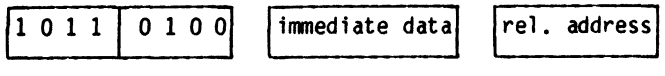
clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H).

CJNE A,direct,rel
Bytes: 3
Cyles: 2



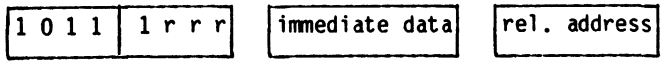
Operation:
```
CJNE
(PC) ← (PC) + 3
IF (direct) < (A)
      THEN (PC) ← (PC) + rel and (C) ← 0
      OR
IF (direct) > (A)
      THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE A,#data,rel
Bytes: 3
Cycles: 2



Operation:
```
CJNE
(PC) -- (PC) + 3
IF  #data < (A)
      THEN (PC) ← (PC) + rel and (C) ← 0
      OR
IF  #data > (A)
      THEN (PC) ← (PC) + rel and (C) ← 1
```

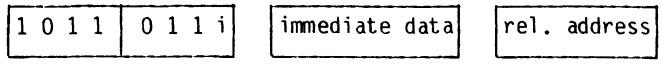
CJNE Rn,#data,rel
Bytes: 3
Cyles: 2



Operation:
```
CJNE
(PC) ← (PC) + 3
IF #data < (Rn)
      THEN (PC) ← (PC) + rel and (C) ← 0
      OR
IF #data > (Rn)
      THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE @Ri,#data,rel

Bytes: 3
Cyles: 2



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

CLR A

Function: Clear Accumulator

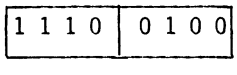
Description: The accumulator is cleared (all bits set to zero). No flags are affected.

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

CLR A

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

Bytes: 1
Cycles 1



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.

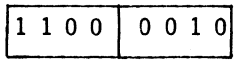
Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

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

CLR C

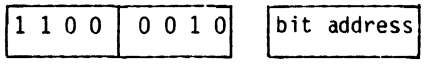
Bytes: 1
Cycles: 1



Operation: CLR
(C) ← 0

CLR bit

Bytes: 2
Cycles: 1



Operation: CLR
(bit) ← 0

CPL A

Function: Complement Accumulator

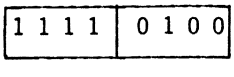
Description: Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.

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

CPL A

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

Bytes: 1
Cycles: 1



Operation: CPL
(A) ← ¬ (A)

CPL bit

Function: Complement bit

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

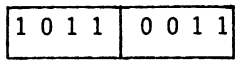
Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

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

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

CPL C

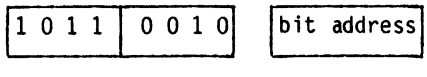
Bytes: 1
Cyles: 1



Operation: CPL
(C) ← ¬ (C)

CPL bit

Bytes: 2
Cycles: 1



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 proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

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 proper 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 wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the 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** simply convert a hexadecimal number in the acumulator 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 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 to mean 30 - 1 = 29.

Bytes: 1
Cyles: 1

1 1 0 1	0 1 0 0

Operation:

DA
-contents of Accumulator are BCD
IF $[[(A_{3-0})>9] \vee [(AC) = 1]]$
THEN $(A_{3-0}) \leftarrow (A_{3-0}) + 5$
AND
IF $[[(A_{7-4})< 9] \vee [(C) = 1]]$
THEN $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

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.

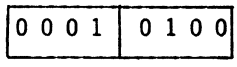
Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectivley. The instruction sequence,

```
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.

DEC A

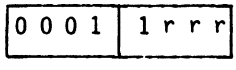
Bytes: 1
Cycles: 1



Operation: DEC
(A) ← (A) - 1

DEC Rn

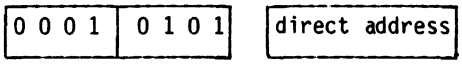
Bytes: 1
Cycles: 1



Operation: DEC
(Rn) ← (Rn) - 1

DEC direct

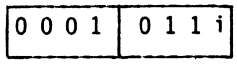
Bytes: 2
Cycles: 1



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

DEC @Ri

Bytes: 1
Cycles: 1



Operation: DEC
((Ri)) ← ((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer 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.

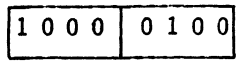
Exception: if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

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



Operation: DIV

$(A)_{15-8} \leftarrow (A) / (B)$

$(B)_{7-0}$

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.

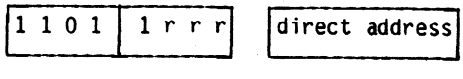
This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
          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 port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2
Cycles: 2

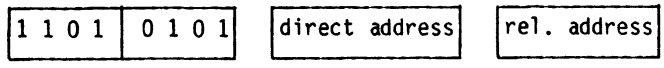


Operation:

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

DJNZ direct,rel

Bytes: 3
Cycles: 2



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

INC byte

Function: Increment

Description: INC increments the indicated variable by 1. An original value of OFFH will overflow to OOH. 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.

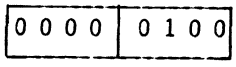
Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain OFFH and 40H, respectively. The instruction sequence,

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

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

INC A

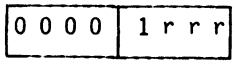
Bytes: 1
Cycles: 1



Operation: INC
(A) ← (A) + 1

INC Rn

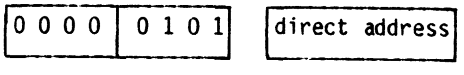
Bytes: 1
Cycles: 1



Operation: INC
(Rn) ← (Rn) + 1

INC direct

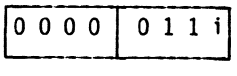
Bytes: 2
Cycles: 1



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

INC @Ri

Bytes: 1
Cycles: 1



Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

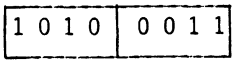
Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo-2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from OFFH to 00H will increment the high-order byte (DPH). No flags are affected.

Example: This is the only 16-bit register which can be incremented. 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



Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if Bit set

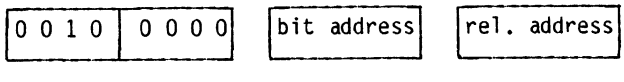
Description: If the indicated bit is a one, jump to the address indicated, otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction.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

Example: The data present at input port 1 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



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 one, branch to the address indicated; otherwise proceed with the next instruction. **In either case, clear the designated 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.

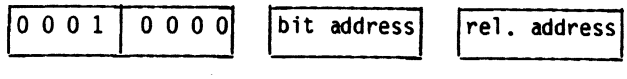
Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, **not** the input pin.

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



Operation:

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

JC rel

Function: Jump if Carry is set

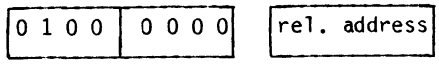
Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example:

```
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



Operation:

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

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting 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 sequences of instruction 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. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1
Cycles: 2

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 zero, 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

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

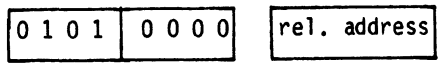
Description: If the carry flag is zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry 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



Operation:

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

JNZ rel

Function: Jump if Accumulator Not Zero

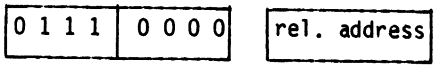
Description: If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is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

Example: The accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2
Cycles: 2



Operation:

```
JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0
      THEN (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

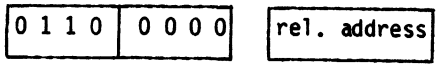
Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator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

Example: The accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC   A
JZ    LABEL2
```

will change the accumulator to 00H and cause program execution to continue as the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2



Operation: JZ
(PC) ← (PC) + 1
IF (A) = 0
THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long Call

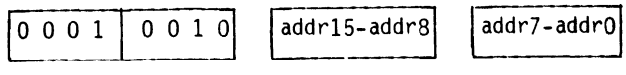
Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low-order byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-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



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

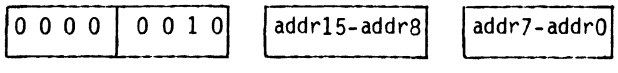
Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

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



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 by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

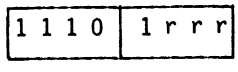
Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
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 both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

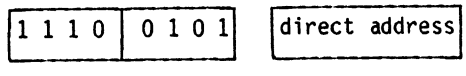
Bytes: 1
Cycles: 1



Operation: MOV
(A) ← (Rn)

MOV A,direct

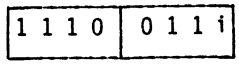
Bytes: 2
Cycles: 1



Operation: MOV
(A) ← (direct)

MOV A,@Ri

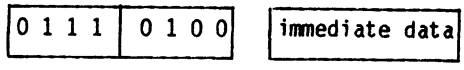
Bytes: 1
Cycles: 1



Operation: MOV
(A) ← ((Ri))

MOV A,#data

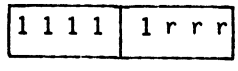
Bytes: 2
Cycles: 1



Operation: MOV
(A) ← #data

MOV Rn,A

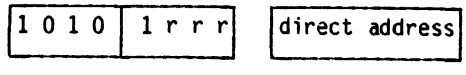
Bytes: 1
Cycles: 1



Operation: MOV
(Rn) ← (A)

MOV Rn,direct

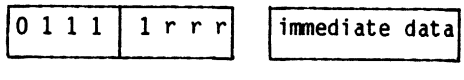
Bytes: 2
Cycles: 2



Operation: MOV
(Rn) ← (direct)

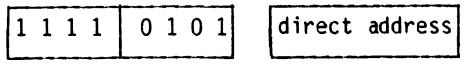
MOV Rn,#data

Bytes: 2
Cycles: 1



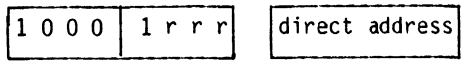
Operation: MOV
(Rn) ← #data

MOV direct,A
Bytes: 2
Cycles: 1



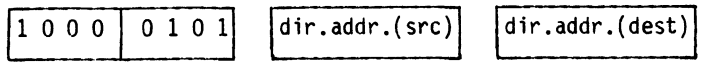
Operation: MOV
(direct) ← (A)

MOV direct,Rn
Bytes: 2
Cycles: 2



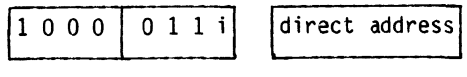
Operation: MOV
(direct) ← (Rn)

MOV direct,direct
Bytes: 3
Cycles: 2



Operation: MOV
(direct) ← (direct)

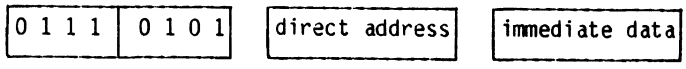
MOV direct,@Ri
Bytes: 2
Cycles: 2



Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

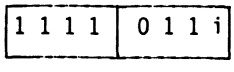
Bytes: 3
Cycles: 2



Operation: MOV
(direct) ← #data

MOV @Ri,A

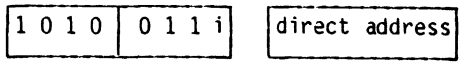
Bytes: 1
Cycles: 1



Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

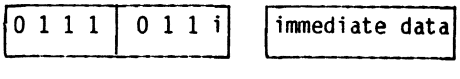
Bytes: 2
Cycles: 2



Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2
Cycles: 1



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.

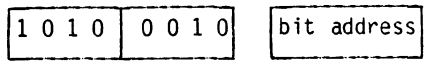
Example: The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B).

```
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

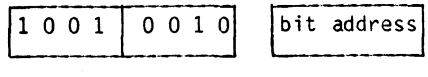
Bytes: 2
Cycles: 1



Operation: MOV
(C)← (bit)

MOV bit,C

Bytes: 2
Cycles: 2



Operation: MOV
(bit)← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

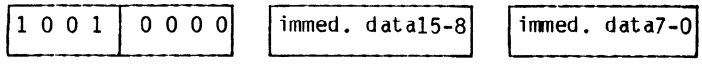
Description: The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

Example: This is the only instruction which moves 16 bits of data at once. 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



Operation: MOV
$(DPTR) \leftarrow \#data_{15-0}$
$DPH \square DPL \leftarrow \#data_{15-8} \square \#data_{7-0}$

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 case, 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 a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

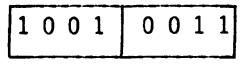
Example: A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
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

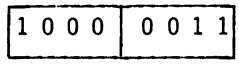
Bytes: 1
Cycles: 2



Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A+PC

Bytes: 1
Cycles: 2



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, differing in whether they provide an eight-bit 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 somewhat 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 64K 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 the code to output high-order address lines driven by 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 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Register 0 and 1 contain 12H and 34H. Location 34H of the external RAM hold 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

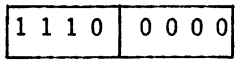
Bytes: 1
Cycles: 2

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

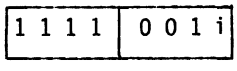
Bytes: 1
Cycles: 2



Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

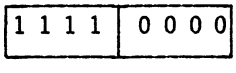
Bytes: 1
Cycles: 2



Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1
Cycles: 2



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 (OFFH) 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

1 0 1 0	0 1 0 0

Opertion: 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: It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. 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

0 0 0 0	0 0 0 0

Operation: NOP
$(PC) \leftarrow (PC) + 1$

ORL <dest-byte><src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise 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 adress, 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.

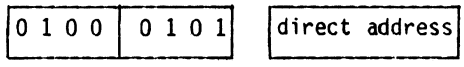
ORL A,Rn

Bytes: 1
Cycles: 1

0 1 0 0	1 r r r

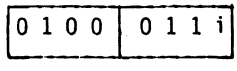
Operation: ORL
(A) ← (A) ∨ (Rn)

ORL A,direct
Bytes: 2
Cycles: 1



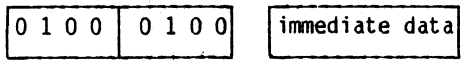
Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri
Bytes: 1
Cycles: 1



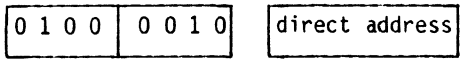
Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data
Bytes: 2
Cycles: 1



Operation: ORL
(A) ← (A) ∨ #data

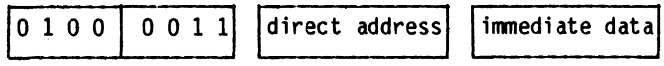
ORL direct,A
Bytes: 2
Cycles: 1



Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3
Cycles: 2



Operation: ORL
(direct) ← (direct) ∨ #data

ORL C,<src-bit>

Function: Logical-OR for bit variables

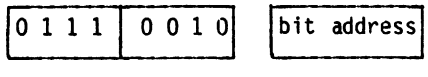
Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. 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 P10
ORL   C,ACC.7         ;OR CARRY WITH THE ACC. BIT 7
ORL   C,/OV           ;OR CARRY WITH THE INVERSE OF OV
```

ORL C,bit

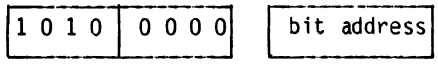
Bytes: 2
Cycles: 2



Operation: ORL
(C) ← (C) ∨ (bit)

ORL C,/bit

Bytes: 2
Cycles: 2



Operation: ORL
$(C) \leftarrow (C) \vee (\overline{bit})$

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected.

Example: The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H and 01H respectively. The instruction sequence,

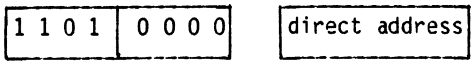
```
POP   DPH
POP   DPL
```

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



Operation: POP
(direct) $\leftarrow$ ((SP))
(SP) $\leftarrow$ (SP) - 1

PUSH direct

Function: Push onto stack

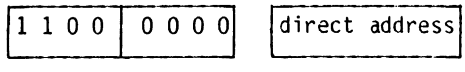
Description: The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

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 0BH, respectively.

Bytes: 2
Cycles: 2



Operation: PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

RET

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

0 0 1 0	0 0 1 0

Operation: RET
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RETI

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 just 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 interruption is processed.

Example: The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H..

Bytes: 1
Cycles: 2

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

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

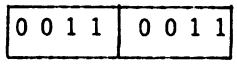
Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

```
RLC   A
```

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

Bytes: 1
Cycles: 1



Operation: RLC
(An + 1) ← (An) n = 0 - 6
(A0) ← (C)
(C) ← (A7)

RR A

Function: Rotate accumulator Right

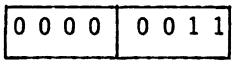
Description: The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

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



Operation: RR
(An) ← (A_{n+1}) n = 0 - 6
(A7) ← (A0)

RRC A

Function: Rotate accumulator Right through Carry flag

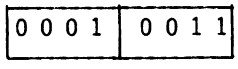
Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

```
RRC   A
```

leaves the accumulator holding the value 62 (01100010B) with the carry set.

Bytes: 1
Cycles: 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.

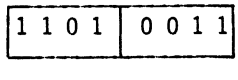
Example: The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instruction,

```
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

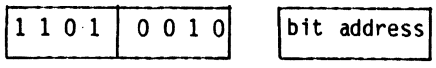
Bytes: 1
Cycles: 1



Operation: SETB
$(C) \leftarrow 1$

SETB bit

Bytes: 2
Cycles: 1



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.

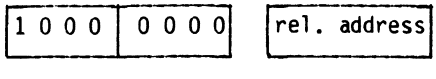
Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

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. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop).

Bytes: 2
Cycles: 2



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 into bit 6, but not into bit 7, or into 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,

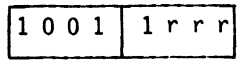
```
SUBB  A,R2
```

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice 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 explicitly cleared by a CLR instruction.

SUBB A,Rn

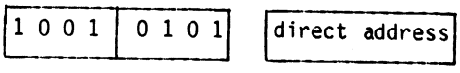
Bytes: 1
Cycles: 1



Operation: SUBB
(A) ← (A) - (C) - (Rn)

SUBB A,direct

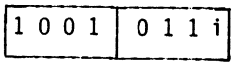
Bytes: 2
Cycles: 1



Operation: SUBB
(A) ← (A) - (C) - (direct)

SUBB A,@Ri

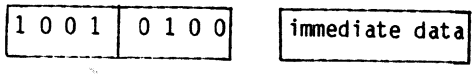
Bytes: 1
Cycles: 1



Operation: SUBB
(A) ← (A) - (C) - ((Ri))

SUBB A,#data

Bytes: 2
Cycles: 1



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 thought of 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

1 1 0 0	0 1 0 0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

XCH A, byte

Function: Exchange 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

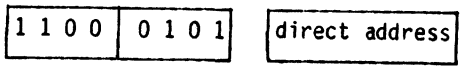
Bytes: 1
Cycles: 1

1 1 0 0	1 r r r

Operation: XCH
$(A) \rightleftarrows (Rn)$

XCH A,direct

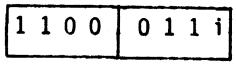
Bytes: 2
Cycles: 1



Operation: XCH
(A) $\rightleftarrows$ (direct)

XCH A,@Ri

Bytes: 1
Cycles: 1



Operation: XCH
(A) $\rightleftarrows$ ((Ri))

XCHD A,@Ri

Function: Exchange Digit

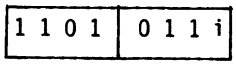
Description: XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

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



Operation: XCHD
(A_{3-0}) $\rightleftarrows$ (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. 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).

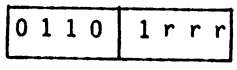
When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
XRL   P1,#00110001B
```

will complement bits 5, 4 and 0 of output port 1.

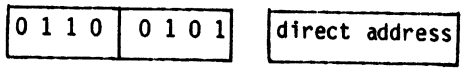
XRL A,Rn

Bytes: 1
Cycles: 1



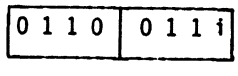
Operation: XRL
(A) ← (A) ⊻ (Rn)

XRL A,direct
Bytes: 2
Cycles: 1



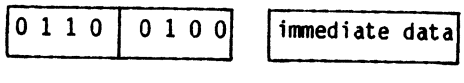
Operation: XRL
(A) ← (A) ⊻ (direct)

XRL A,@Ri
Bytes: 1
Cycles: 1



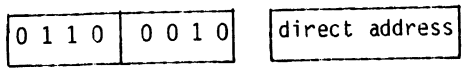
Operation: XRL
(A) ← (A) ⊻ ((Ri))

XRL A,#data
Bytes: 2
Cycles: 1



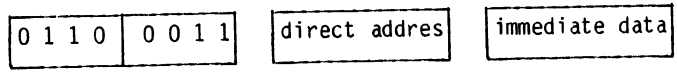
Operation: XRL
(A) ← (A) ⊻ #data

XRL direct,A
Bytes: 2
Cycles: 1



Operation: XRL
(direct) ← (direct) ∨ (A)

XRL direct,#data
Bytes: 3
Cycles: 2



Operation: XRL
(direct) ← (direct) ∨ #data

6.0 APPLICATION EXAMPLES FOR THE 8051/80C51 FAMILY OF MICROCONTROLLERS

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 arithmetic

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.

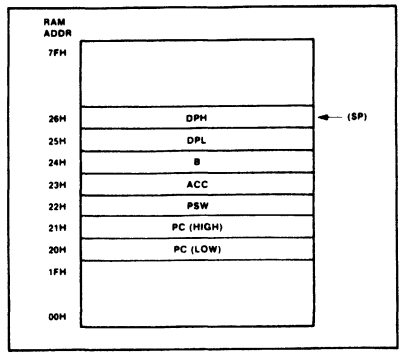


Fig. 6.1 The stack contents during an interrupt.

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 rare situations, 128 options are insufficient, the destination routines may cross a 2K 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 stackpointer 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.

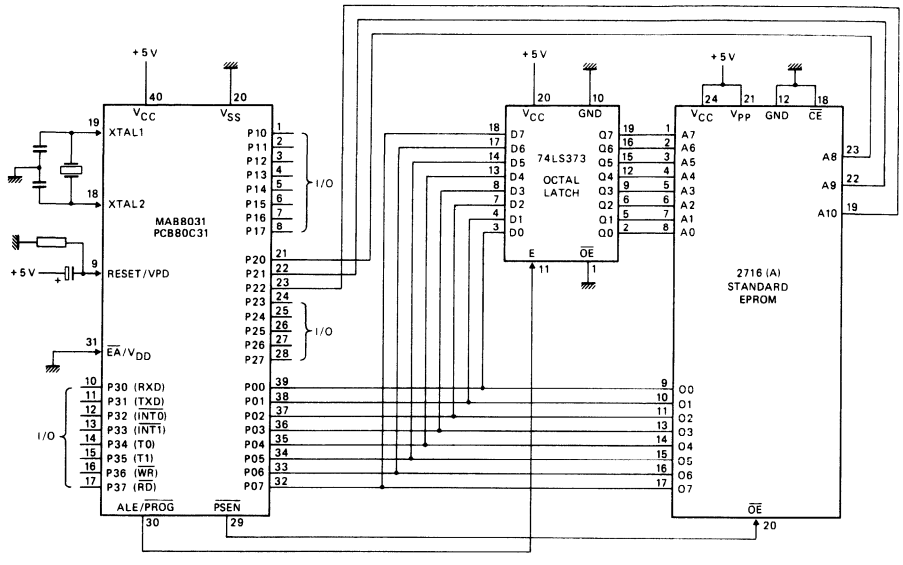


Fig. 6.3 External program memory using a standard EPROM.

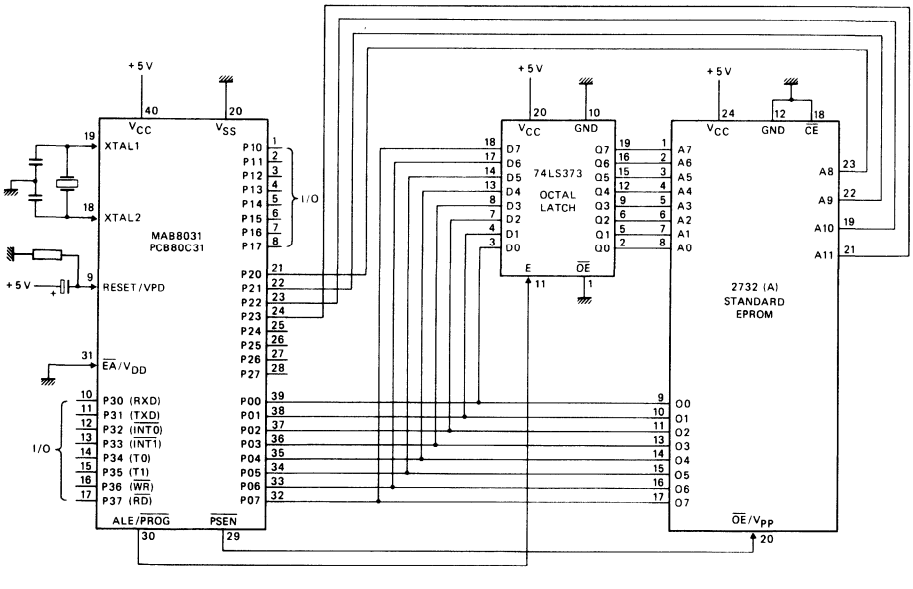


Fig. 6.4 External program memory using a standard EPROM.

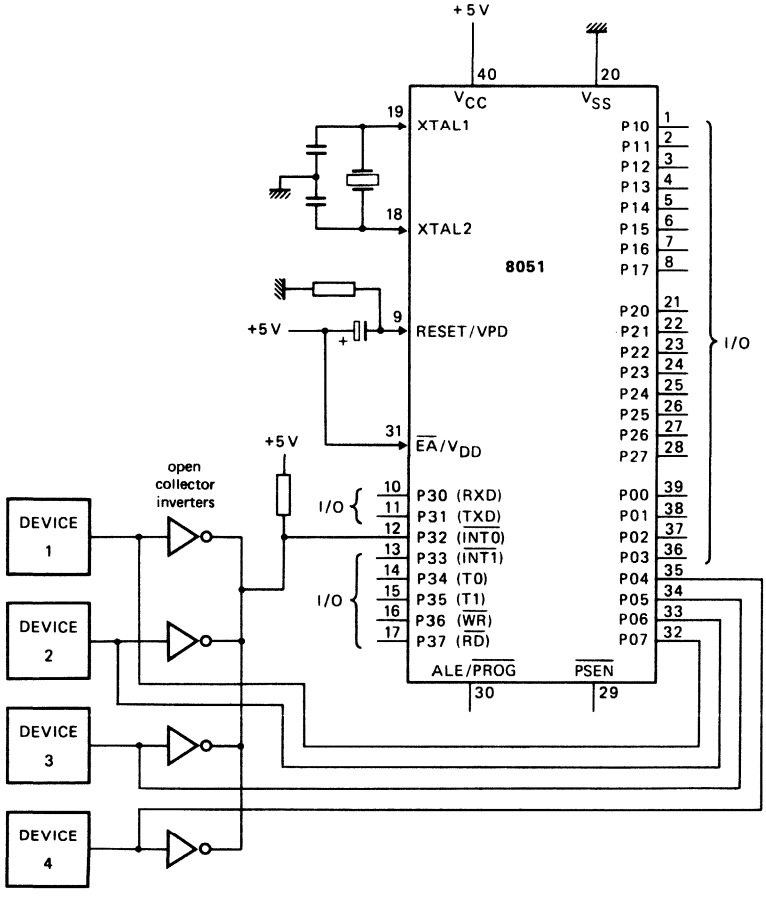


Fig. 6.5 Multiple interrupt sources.

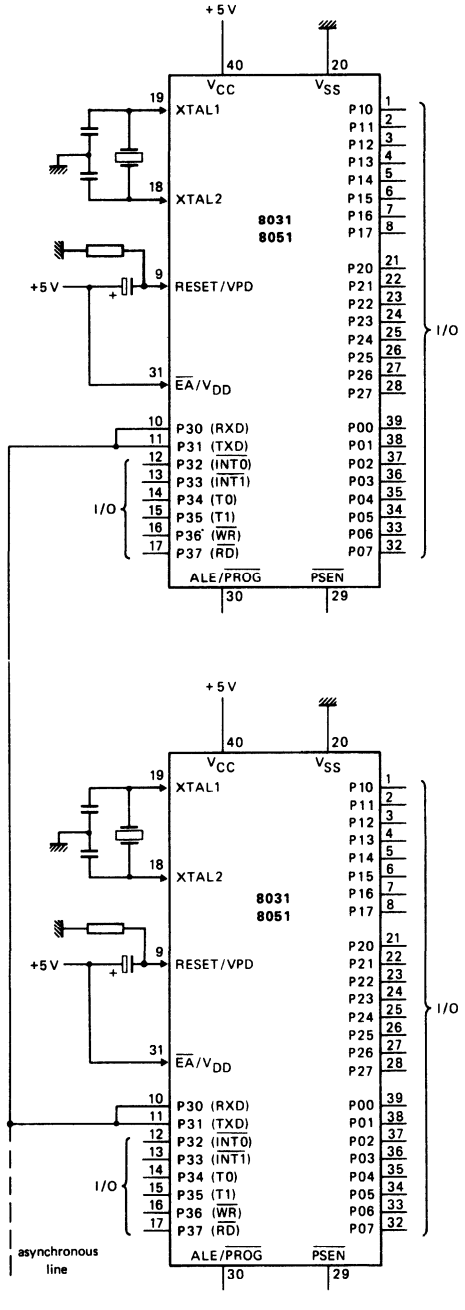


Fig. 6.6 Multiple 8051's using half-duplex serial communication.

4. The MAB84XX microcontroller family

CONTENTS – MAB84XX MICROCONTROLLER FAMILY

1.0 DESCRIPTION

The MAB84XX family of single-chip 8-bit microcontrollers are manufactured in N-MOS. The family consists of 6 devices:

- MAB8400 : NO ROM/128 RAM bytes
- MAB8401 : MAB8400 plus 8-Bit LED-driver (10 mA)
- MAB/F8411 : 1K ROM/64 RAM bytes plus 8-bit LED driver
- MAB/F8421 : 2K ROM/64 RAM bytes plus 8-bit LED-driver
- MAB/F8441 : 4K ROM/128 RAM bytes plus 8-bit LED-driver
- MAB/F8461 : 6K 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 8400/01)
- 1K, 2K, 4K or 6K 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 (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 ($\pm$ 10%)
- Operating temperature range:

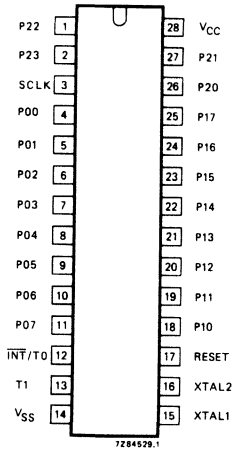
0 to + 70°C	MAB84X1 family
-40 to + 85°C	MAF84X1 family
-40 to + 110°C	MAF84AX1 family

extended temp. range not available for MAB8400 and MAB8401

3.0 PACKAGE OUTLINES

MAB/8400/01B : 28-lead "Piggy-back" package (with 28-lead EPROM on top)
MAB8400/01WP : 68-lead, plastic leaded chip carrier (PLCC); SOT-188
MAB/F8411/21/41/61P: 28-lead DIL; plastic (SOT-117D)
MAB8411/21/41/61: 28-lead DIL; plastic (SO-28;SOT-136A)

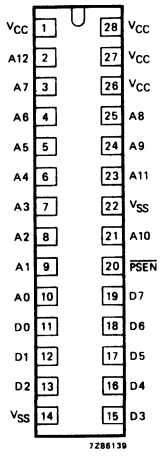
3.1 Figure 1. PIN DESIGNATION MAB8411/21/41/61 also MAB8400/01B "piggyback" bottom pinning. (For top pinning see figure 2).



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 & 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 MAB8400/01B (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 MAB8400/01B '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 1µS

Note: 1) A 2732 EPROM has to be inserted with pin 1 in the pin 3 location of the MAB84XXB top pinning
2) A 2716 EPROM may be used if its Vpp pin (pin 23 on MAB8400/01B top pinning) is disconnected from A11 and connected to V_{CC}.

When emulating 84XX family software using the MAB8400/01B, 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 MAB84XX/01WP

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 MAB84XX family, including emulation facilities for the MAB/F8422/42 (20-pin version). It can address 8K bytes of ROM. The RAM has 128 bytes.

The PLCC (plastic leaded chip carrier) is designed for use in SMD (surface mounted device) circuitry. Figure 3. shows the pad designation.

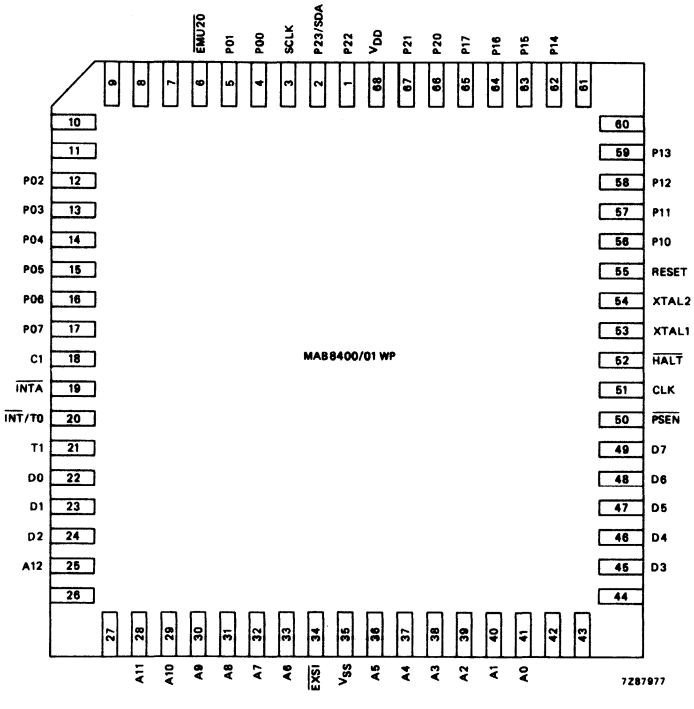


Fig. 3 MAB8400/01WP PLCC pad designation

CHIP CARRIER DESIGNATION

4.1 Pad Designation

Designation	pad no	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 to a negative going edge), testable using JT0 or JNT0 instructions.
T1	21	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 (84XX 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
CLK	51	Clock output buffered from XTAL2

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 INTA active, the address bus shows the address that has been saved in the stack (return address) the 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). 8400 not connected.

The diagram below Figure 3(a), shows the connection of EPROM to 'piggy-back' package MAB8400/01B.

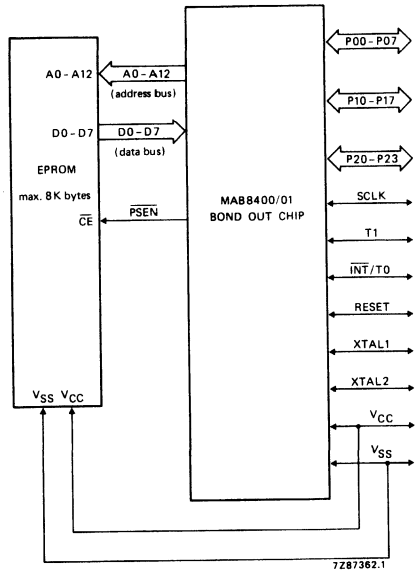


Fig. 3(a) connection of EPROM to MAB8400/01B

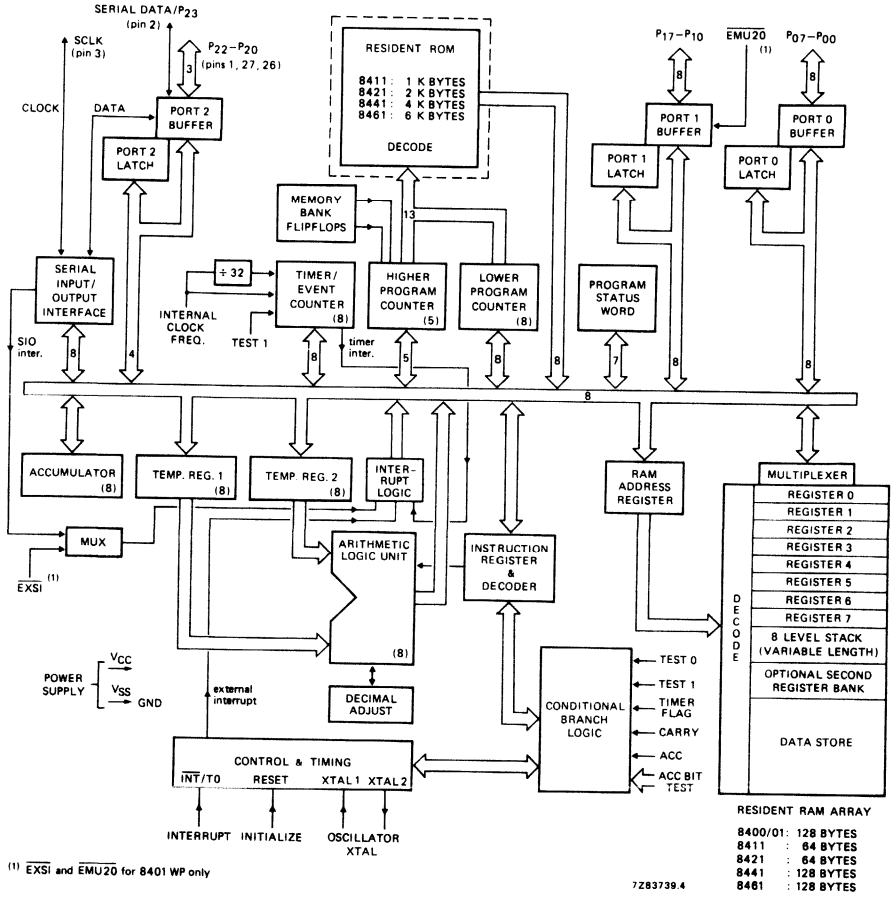
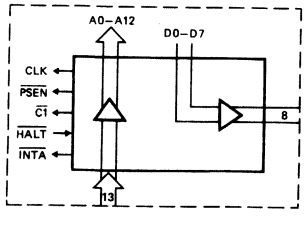
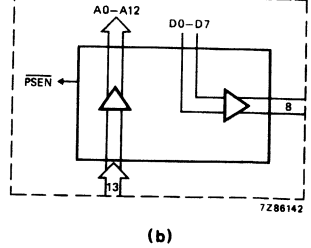


Fig. 4a Block diagram of the MAB84XX family.



(a)

Fig. 4b Replacement of dotted part in fig. 1a showing the MAB8400/01WP bond-out version



(b)

Fig. 4c Replacement of dotted part in Fig. 1a for MAB8400/01B ('Piggy-back').

5.0 FUNCTIONAL DESCRIPTION

5.1 Program memory

The program memory consists of 1024, 2048 or 4096 bytes (8-bit words), which are addressed by the program counter. The memory is mask-programmed at production. Because the MAB84XX 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.

Figures 5 & 6 show the program memory and data memory maps. Figure 7 illustrates the structure of the program counter stack.

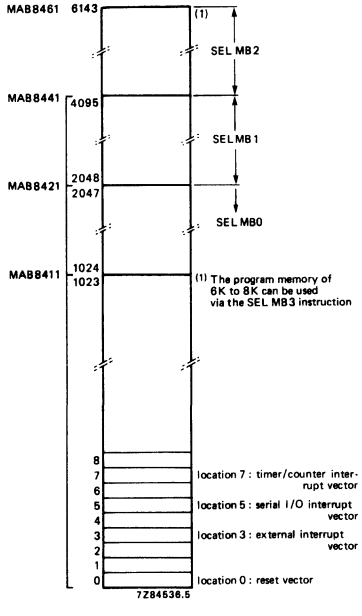


Fig. 5 The program memory map

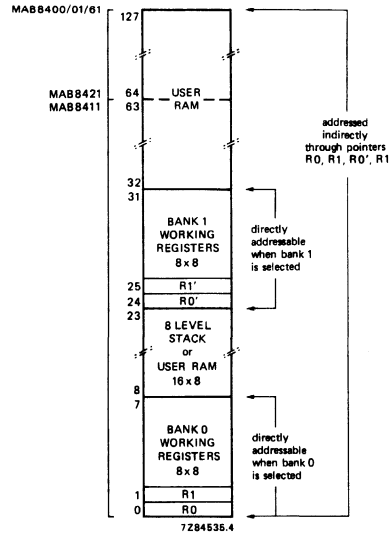


Fig. 6 The data memory map

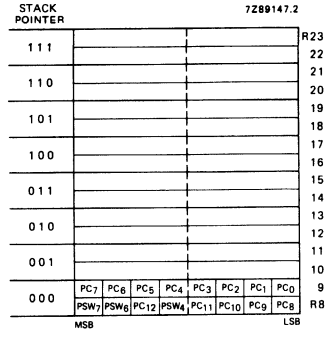


Fig. 7 Program counter stack

5.3 Input/Output

The MAB84XX family has 23 I/O lines arranged as:

- two parallel ports of 8 lines (P00 to P07, P10 to P17)
- 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 *. Figure 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.

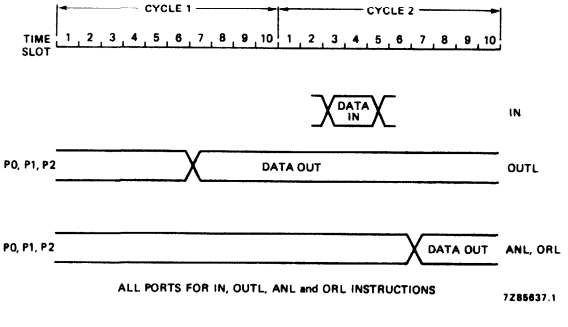


Fig.8 Timing diagram of all ports on IN and OUTL instructions; for ANL and ORL instructions, the ports change on the time slot 7 of cycle 2.

* Port 1 can sink up to 8 LED's.

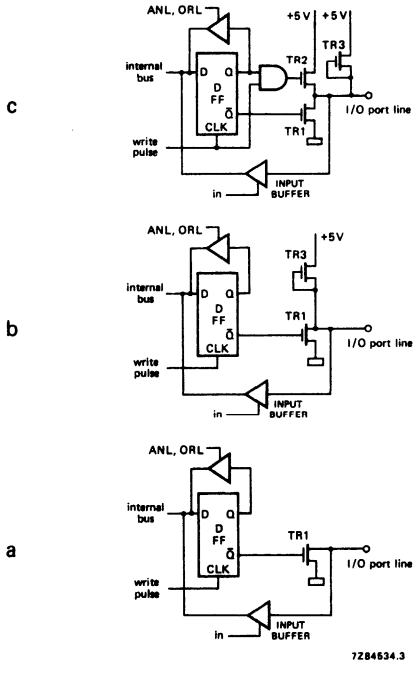


Fig. 9 Quasi-bidirectional I/O interface with port options:

(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 MAB8400/01 B/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 MAB84XX family serial I/O system allows any number of MAB84XX family devices to be interconnected by the two-line I^2C serial bus. The interface lends the ability to any two 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 it's 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 seperate 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 $\overline{INT}$/T0 input initiates an external interrupt subroutine which causes a call to program memory location 3 following completion of the current instruction. Serial I/O interrupt, when enabled, causes a call to location 5, and a timer/ event counter overflow a call to location 7, when the interrupt is enabled.

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 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 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 MAB84XX is active on the negative edge; a HIGH to LOW transition on the $\overline{INT}$/T0 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 flag '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 $\overline{INT}$ 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.

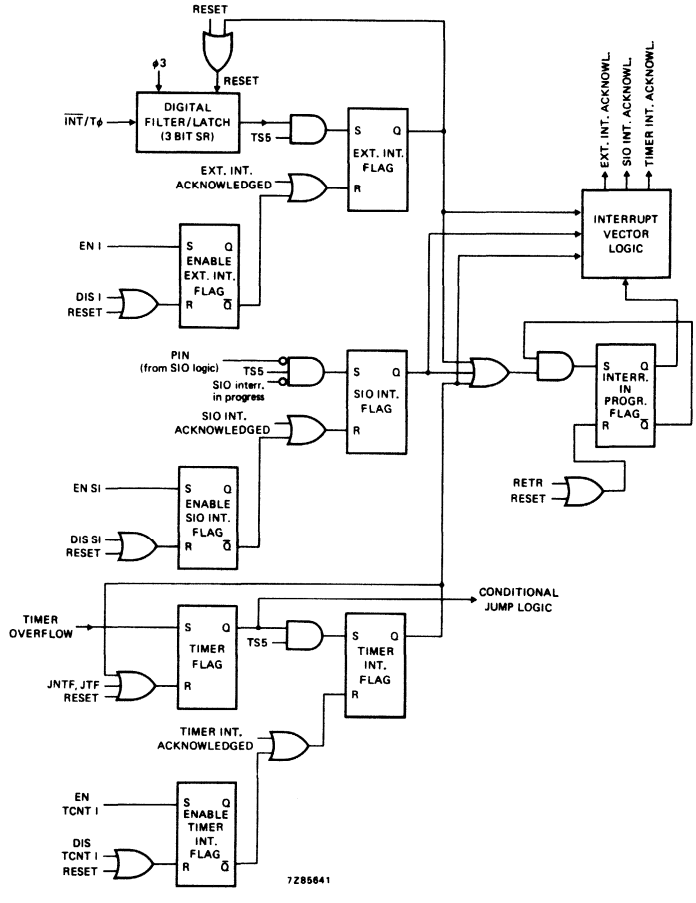


Fig. 10 Interrupt logic

Note:
1. INT/TO negative edge is always latched in the digital filter/latch.
2. Interrupt is ensured when INT/TO 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. Within a timer/counter interrupt routine, the timer flag will not respond to timer overflows.

5.4.2 Interrupt program examples

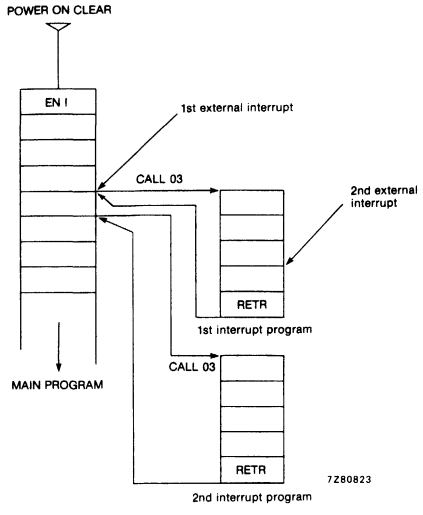


Fig. 11 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

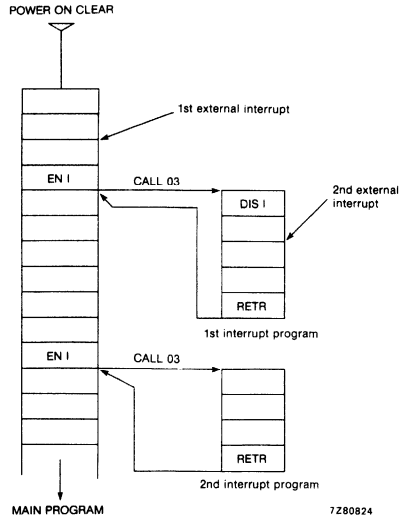


Fig. 12 Second external interr -upt is postponed until enabled again

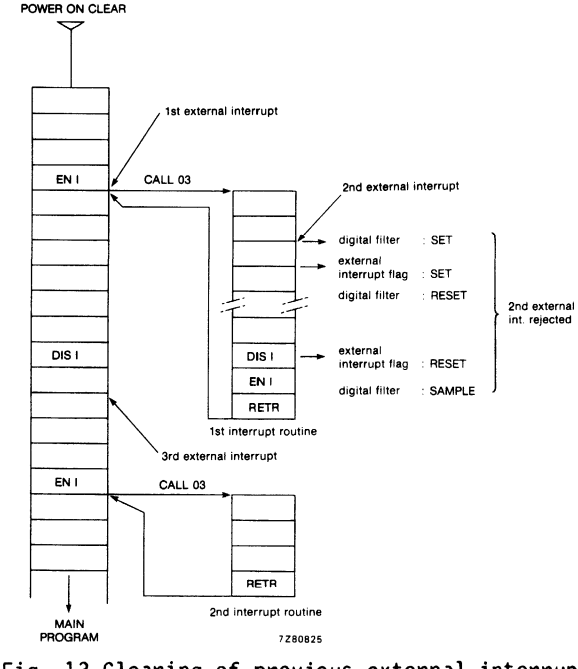


Fig. 13 Clearing of previous external interrupt

5.4.2 Interrupt program examples (continued)

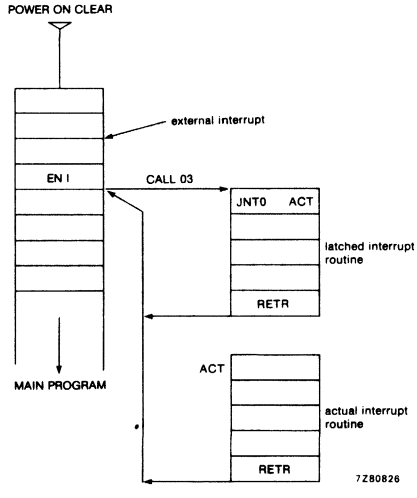


Fig. 14 Detection of previous and actual external interrupt.

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: MAB8400B/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.

1 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:

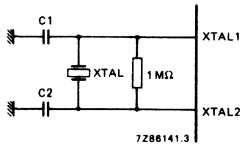
- an external interrupt
- an input level that may be tested by the conditional jump instructions JTO and JNTO.

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 it's negative going edge. The interrupt facility is discussed in more detail in chapter 5.4.

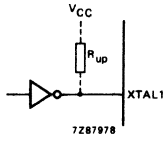
5.7 Oscillator and clock

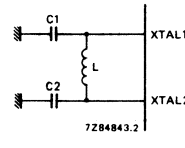
A crystal, ceramic resonator, inductor or resistor connected between XTAL1 and XTAL2 (see Fig. 15) determines the frequency of the internal oscillator. An externally generated clock signal can also be applied to XTAL1 as the frequency reference. A machine cycle consists of 10 states, each state being 3 oscillator periods. A 6 MHz crystal gives a 5,0 μs machine cycle.

The MAB84XX family has dynamic logic; for adequate refreshing the oscillator frequency must be at least 1 MHz.



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 7.0. See also user manual section on Clock Pulse Generation.

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 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.

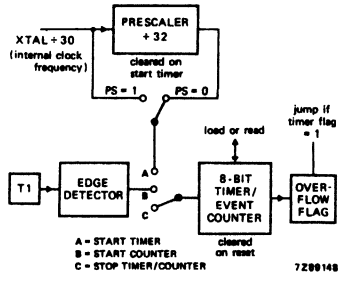


Fig. 16 Timer event counter.

5.9 Program status word

The program status word (PSW) is an 8-bit word (1byte) 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(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.

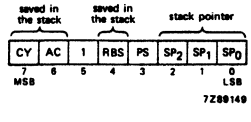


Fig. 17 Program status word

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. In the case of an interrupt. Bits 7, 6 and 4 are stored in the program counter stack during subroutines and interrupt calls. These bits are restored in the PSW (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.

5.10 Program counter

A 13-bit program counter is used so that up to 8K bytes of ROM can be addressed. Figure 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.

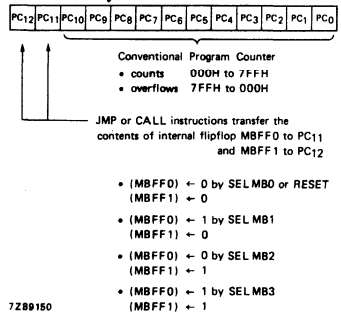


Fig. 18 Program counter.

5.11 Central processing unit

The MAB84XX 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 condition
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.

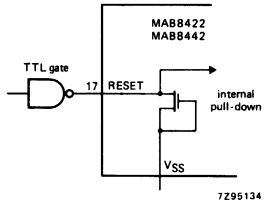


Fig. 19 External reset.

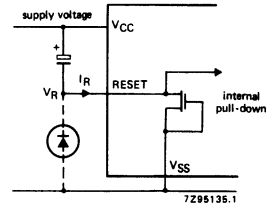


Fig. 20(a) Power-on reset.

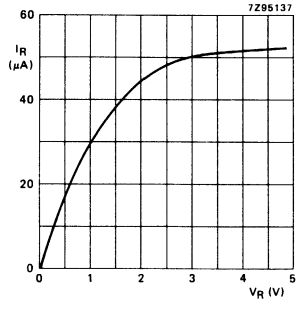


Fig. 20(b) Typical input characteristics

5.13 Instruction set

The MAB84XX 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 6 shows the instruction set of the MAB84XX family; Table 3 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 4 shows the MAB84XX family instructions (including the five instructions for serial I/O operation) absent from the MAB8048 instruction set.

Table 4

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 PO,A
		SEL MB2
		SEL MB3
			JNTF addr

Table 5 shows the MAB8048 instructions absent from the MAB84XX family instruction set

Table 5

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 MAB84XX family:

	8021	8048	84XX family
ROM capacity(bytes)	1K	1K	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(µS)	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 6 Instruction set is shown on the next 5 pages.

	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	$(A_{n+1})\leftarrow A_n$ $(A_0)\leftarrow(C)$, $(C)\leftarrow(A_7)$ 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 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 99 9A	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 89 8A	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)	9
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 9D 9E	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.
9. Only for software-transfer from the MAB8021.

* : 8, 9, A, B, C, D, E, F
● : 0, 2, 4, 6, 8, A, C, E
▲ : 1, 3, 5, 7, 9, B, D, F

MAB84XX 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 PO,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 MAB8400/01WP $\overline{\text{HALT}}$ 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. Figure 21 shows the timing cycle of the $\overline{\text{HALT}}$ function.While stopped, the address of the next instruction to be fetched is available on the address bus.

The MAB8400/01WP 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.

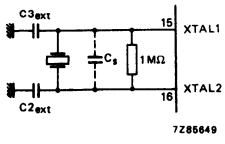
3. Each following cycle that the $\overline{\text{HALT}}$ line is sampled active, a NOP is inserted in the instruction register (IR).

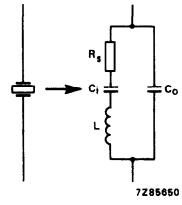
4. The processor continues if the $\overline{\text{HALT}}$ input is found HIGH during TS4-TS9, i.e. instead of NOP, the next instruction is fetched and inserted in IR.

7.0 Timing

7.1 Clock adjustment

Quartz crystal adjustment





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)

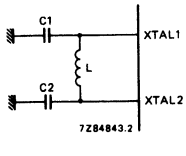
Series resistance: $\leqslant$ 60 Ω

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



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

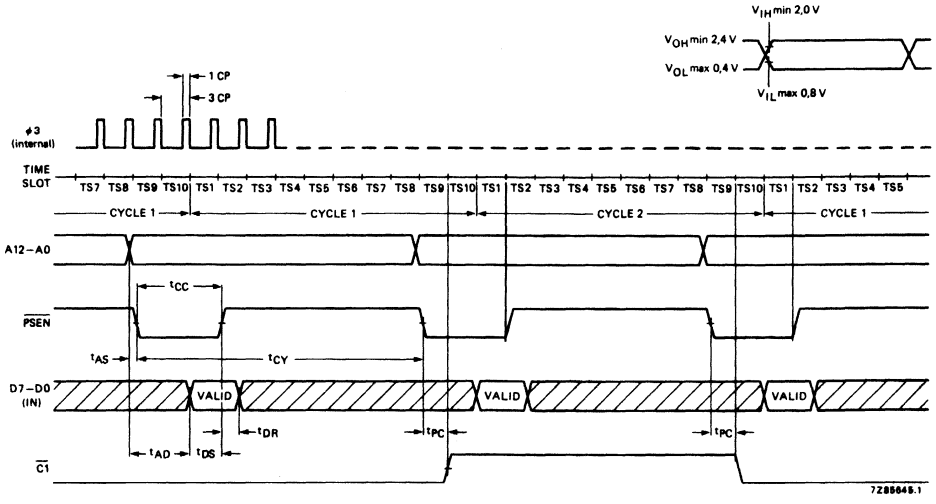


Fig. 20(a) I/O voltage parameters, (b) memory access timing MAB8400/01WP

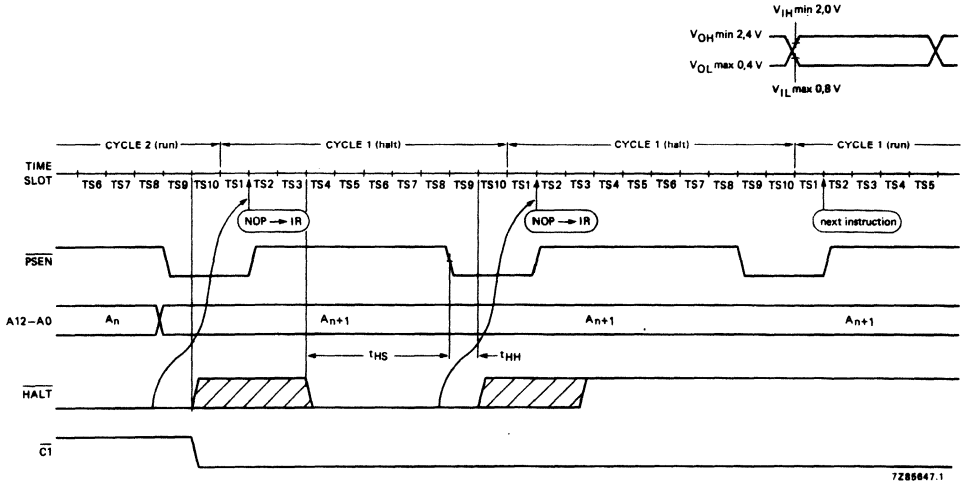


Fig. 21(a) I/O voltage parameters, (b) INTA and HALT timing MAB8400/01WP

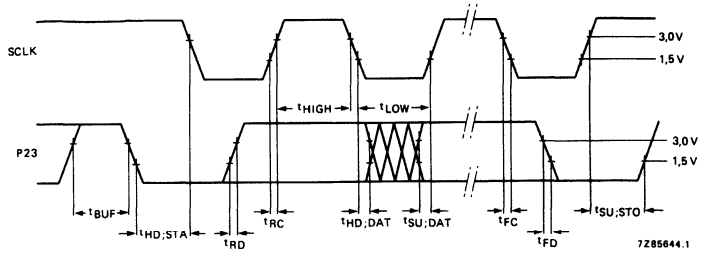


Fig. 22 Input timing.

7.2 Timing requirements for the P23 and SCLK input signals

t_{BUF}	$\geqslant 14t_{XTAL}$	t_{HIGH}		$\frac{1}{2}(DF+3)\,t_{XTAL}$
$t_{HD;STA}$	$\geqslant 14t_{XTAL}$	t_{LOW}		$\frac{1}{2}(DF-3)\,t_{XTAL}$
t_{HIGH}	$\geqslant 17t_{XTAL}$	$t_{SU;STO}$		$\frac{1}{2}(DF-3)\,t_{XTAL}$
t_{LOW}	$\geqslant 17t_{XTAL}$	$t_{HD;DAT}$	(only for	$\geqslant 6\,t_{XTAL}$
$t_{SU;STO}$	$\geqslant 14t_{XTAL}$		SLAVE TRANSMITTER)	$\leqslant 12\,t_{XTAL}$
$t_{HD;DAT}$	> 0	$t_{SU;DAT}$	(only for	$6\,t_{XTAL}$
$t_{SU;DAT}$	$\geqslant 250$ ns		MASTER TRANSMITTER)	
t_{RD}	$\leqslant 1$ µs	t_{AC}		$\geqslant 6\,t_{XTAL}$
t_{RC}	$\leqslant 1$ µs			$\leqslant 12\,t_{XTAL}$
t_{FD}	$\leqslant 1$ µs	t_{FD}, t_{FC}		$\leqslant 100$ ns at $C_b = 400$ pF*
t_{FC}	$\leqslant 1$ µs			

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 ± 0,5 MHz.

7.3 Timing requirements for the P23 and SCLK output signals

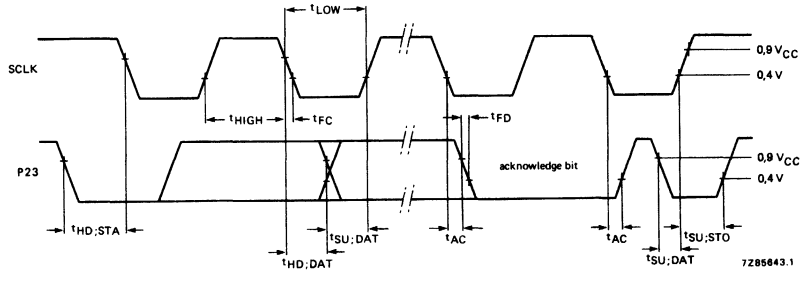


Fig.23 Output timing

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 MAB84XX family)

* C_b is maximum bus capacitance for each line.

8.0 84XX MICROCONTROLLER FAMILY DEVELOPMENT SYSTEM

8.1 PMDS 2 microcomputer development system

A complete development system PMDS 2, available from Philips T & M suppliers, supports a wide range of microcontrollers and microprocessors. PMDS 2 provides an enviroment for both software and hardware development. Hard-ware debugging and the all important software/hardware integration phase of the development cycle are managed by a powerful real-time operating system.

Because PMDS 2 is a universal development system, the only hardware that needs to be exchanged is a unit known as the Microcomputer adapter box MAB. The MAB is the hardware unit that acts as an interface between the PMDS and the users prototype. It is connected to the PMDS via two flat cables and the emulation probe, the emulation probe replaces the target microcontroller in the prototype system. In addition to an MAB, a software debugger and cross-assembler are required for full emulation.

When using PMDS 2 to emulate the 84XX family, the complete support system consists of a:-

- PM 8367 Cross-assembler 8411, 8421, 8441, 8461
- PM 8327 Microcomputer adapter box MAB 8411, 8421, 8441, 8461
- PM 8387 Debugger 8411, 8421, 8441, 8461

8.2 PM4300 microcomputer instructor

An alternative development system for the 84XX family is the Philips PM4300 microcomputer instructor. The PM4300 is a low-cost, universal prototyping and debugging tool. Microcontrollers/Microprocessors supported by the PM4300 include the: 8086, Z8002, Z80, 8048, 8049, 8088 and the MAB 84XX.

Changing from one micro to another is achieved by simply plugging in a new uP/uC personality module.

The personality module contains a bond-out version of the target micro. Signals are brought out through the μP/μC emulation cable -a flat cable whose plug matches the pin-out of the test micro. This cable can be plugged into the uP/uC socket of the prototype system. The PM4300 may be used for real-time execution, single-stepping of programs in the test system enviroment, and to check hardware and software integration.

Detailed information on Philips microcontroller development systems are contained in another chapter of this manual, the Development support section .

5. The MAB8422/42 microcontroller family

CONTENTS – THE MAB8422/42 MICROCONTROLLER FAMILY

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 - 2K ROM/64 RAM bytes
- MAB8442 - 4K 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.

2.0 FEATURES

- 8-bit: CPU, ROM, RAM and I/O.
- 20 pin package
- MAB8422 : 2K ROM/64 RAM bytes
- MAB8442 : 4K ROM/128 RAM bytes
- 13 quasi-bidirectional I/O port lines
- Two testable inputs T1 and INT/T0
- High current output on P0
- One external interrupt line combined with the testable input line INT/T0
- Single-level interrupts: external, timer/event counter, 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

Fig. 2.1 shows the block diagram of the MAB8422/8442. Both microcontrollers are in 20 pin DIL packages. The pin configuration is given in Fig. 2.2 with the designation of the pins in Table 1.

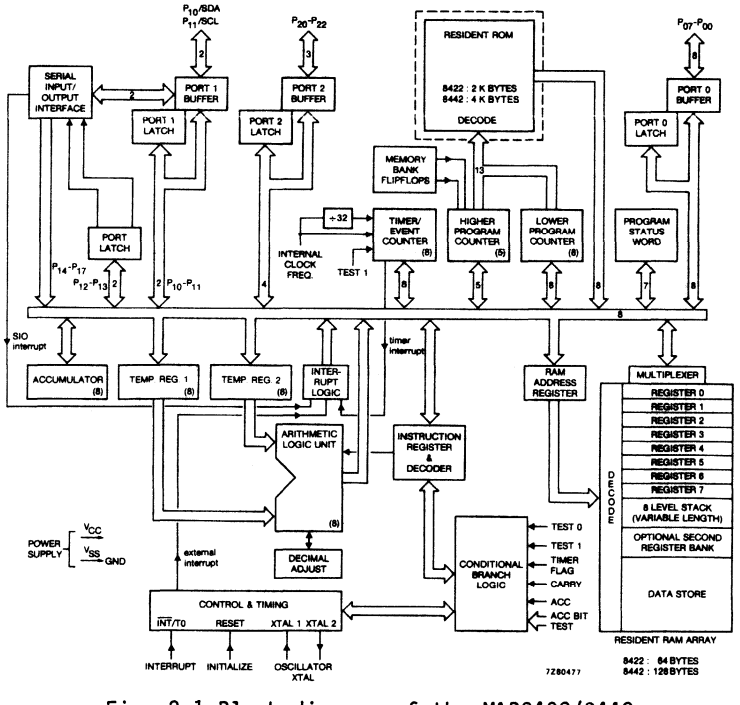


Fig. 2.1 Block diagram of the MAB8422/8442.

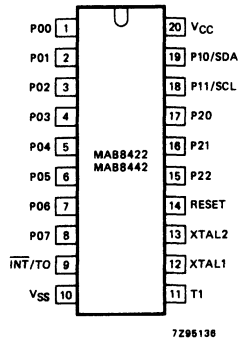


Fig. 2.2 MAB8422/8442 Pinning diagram

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).
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.

3.0 THE MEMORY

3.1 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 long so that up to 8 Kbytes 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. 3.1.

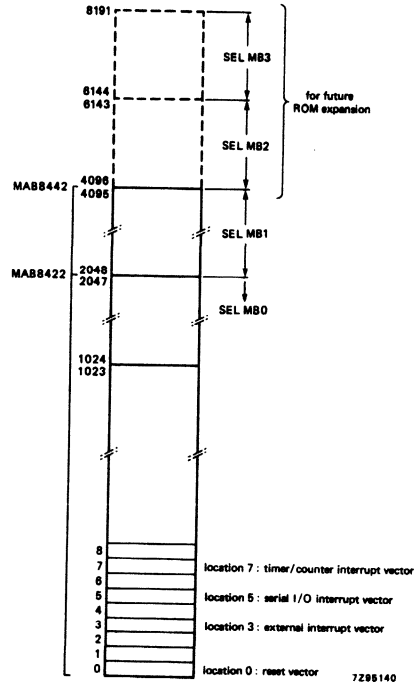


Fig. 3.1 The program memory map.

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 2K 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 wanted, 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 located.

3.2 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. 3.2 shows the data memory map.

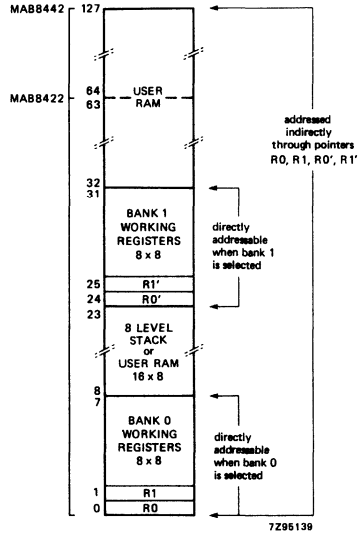


Fig. 3.2 The data memory map.

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. 3.3) 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 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.

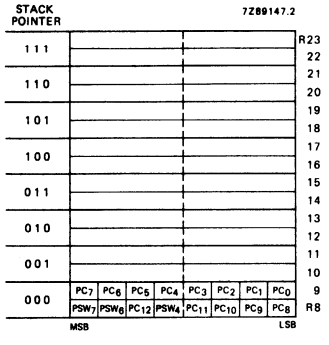


Fig. 3.3 Program counter stack.

4.0 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-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 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.

4.1 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 4.1(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 TR4. 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 CMOS or TTL circuits. The I/O electrical characteristics of P10/SDA and P11/SCL are I^2C compatible.

Two other interface output configurations can also be specified.

- open drain output with pull-up (Fig. 4.2(b))
- open drain output without pull-up (Fig. 4.2(c))

Port 0 and port 2 can be be specified in the three configurations. Port 1 is an open drain output without pull-up.

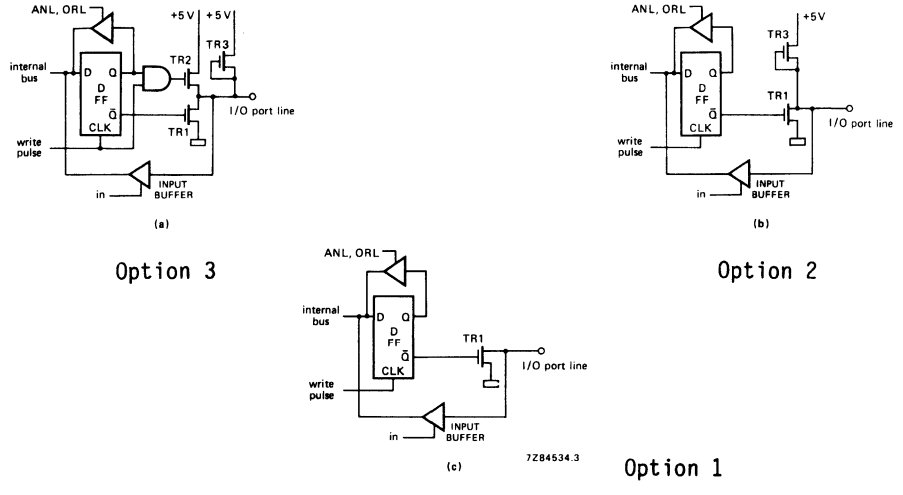


Fig 4.1 Quasi-bidirectional I/O interface with (a) push-pull output with pull-up; (b) open drain output with pull-up; (c) open drain output without pull-up.

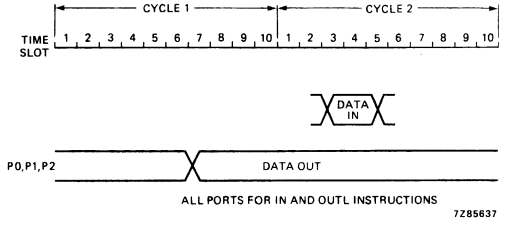


Fig. 4.2 Timing diagram; for ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

Fig. 4.3 gives an overview of the port signals in the parallel mode.

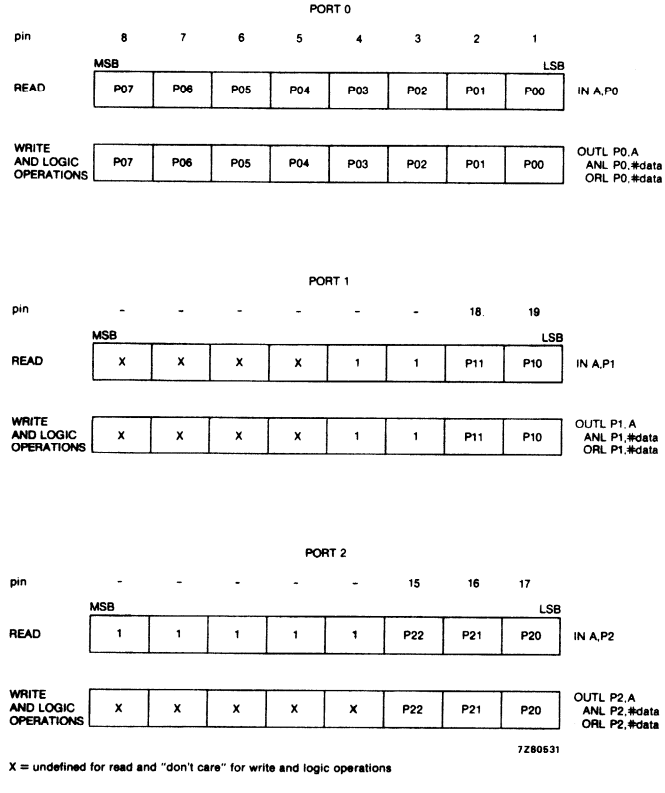


Fig. 4.3 Overview of port signals in the parallel mode.

4.2 Serial I/O mode

4.2.1 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.

The following functions should be done 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.

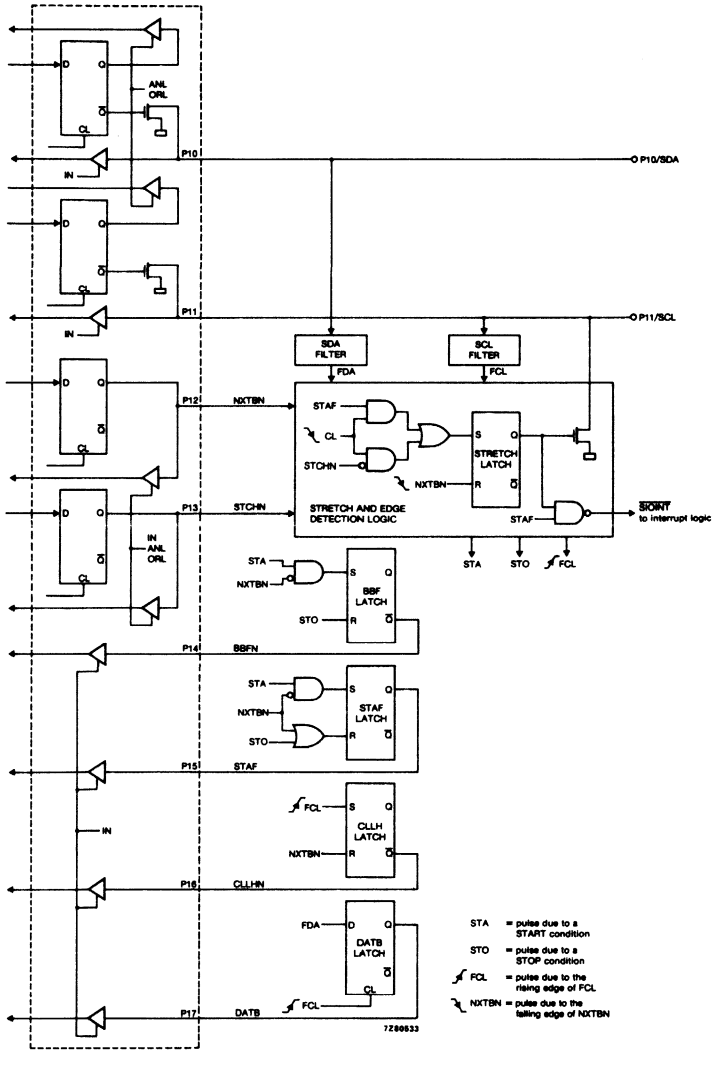


Fig. 4.4 The serial I/O interface.

4.2.2 Description of the signals to and from port P10 to 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 4.1.

* 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 this pin. 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 this pin. 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 this pin. 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. 4.5 gives an overview of the port signals in the serial mode.

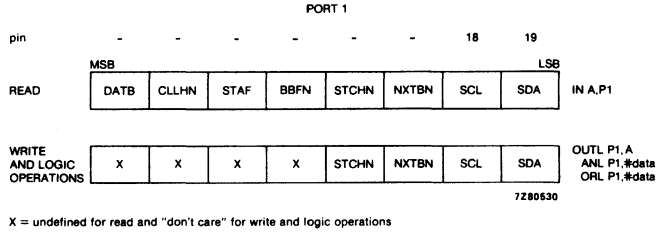


Fig. 4.5 Overview of port signals in the serial mode.

4.2.3 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 13.0.

4.3 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 respectively test for '1'and '0' levels. The T1 input has a self-biasing circuit that can detect when an a.c. signal crosses zero (within $\pm$ 135 mV when coupled through a 1 uF 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 10.0.

4.4 Test input T0/$\overline{\text{INT}}$

The T0/$\overline{\text{INT}}$ input may be used as:

- an external interrupt
- an input level that may be tested by the conditional jump instructions JT0 and JNT0.

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 it's negative going edge. The interrupt facility is discussed in more detail in chapter 5.0.

4.5 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).

5.0 INTERRUPT

When the external interrupt is enabled, a HIGH to LOW transition on the INT/T0 input initiates an external interrupt subroutine that, following completion of the current instruction causes a call to program memory location 3. The serial I/O interrupt causes a call to location 5, and a timer/event counter overflow a call to location 7, when the corresponding interrupt is enabled.

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. Before an interrupt subroutine starts, the program counter contents and bits 4, 6 and 7 of the PSW are saved in the program counter stack. Accumulator contents have to be saved by software. Interrupt acknowledge can be carried out by software via port pins. All interrupt routines must reside in memory bank 0.

The interrupt system is single-level, i.e. once an interrupt is detected, further interrupt requests are latched but ignored. After executing RETR, the microcontroller continues with the main program. If a second interrupt occurs during the running of the first routine the microcontroller will enter the second routine after executing 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 input will then initiate an interrupt subroutine and cause a call to the timer/counter interrupt vector location7.

5.1 Interrupt logic

The external interrupt of the MAB8422/8442 is active on the negative edge; a HIGH to LOW transition on the INT/T0 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. 5.1) and, when enabled, initiates an external interrupt service routine. The interrupt activity remains latched in the 'digital filter' untl 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 forces a subroutine CALL to program memory location 3. 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 and that causes other interrupts to be latched but ignored until the RETR instruction is executed.

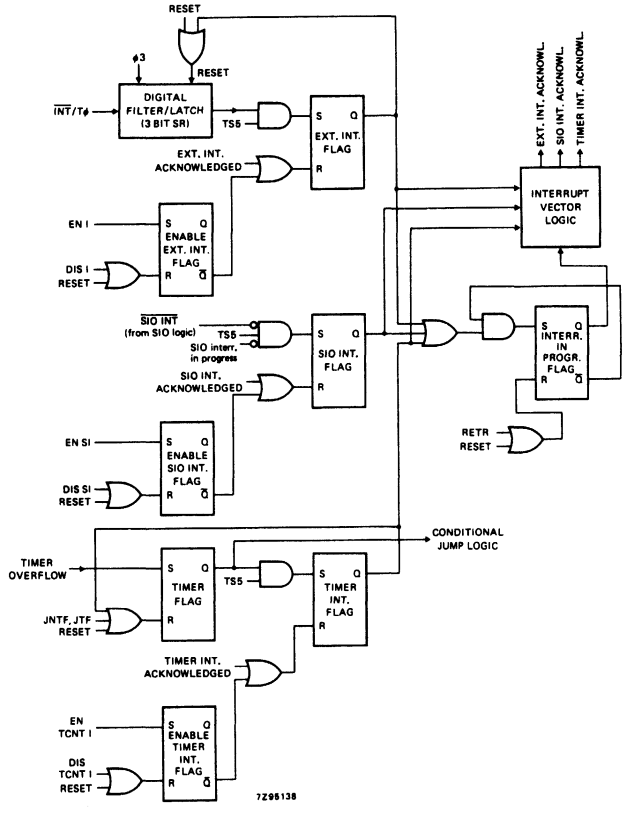


Fig. 5.1 Interrupt logic

Notes:

1. $\overline{\text{INT}}$/T0 negative edge is always latched in the digital filter/latch.

2. Proper interrupt timing is ensured when $\overline{\text{INT}}$/T0 is HIGH for more than 4 oscillator periods followed by LOW for more than 7 oscillator periods.

3. A DIS I instruction within an interrupt service routine clears a pending external interrupt.
   A DIS TCNT I instruction 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 interrupt is enabled, the timer flag is set only by a timer overflow for less than one machine cycle and is therefore testable with the JNTF and JTF instructions.

For proper latching in the 'digital filter/latch', the external interrupt should be HIGH for at least 4 clock cycles and LOW for at 7 clock cycles. The digital filter, therefore, ensures that the circuit only responds to genuine interrupts signals.

The maximum rate at which interrupt edges can be repeated and latched is 4 machine cycles.

Note that $\overline{INT}$/TO 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.

5.2 Interrupt program examples

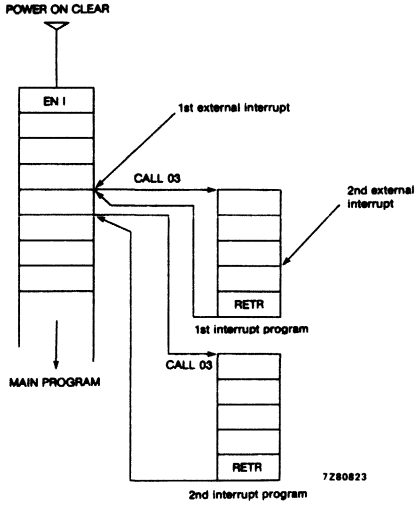


Fig 5.2 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

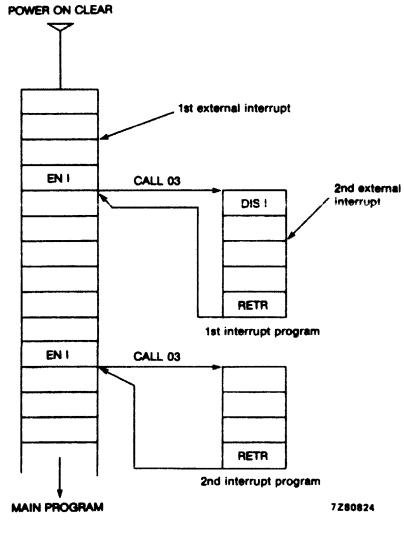


Fig. 5.3 Second external interrupt is postponed until enabled again.

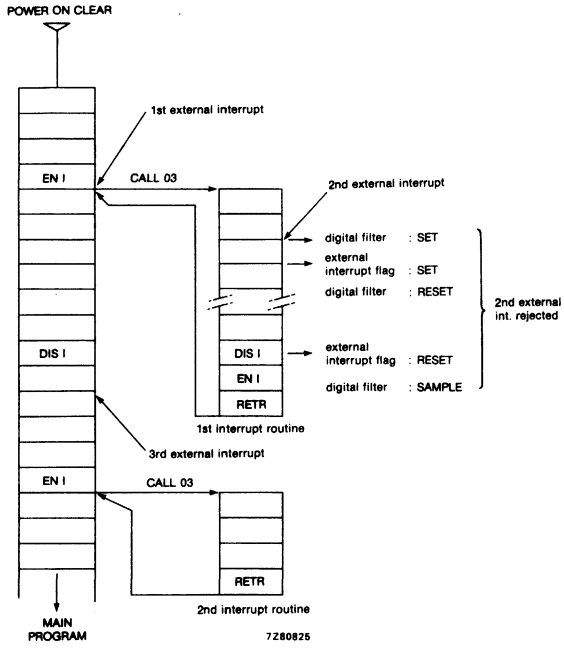


Fig. 5.4 Clearing of previous external interrupt.

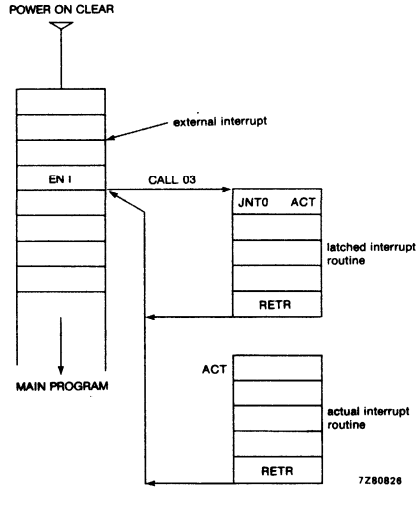


Fig. 5.5 Detection of previous and external interrupt.

6.0 CENTRAL PROCESSING UNIT

The MAB8422/8442 has arithmetic, logical and branching capabilities. The DA A, SWAP A, and XCHD instructions simplify BCD arithmetic and the handling of nibbles. The MOVP A,@A instruction permits efficient table look up from the current ROM page.

The conditional branch logic within the processor enables several conditions, internal and external to the processor, to be tested by the user's program. Table 2 lists the conditional jump instructions used to change the program execution 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 flag	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

7.0 PROGRAM STATUS WORD

The program status word (PSW) is an 8-bit word in the CPU which stores information about the current status of the microcontroller (Fig. 7.1). 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.

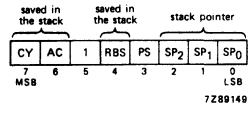


Fig. 7.1 Program status word.

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 or when an interrupt occurs. 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 cannot be used at the end of an interrupt.

8.0 PROGRAM COUNTER

A 13-bit program counter is used so that up to 8K bytes of ROM can be addressed. Fig. 8.1 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 during CALL and interrupt routines.

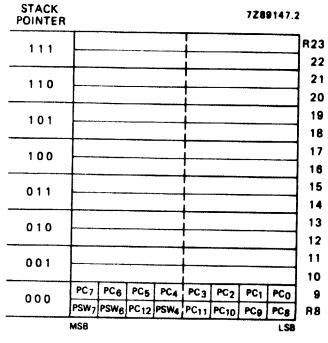


Fig. 8.1 Program counter.

9.0 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 more details.)

10.0 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. 10.1 overleaf). Table 3 gives the instructions that control the counter and the 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 the input of 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 (200 kHz for a 5 us 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 also generates an interrupt to the processor when the timer /event counter is enabled.

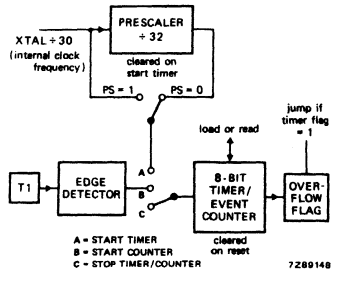


Fig. 10.1 Timer event/counter.

11.0 RESET

A psitive-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 serfial I/0)
- 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. Refer to the data sheet at the end of the descriptive section for details on the reset circuitry.

RESET has to be active HIGH for at least 2 machine cycles after the power supply and clock have been stabilized

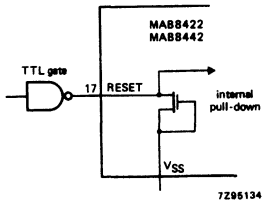


Fig. 11.1 External reset

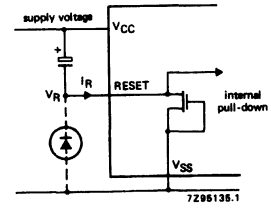


Fig. 11.2 Power-on reset

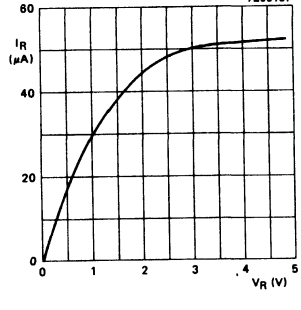


Fig. 11.3 Typical input characteristics

12.0 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 case the emulation probe is replaced by the ADB84X2.

- For prototyping of MAB8422/8442 based systems; in this case the user program is executed from an EPROM located on the board.

The way in which the adapter board is used is selected via jumpers, a block diagram of the board is given in Fig. 12.1.

On the board, a combination of the MAB8401WP-V8 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-V8 lead to a connector for the PMDS Microcomputer Adapter Box 8400 (PM827). The address/data bus is also connected to a 28-pin socket that contains the user program in EPROM during prototyping.

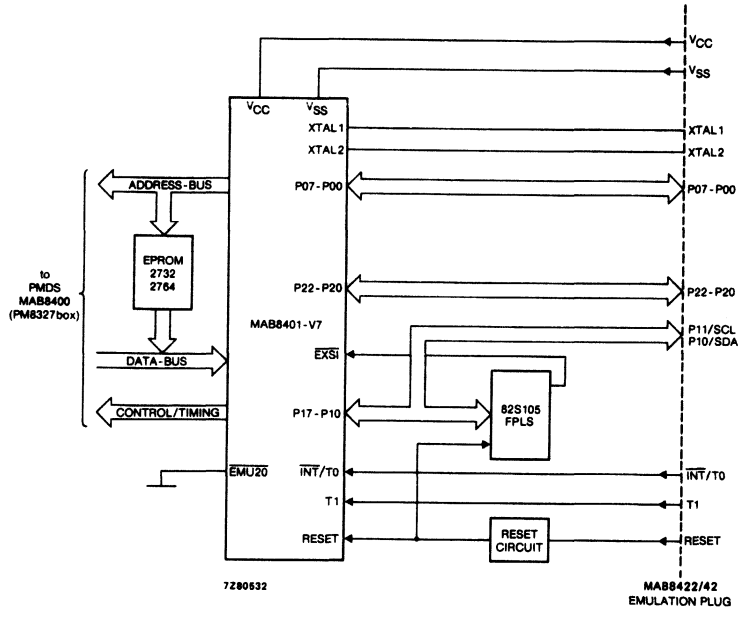


Fig. 12.1 The MAB8422/8442 adapter board.

13.0 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 USER and SLAVE USED, respectively.

13.1 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 Kbytes 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 section, 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. When a bus error or spurious START and/or STOP conditions occur 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
```

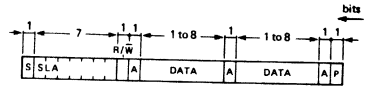
this slave subroutine is also called when in the master routine the arbitration is lost and the devices own slave address is called.

DATA FORMATS

SLAVE TRANSMITTER

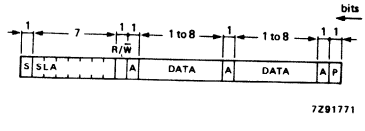
R/W BIT = '0'

FINAL ACKNOWLEDGE BIT = '1'



SLAVE RECEIVER

R/W BIT = '1'



S = START CONDITION
SLA = 7-BIT SLAVE ADDRESS
R/W = READ/NOT WRITE BIT
A = ACKNOWLEDGE BIT
DATA = 8-BIT DATA BYTE
P = STOP CONDITION

Fig. 13.1 Slave transmitter/receiver modes.

signal definition			pin name	pin number	function
SDA	EQU	H'01'	SDA/P10	19	I^2C-bus data I/O
SCL	EQU	H'02'	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.

Table 13.2

signal defintition			port bit	R/W	function
NXTBN	EQU	H'04'	P12	W	Next bit transfer NOT
STCHN	EQU	H'08'	P13	W	Stretch SCL LOW period NOT
BBFN	EQU	H'10'	P14	R	Bus busy flag NOT
STAF	EQU	H'20'	P15	R	Start condition flag
CLLHN	EQU	H'40'	P16	R	SCL LOW to HIGH NOT
DATB	EQU	H'80'	P17	R	Data bit received

NOT = active-LOW

```
        EJECT
*       SYMBOL DEFINITION.
*       ##################
*
ERRF2   EQU     H'40'           SCL PERIOD EXEEDS TIME LIMIT.
SCLC    EQU     250             250 X 12 MACHINE CYCLES TO DEFINE SCL PERIOD
*                               TIME LIMIT.(15 MSEC. @ 6 MHZ XTAL)
NTBY    EQU     4               MAX. NUMBER OF BYTES TO BE TRANSMITTED.
*                               (MUST BE ADAPTED TO THE APPLICATION).
NRBY    EQU     4               MAX. NUMBER OF BYTES TO BE RECEIVED.
*                               (MUST BE ADAPTED TO THE APPLICATION).
OWNAD   EQU     H'50'           OWN SLAVE ADDRESS IN THE 7 MSB. BIT 0 ="0".
*                               (MUST BE ADAPTED TO THE APPLICATION).
*
USED    EQU     1               FOR THE PRESENTS OF A MASTER SUBROUTINE.
MASTER  EQU     .NOT.USED       MASTER SUBROUTINE IS NOT PRESENT.
*                               IF MASTER SUBROUTINE IS PRESENT, CHANGE INTO:
*                               MASTER   EQU    USED
*
*       DATA MEMORY ALLOCATION.
*       ######################
*
SLVST   EQU     H'30'           SLAVE STATUS REGISTER. CAN BE RELOCATED.
*
*               SLAVE STATUS REGISTER CONTAINS:
*-------------------------------------------------------------------------
SRDAVF  EQU     H'80'           SLV/REC DATA-VALID FLAG.
STDRF   EQU     H'40'           SLV/TRX DATA-READY FLAG.
*-------------------------------------------------------------------------
*
STDTR1  EQU     SLVST+1         SLV/TRX DATA BYTE REGISTER 1
STDTR2  EQU     STDTR1+1                ,,                  2
STDTR3  EQU     STDTR2+1                ,,                  3
STDTR4  EQU     STDTR3+1                ,,                  4
*ETC.
SRDTR1  EQU     STDTR4+1        SLV/REC DATA BYTE REGISTER 1 (LAST STDTR+1)
SRDTR2  EQU     SRDTR1+1                ,,                  2
SRDTR3  EQU     SRDTR2+1                ,,                  3
SRDTR4  EQU     SRDTR3+1                ,,                  4
*ETC.
*THE NUMBER OF DATA BYTE REGISTERS MUST BE ADAPTED TO THE APPLICATION
*-------------------------------------------------------------------
```

```
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
```

13.2 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

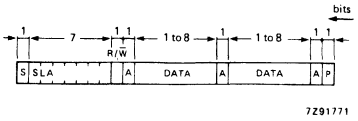
Any additional clock pulses are dealt with in the synchonization of the serial clocks, and require no software action. However, transfer is terminated and appropriate error flags set when spurious START and/or STOP conditions occur. Furthermore, noise that causes an error in the level of a bit will set 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)

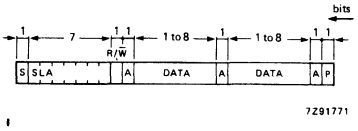
MASTER TRANSMITTER NO POINTER

R/$\overline{W}$ BIT = '0'

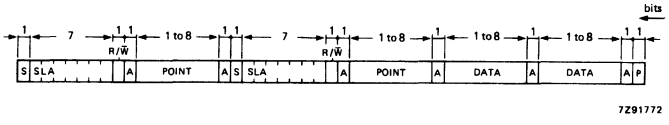


MASTER RECEIVER NO POINTER

R/$\overline{W}$ BIT = '1', FINAL ACKNOWLEDGE BIT = '1'



(b)



MASTER TRANSMITTER WITH POINTER AND
REPEATED START CONDITION

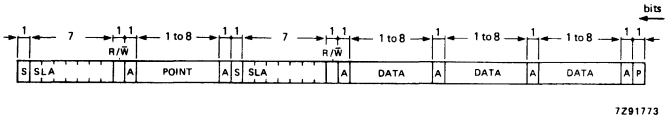


Fig. 13.2 Multi-master transmitter/receiver modes.

MASTER RECEIVER WITH POINTER AND
REPEATED START CONDITION
FINAL ACKNOWLEDGE BIT = '0'

S = START CONDITION
SLA = 7-BIT SLAVE ADDRESS
R/$\overline{W}$ = 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'

signal definition			pin name	pin number	function
SDA	EQU	H'01'	SDA/P10	19	I^2C-bus data I/O
SCL	EQU	H'02'	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 defintition			port bit	R/W	function
NXTBN	EQU	H'04'	P12	W	Next bit transfer NOT
STCHN	EQU	H'08'	P13	W	Stretch SCL LOW period NOT
BBFN	EQU	H'10'	P14	R	Bus busy flag NOT
STAF	EQU	H'20'	P15	R	Start condition flag
CLLHN	EQU	H'40'	P16	R	SCL LOW to HIGH NOT
DATB	EQU	H'80'	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
231 000000  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

14.0 INSTRUCTION SET

The MAB8422/8442 instruction set consists of over 80, one and two byte instructions. It is identical to the MAB8400 instruction set, except that the instructions MOV A,Sn, MOV Sn,A and MOV Sn,#data are not used. Program code efficiency is high because all RAM locations on a 256 byte page require only a single byte address.

Table 5 gives the instruction set of the MAB8422/8442; Table 4 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

	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)←((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		

	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 39 3A	1/2	output accumulator data to port p	(P0)←(A) (P1)←(A) (P2)←(A)	
	ANL Pp, #data	98 99 9A	2/2	AND port p data with immediate data	(P0)←(P0) AND data (P1)←(P1) AND data (P2)←(P2) AND data	9
	ORL Pp, #data	88 89 8A	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		

(90) (P0)←(A)

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) = 1111 P23, P22, P21, P20.
8. (S1) has a different meaning for read and write operation, see serial I/O interface.
9. Only for software transfer from the MAB8021.

* : 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 4 The instruction set 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 0 1 2							
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						
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						
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. The PCF84CXX microcontroller family

CONTENTS – PCF84CXX MICROCONTROLLER FAMILY

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
- PCF84C20 - 2 K ROM/64 RAM bytes
- PCF84C40 - 4 K ROM/128 RAM bytes

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. As well as being pin compatible, the instruction set is based on that of the MAB8048 and thus is completely compatible with the MAB84XX family. The microcontrollers have extensive 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 or 4 K ROM bytes plus a ROM-less version
- 64 or 128 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
- Serial I/O 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 (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

PCF84C20/40P: 28-lead DIL; plastic (SOT-117D).
PCF84C20/40D: 28-lead DIL; ceramic (CERDIP) (SOT-135A).
PCF84C20/40T: 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)
PCF84C00WP : 68-lead plastic leaded chip carrier (PLCC) (SOT-188A)

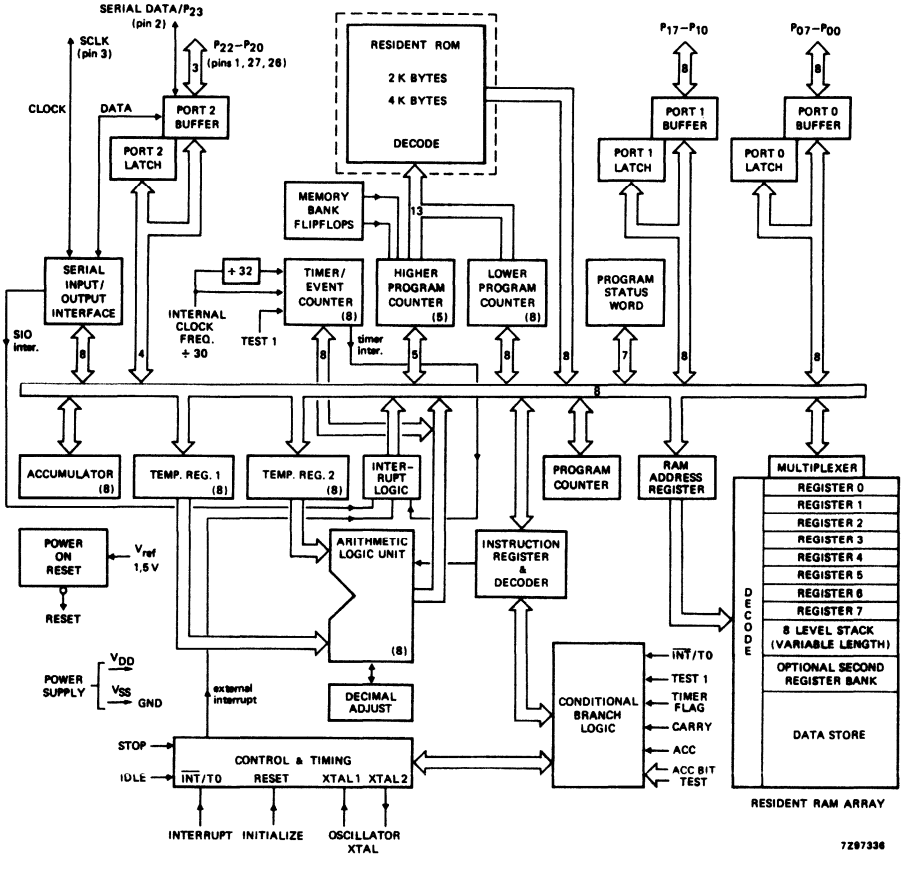
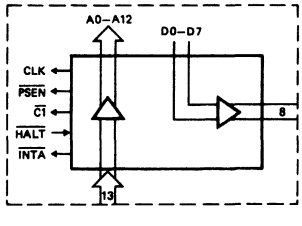
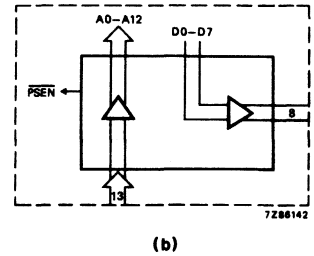


Fig. 1 Block diagram of the PCF84CXX



(a)

Fig. 1(a) Replacement of dotted part in Fig. 1 for the PCF84C00WP bond-out version.



(b)

Fig. 1(b) Replacement of dotted part in Fig. 1 for the PCF84CXX 'piggy-back' version.

3.1 Pin assignment

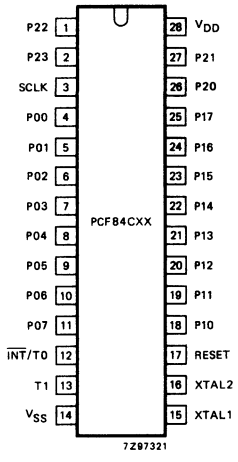


Fig. 2 Pinning diagram PCF84CXX.

3	SCLK	Clock: bidirectional clock for serial I/O.
4-11	P00-P07	Port 0: 8-bit quasi-bidirectional I/O port.
12	$\overline{INT}$/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.
13	T1	Test 1: test input pin, directly tested by conditional branch instructions JT1 and JNT1. T1 also operates as an input to the 8-bit timer/event counter, using the STRT CNT instruction.
14	V_{SS}	Ground: circuit earth potential.
15	XTAL 1	Oscillator input: connection to timing component which determines the frequency of the internal oscillator also the input for an external clock source.
16	XTAL 2	Oscillator output
17	RESET	Reset input: used to initialize the processor (active HIGH), or output of power-on-reset circuit.
18-25	P10-P17	Port 1: 8-bit quasi-bidirectional I/O port.
26, 27, 1, 2	P20-P23	Port 2: 4-bit quasi-bidirectional I/O port. P23 is the serial data input/output in serial I/O mode.
28	V_{DD}	Power supply: 2,5 V to 5,5 V.

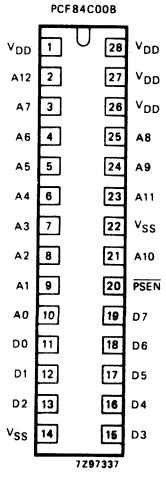


Fig. 3 Pinning diagram: PCF84C00B 'Piggy-back' version top pinning; to access a 2732 or 2764 EPROM.

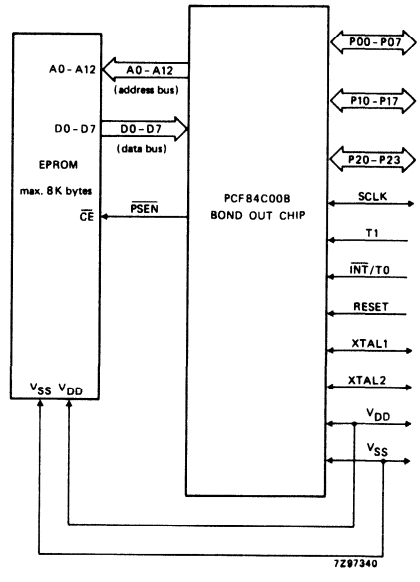


Fig. 3a Connection of EPROM to 'Piggy-back' package PCF84C00B.

PIN DESIGNATION		
14, 22	V_{SS}	Ground
1, 26-28	V_{DD}	Power supply
10-3, 25, 24, 21, 23,2	A0-A12	Address outputs
11-13, 15-19	D0-D7	Data
20	$\overline{PSEN}$	Program store enable

Notes

1. RAM capacity of PCF84C00B is 256 bytes.
2. Access time for ROMS/EPROMS $< 7 \times {}^{1}/_{f_{XTAL}}$.

4.0 BOND-OUT VERSION

The bond-out version is exactly the same as the MAB8400WP, except that the zero -cross voltage detection is not included in the PCF84C00WP.

The PCF84C00WP contains no on-board ROM, but has all address data and control lines brought out to access up to 8 K of external memory and supports emulation.

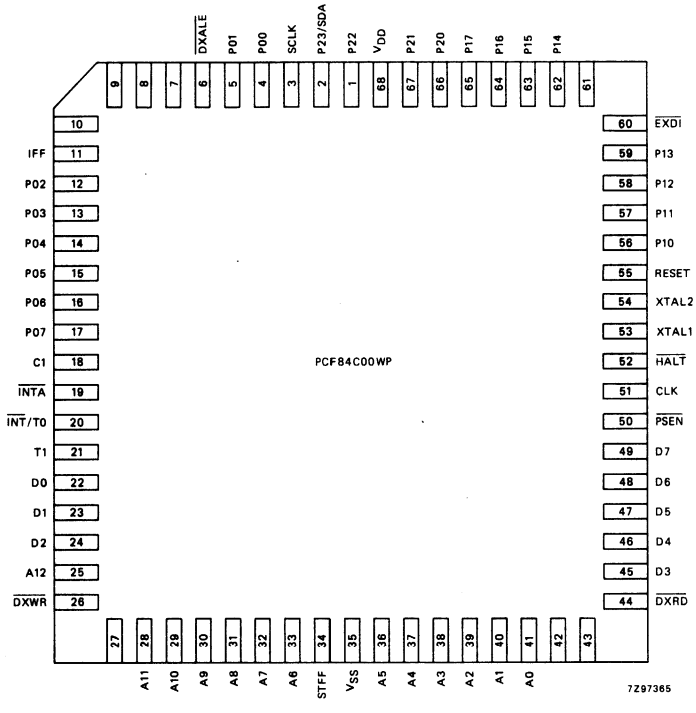


Fig. 4 Pinning diagram PCF84C00WP

4.1 Pad assignment

Standard Pins

Mnemonic	Pin	Function
VSS	35	Ground (0V), reference potential
VDD	68	Power supply voltage
P00 to P07	4,5,12-17	8-bit quasi bidirectional I/O port (port 0)
P10 to P17	55-59, 62-65	8-bit quasi bidirectional I/O port (port 1)
P20 to P23	66,67,1,2	4-bit quasi bidirectional I/O port (port 2)
P23/SDA		1-bit quasi-bidirectional I/O port P23; In the I2C mode it is the serial data input open drain output pin SDA
SCLK	4	Serial clock input/open drain output in the I2C mode
$\overline{INT}$/T0	20	External interrupt input (sensitive to negative-going edge), input pin testable using the JT0, JTN0 instructions.
T1	21	Input pin testable using the JT1, JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction.
RESET	55	Input , used to initialize the processor (active HIGH), or output of power-on-reset circuit.
XTAL1	53	Oscillator input, crystal (determines the internal oscillator frequency) or external clock generator.
XTAL2	54	Oscillator output.

Non-standard Pins

The following 32 pins are required for EPROM access on the piggyback package and in-circuit emulation.

Mnemonic	Pin	Function
A00 to 12	41-36, 33-28,25	Program memory address lines (active HIGH A0 = LSB, A12 = MSB.
D0 to D7	22-24, 45-49	Data lines (active HIGH), used for reading external program memory and reading/writing to external Dx registers. D0 = LSB, D7 = MSB. Internal pull ups are provided.
CLK	51	Clock output buffered and inverted from XTAL2.
$\overline{PSEN}$	50	Program Store Enable (active LOW). Used for enabling the EPROM on the piggyback package. During emulation it enables the emulation memory and it is used to indicate machine cycles. Active during TS9 and TS10 of each machine cycle and TS1 of the next machine cycle. In IDLE mode PSEN is still operational, in STOP mode this signal is halted.
$\overline{C1}$	18	Cycle 1 indication output (active LOW). During emulation this signal indicates the moment of opcode fetch (useful for instruction decoding, real time trace). Active from begin of TS10 of the cycle preceeding cycle 1 to begin TS10 of cycle 1. In the IDLE or STOP mode C1 is LOW.

Mnemonic	Pin	Function
$\overline{\text{HALT}}$	52	Halt input (active LOW). If activated, the current instruction is terminated and the microcomputer stops execution (HALT mode). The next program counter address is preset on the address bus. Program counter and timer/counter are not updated in the HALT mode; the serial I/O logic is not affected and will finish the current transmit/receive action. Interrupts are sampled only in operating mode but not in HALT mode. If HALT# goes HIGH, program execution continues. An internal pull-up is provided for this pin.
$\overline{\text{INTA}}$	19	Interrupt acknowledge output (active LOW) indicates that an interrupt has been accepted. Active from the beginning of TS6 in the interrupt cycle to the beginning of TS7 in the second cycle of the internally forced 'CALL vector address' instruction. While INTA is active the address bus presents the address that has been saved in the stack (return address); the C1 output indicates opcode-fetch cycles as if a user 'CALL' is being executed.
$\overline{\text{DXALE}}$	6	Address Latch Enable output for Dx registers (active high). With the falling edge, the Dx address (output of the external program memory during the 2nd byte fetch) can be latched in an appropriate external latch. This signal occurs only during MOV Dx,A, MOV A,Dx, ANL Dx,A and ORL Dx,A instructions, with x = 0 to FF H.

Mnemonic	Pin	Function
$\overline{\text{DXRD}}$	44	Read strobe for Dx registers (active LOW). While this signal is active, external registers emulating Dx registers can be enabled to the databus of the 84C00WP. This signal occurs only during MOV A,Dx, ANL Dx,A and ORL Dx,A instructions, where x = 0 to FF H.
$\overline{\text{DXWR}}$	26	Write strobe for Dx registers (active LOW). On the rising edge, data on D0 to D7 can be latched in appropriate external registers, emulating Dx. This signal occurs only during MOV Dx,A, ANL Dx,A and ORL Dx,A instructions, where x = 0 to FF H.
$\overline{\text{EXDI}}$	60	External derivative interrupt input (active LOW). EXDI is 'OR'-ed with the internal serial I/O interrupt. It can be used to initiate an interrupt from external hardware, emulating derivative functions. An internal pull-up is provided. A derivative interrupt is sampled during TS6 and accepted if EN SE has been executed before and the 84C00WP is not in an interrupt routine. Derivative interrupts are not latched in the 84C00WP.
IFF	11	IDLE flag; if set to '1', indicates that the 84C00WP has entered the IDLE state. It will be set to '1' during execution of the 'IDL' instruction.
STFF	34	STOP flag; if set to '1', indicates that the 84C00WP has entered the STOP state. It will be set to '1' during execution of the 'STOP' instruction.

5.0 FUNCTIONAL DESCRIPTION

5.1 Program memory (ROM)

The program memory consists of 1024, 2048 or 4096 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, organised 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.

5.2 Data memory (RAM)

Data memory consists of 64 or 128 bytes and all locations are indirectly addressable using RAM pointer registers; for this, 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 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, 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.

Figures 5 & 6 show the program memory and data memory maps. Figure 7 illustrates the structure of the program counter stack.

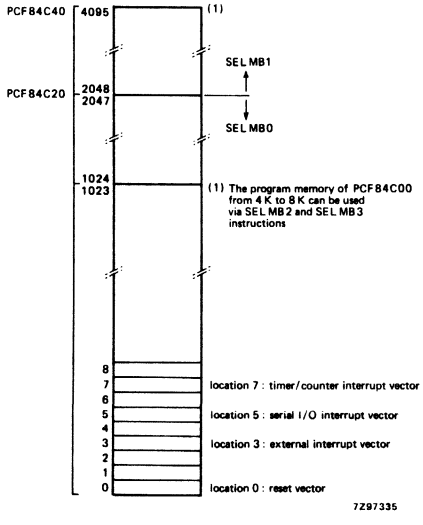


Fig. 5 Program memory map

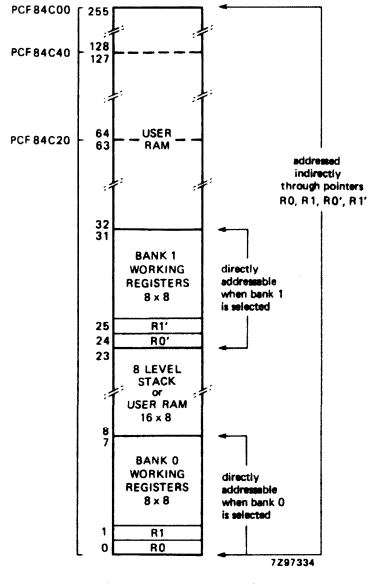


Fig. 6 Data memory map

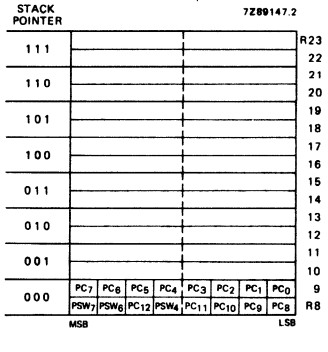


Fig. 7 Program counter stack

5.3 Input/Output - See MAB84XX section

Identical to MAB84XX except for parallel ports

5.3.1 Parallel ports

The I/O ports of the PCF84CXX are functionally equivalent to the MAB84X1 except that the circuitry is optimised for low current consumption.

Output data written to P0, P1 or P2 is latched and remains unchanged until read by an input instruction.

Input lines are fully CMOS compatible, output lines can drive one LS-TTL or CMOS load.

Figure 8(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 current 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 then switched on to provide current sink. 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 select individually 19 of the 20 parallel port pins (not P23), by 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. 8(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. 8(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. 8(c)).

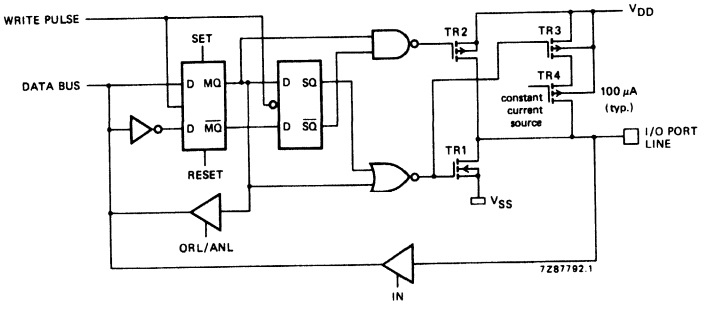


Fig. 8(a) Standard output with switched pull-up current source.

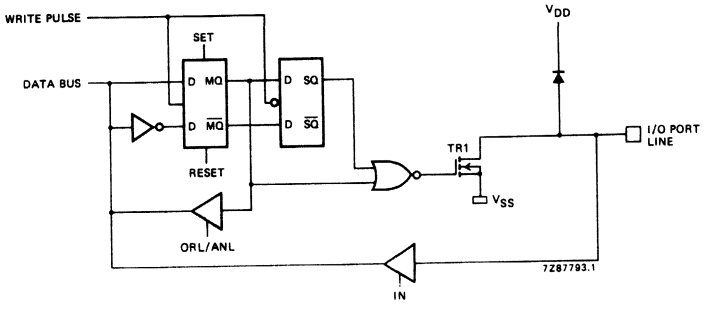


Fig. 8(b) Open drain output.

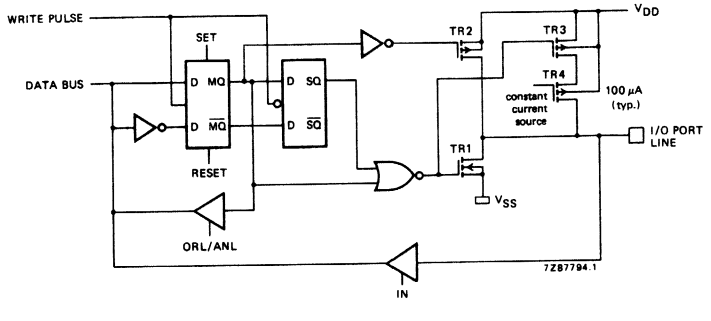


Fig. 8(c) Push-pull output.

Timing reference on PCF84CXX ports is identical to that of the MAB84X1, see figure 9 below.

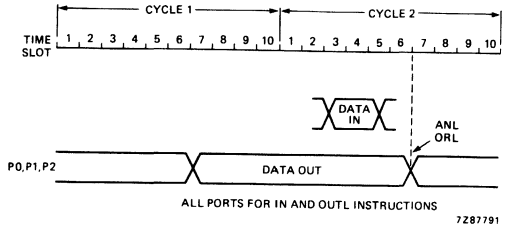


Fig. 9 Timing diagram of all ports on IN and OUTL instructions. For ANL and ORL instructions, the ports change on time slot 7 of cycle 2.

5.3.2 Serial I/O

Identical to MAB84XX except:

- For the PCF84CXX all values of control register S2 (H'01' to H'1F') may be used in the 'low-speed' mode. See table 3 in the MAB84XX serial I/O section.
- The electrical specification of P23 and SCLK of the PCF84CXX is slightly different to accommodate the I^2C bus timing at 10 MHz. This information is included in the data sheet section at the back of this manual.
- In NORMAL (running) and IDLE mode, the serial I/O logic remains active; it's 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. This must be carried out only when the PCF84CXX has finished a serial data transfer.

5.4 Interrupts

When the external interrupt is enabled, a HIGH to LOW transition on the INT/T0 input initiates an external interrupt subroutine which causes a call to program memory location 3, following completion of the current instruction. Serial I/O interrupt, when enabled, causes a call to location 5, and a timer/event counter overflow a call to location 7, when the interrupt is enabled.

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 are 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 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 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. 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 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 84CXX is shown in figure 10.

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.

Upon execution of a DIS I instruction, the PCF84CXX always clears both the digital filter/latch and the external interrupt flag. The MAB84X1 will not clear it's digital filter latch with this instruction while executing main program. However, both the MAB84X1 and PCF84CXX will clear their respective digital filter latch and external interrupt flag if they are operating in an external interrupt program.

There is no internal pull-up or pull-down device connected to the external interrupt input (pin 12). If required pin 12 must be externally connected to a resistor ($R = \leqslant 100\ k\Omega$). When the external interrupt is not used pin 12 must be connected to V_{SS}.

5.4.1 Interrupt logic

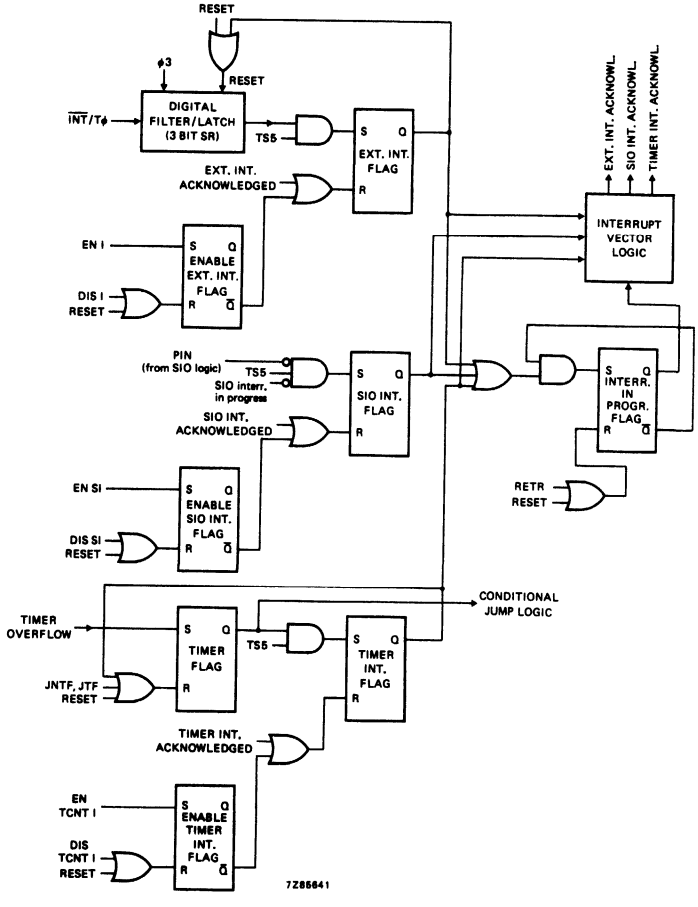


Fig. 10 Interrupt logic PCF84CXX

NOTE:

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 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

5.4.2 Interrupt program examples

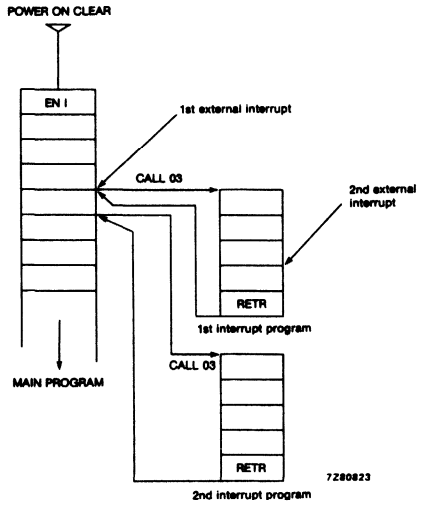


Fig. 11 An external interrupt during the first interrupt program remains latched until the interrupt program is complete.

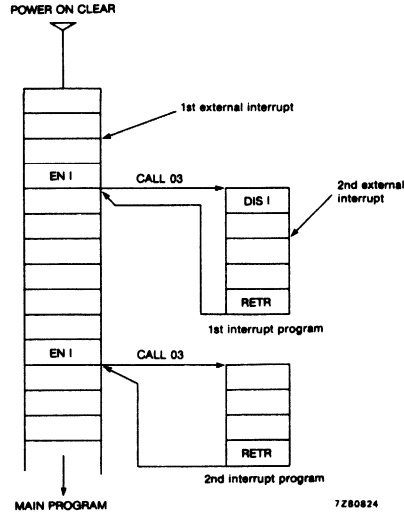


Fig. 12 Second external interr -upt is postponed until enabled again

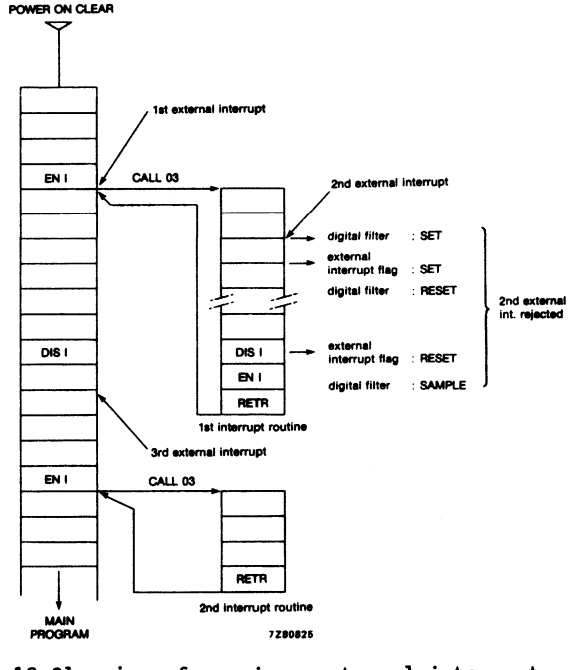


Fig. 13 Clearing of previous external interrupt

5.4.2 Interrupt program examples (continued)

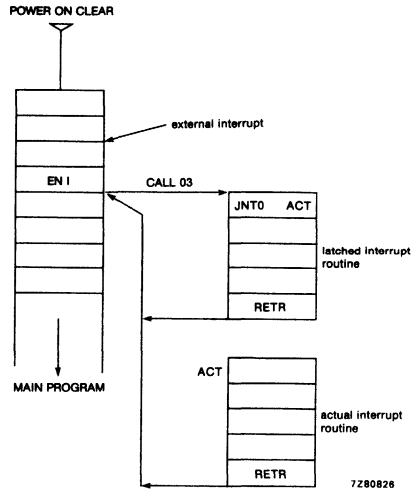


Fig.14 Detection of previous and actual external interrupt

5.4.3 Timer interrupt logic

A minor difference exists between the timer interrupt logic in the MAB84X1 and PCF84CXX;

MAB84X1: The Timer Flag (TF) is reset when the Timer interrupt flag goes high. The Timer interrupt flag is set if the timer interrupt is enabled and the Timer Flag set.

PCF84CXX: 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.

5.4.4 Exiting the WAIT mode via an interrupt

The microcontroller will exit the IDLE mode if any one of the following three interrupts are enabled:-

1) External
2) Serial I/O
3) Timer/event counter

5.5 Test inputs T1, $\overline{\text{INT}}$/T0

5.5.1 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. The maximum input voltage is 3 V (peak to peak), so maximum voltage ratings must be 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.

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 pin 13 must be connected to 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.

5.6 Test input $\overline{INT}$/T0

The $\overline{INT}$/T0 input may be used as:

- an external interrupt
- an input level that may be tested by the conditional jump instructions JT0 and JNT0.

When used as an external interrupt input, the interrupt signal must be held LOW for a minimum of 4 clock periods and HIGH for a minimum of 4 clock periods. The interrupt is detected on it's negative going edge. The interrupt facility is discussed in more detail in chapter 5.4.

5.7 Oscillator and clock

Oscillator

The oscillator can be inhibited by the STOP instruction under software control. It is also inhibited when a low-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 re-started after a STOP instruction by a LOW level at the $\overline{INT}$/T0 pin or a HIGH level at the RESET pin.

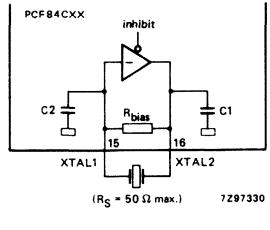


Fig. 15 Oscillator with integrated elements.

Due to the dynamic logic of circuitry the minimum oscillator frequency must be 100 kHz. If a quartz crystal is used no additional external components are needed (see fig. 15).

There are two other possibilities of clock generation (see fig. 16):

a) Applying an external clock to pin 15, XTAL1 (see fig. 16)

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:

$$f = \frac{1}{2\pi\sqrt{L\,\frac{C1T \cdot C2T}{C1T + C2T}}}$$

where: $C1T = C_1 + C_{1I} + C_{1W}$

$C2T = C_2 + C_{2I} + C_{2W}$

$C_{1I} \approx 4pF$; $C_{2I} \approx 6pF$

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.

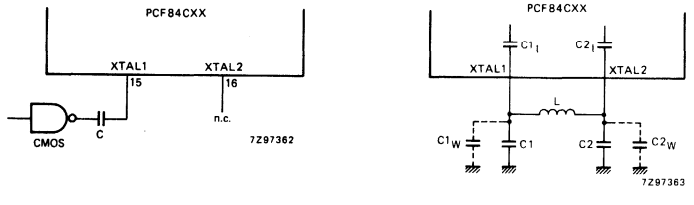


Fig. 16 Two possible clock circuit arrangements

5.8 Timer/event counter

T1 as an input to the timer/event counter is described in section 5.5

An internal 8-bit binary up-counter is provided. This can count external events, modulo-32 machine cycles, or machine cycles directly (fig. 17). 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 us 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.

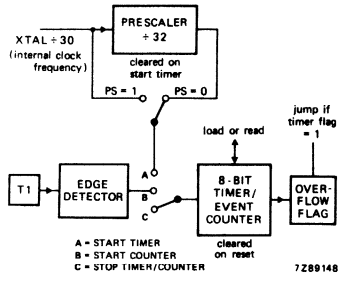


Fig. 17 Timer event counter.

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.18). 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.

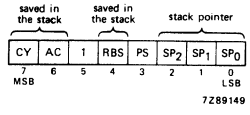


Fig. 18 Program status word

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 (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.

5.10 Program counter

The 13-bit program counter can address up to to 8 K bytes of ROM/RAM, figure 19 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.

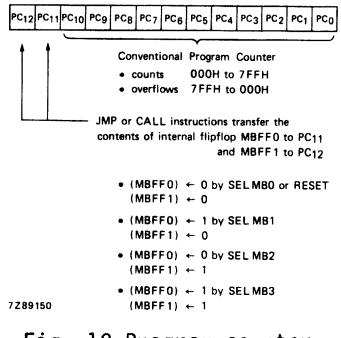


Fig. 19 Program counter.

5.11 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 not 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

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,
- cancels the WAIT mode,
- cancels the STOP mode.

The external power-on-reset circuit should consist of a 1 uF capacitor connected between V_{DD} and the RESET pin, and a resistor (<100 kΩ) connected between the RESET pin and ground. An optional diode ensures a correct reset in the event of a dip in V_{DD}. The power-on-reset time constant (T_R) must be greater than or equal to four times the rise time during power-up.

The internal reset circuit monitors the PCF84CXX supply voltage. When the supply voltage drops below the switching level (approx 1 V), a reset (HIGH) is applied to pin 17. This reset is cleared after a fixed delay (1866 clock periods) when the supply voltage rises above the switching level once more.

Figure 20 illustrates internal/external power-on-reset circuitry.

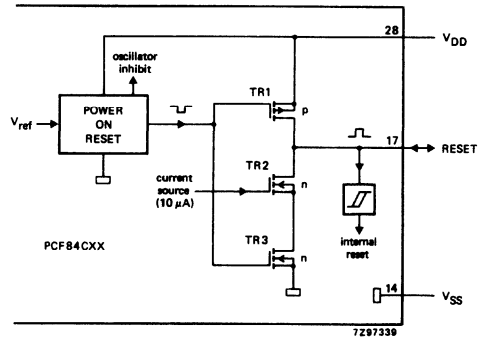
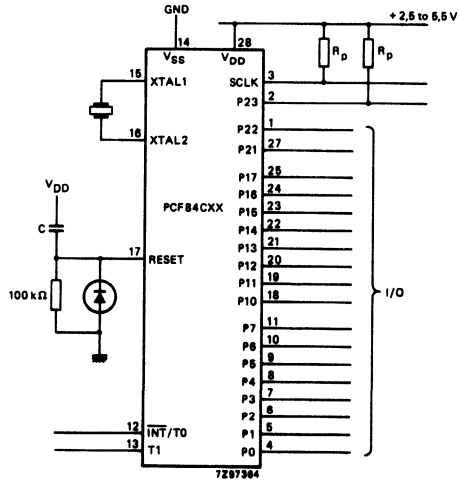


Fig. 20 Power-on reset.

5.13 Figure 21 shows connection of the 84CXX in a typical system



Note to figure 21: All unused pins should be connected to V_{SS} or V_{DD}. The values of the timing components can be found in section 5.7.

The value of pull-up resistors R_p can be found in the I^2C section of this manual.

5.14 The instruction set

Th Instruction set of the PCF84CXX is identical to that of the MAB84X1 with the addition of two instructions for the power saving modes;

WAIT: opcode H'01' (1 byte/1 cycle)
STOP: opcode H'22' (1 byte/1 cycle)

Table 3 shows the instruction map of the PCF84CXX

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	JB0 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								

	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	

INSTRUCTION SET (continued)

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

	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 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	

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 99 9A	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 89 8A	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 9D 9E	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.

6.0 IDLE AND STOP MODES

IDLE mode

When the microcontroller enters the IDLE mode via the IDLE instruction (H'01') 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. 22).

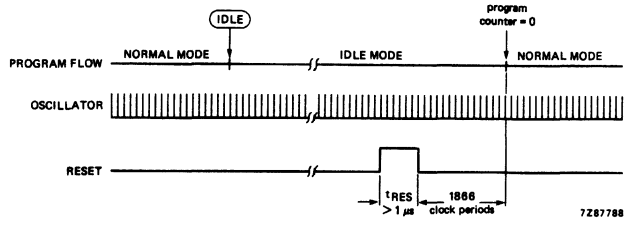


Fig. 22 Exit from IDLE mode via a RESET.

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. 23).

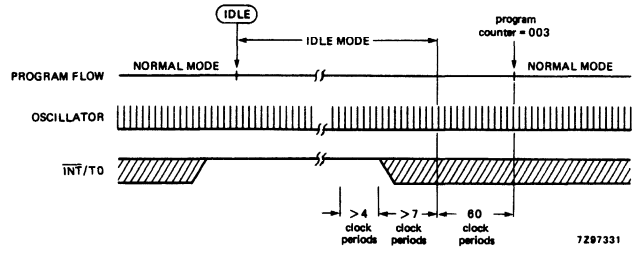


Fig. 23 Exit from IDLE mode via an interrupt.

Exit from the IDLE mode is ensured when $\overline{INT}$/T0 is HIGH for 4 CP (clock periods) followed by a LOW for 7 CP. After the initial forced CALL H'003' operation (60 CP) the program continues with the external interrupt service routine.

STOP mode

The microcontroller enters the STOP mode by the STOP instruction (H'22') 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. 10).

If the microcontroller exits from the STOP mode by activating RESET, a normal RESET sequence is executed.

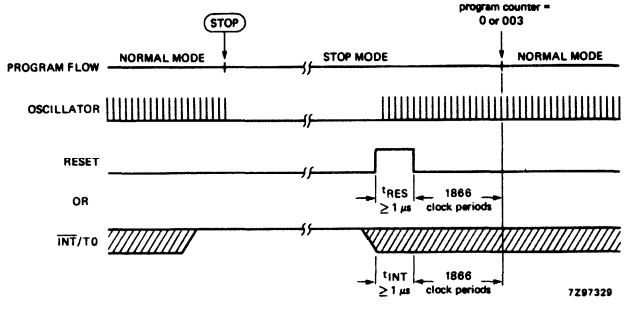


Fig. 24 Entering and exiting the STOP mode.

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 us will cause the microcontroller to exit the STOP mode.

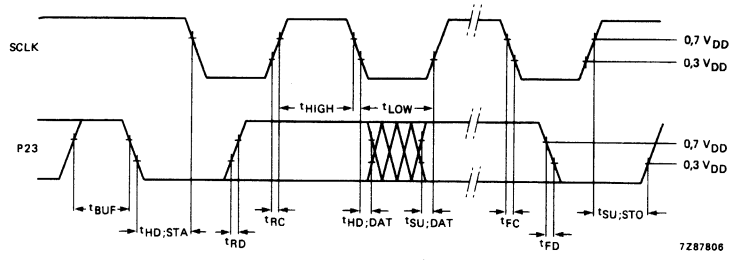


Fig. 25 PCF84CXX timing requirements for the P23 and SCLK input signals.

Table 8 Input timing shown in figure 25

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 µs
t_{RC}	$\leqslant$	1 µs
t_{FD}	$\leqslant$	1 µs
t_{FC}	$\leqslant$	0,3 µs

Notes to Table 8

t_{XTAL} = one period of the XTAL input frquency (f_{XTAL})
= 167 ns for f_{XTAL} = 6 MHz.
These figures apply to all modes

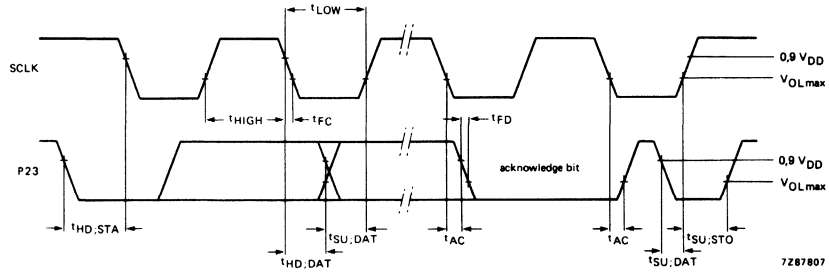


Fig. 26 PCF84CXX timing requirements for the P23 and SCLK output signals.

Table 9 Output timing shown in figure 26

symbol	timing	
	normal mode (ASC in S2 = 0)	low-speed mode (ASC in S2 = 1)
$t_{HD;STA}$	1/2 (DF + 9) t_{XTAL}	3/4 (DF + 9) t_{XTAL}
t_{HIGH}	1/2 (DF) t_{XTAL}	3/4 (DF) t_{XTAL}
t_{LOW}	1/2 (DF) t_{XTAL}	1/4 (DF) t_{XTAL}
$t_{SU;STO}$	1/2 (DF - 3) t_{XTAL}	1/4 (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 $\geqslant$ 51	$\geqslant$ $15t_{XTAL}$ $\leqslant$ $24t_{XTAL}$	—
for DF $\geqslant$ 99	—	$\geqslant$ $15t_{XTAL}$ $\leqslant$ $24t_{XTAL}$
for DF $\leqslant$ 51	$\geqslant$ $9t_{XTAL}$	$\geqslant$ $9t_{XTAL}$
for DF $\leqslant$ 99	—	$\geqslant$ $9t_{XTAL}$
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 9

t_{XTAL} = one period of the XTAL input frequency (f_{XTAL})
= 167 ns for f_{XTAL} = 6 MHz.
DF = divisor (see Table 2 Serial I/O section).
C_b = the maximum bus capacitance for each line.

7. The MAB84XX serial I/O

CONTENTS – MAB84XX SERIAL I/O

1.0 INTRODUCTION

The increasing demand for microcontrollers in domestic and industrial applications has led to the development of the MAB84XX family. More widespread use of distributed intelligence in todays controller sytems, has demanded the introduction of simpler, more efficient data exchange. To the MAB84XX 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 MAB84XX (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 necersary 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-line 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 it's 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.

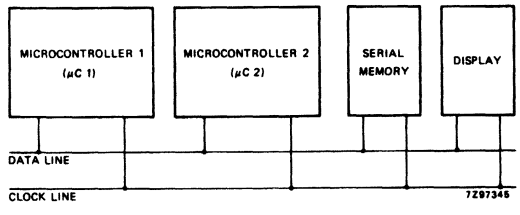
Transmitter	- The device which sends data to the bus
Receiver	- The device which receives data from the bus
Master	- The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	- The device addressed by a master
Multi-master	- More than one master can attempt to control 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 (H'00'). 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).

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 IC's that can be connected to the bus, and it s length, are solely limited by the maximum bus capacitance of 400 pF.

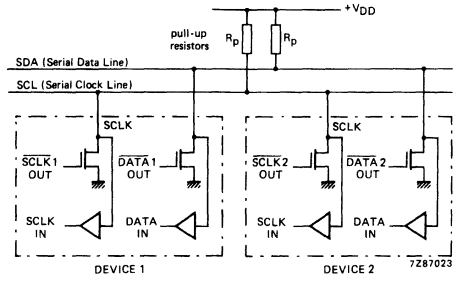


Fig. 2 Connection of two microcontrollers to the serial bus.
R_p = 1 to 5 kΩ (depending upon current required)

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

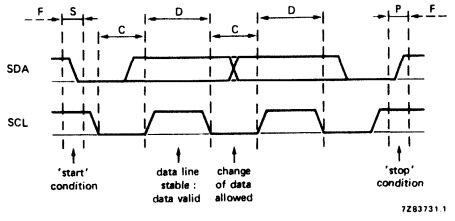


Fig. 3. Sequence of bit transfer on the serial bus.

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 are used:

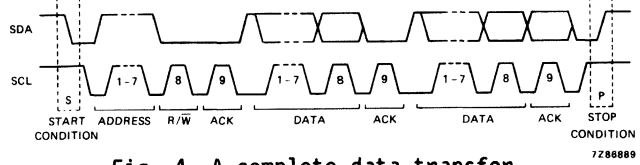


Fig. 4. A complete data transfer

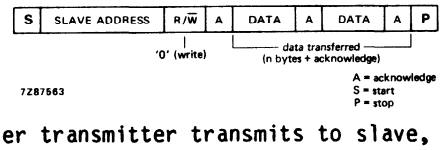
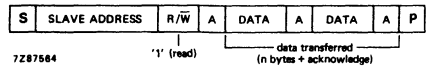
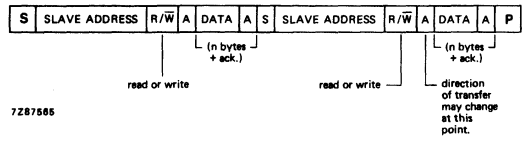


Fig. 5.(a) Master transmitter transmits to slave, direction unchanged



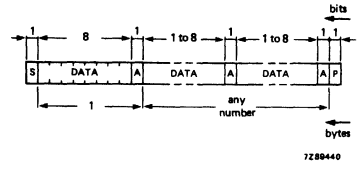
(b) Master reads slave immediatly after first byte

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.



(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').



(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 in synchrony 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 it's 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 it's 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 it's 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 SCLK open drain only).

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.

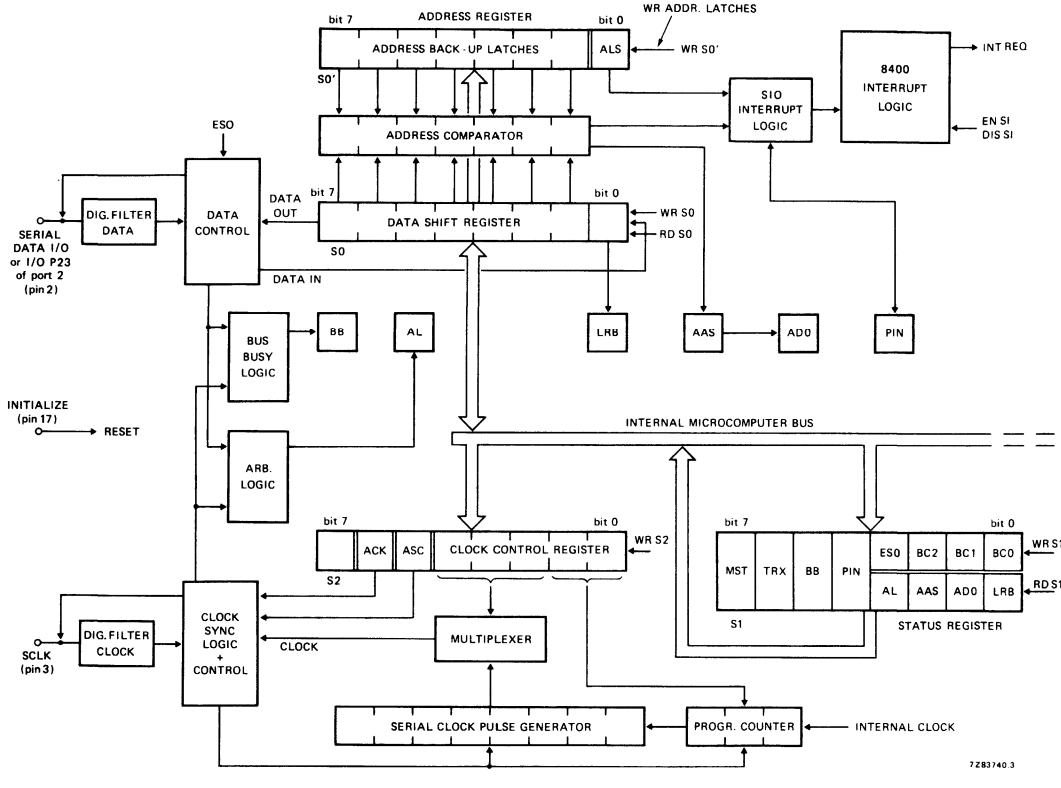


Fig. 6. Block diagram of the MAB84XX SIO interface

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 receiver on the serial bus is shifted into SO, most significant bit first. After transmission of a complete byte or reception 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, onto 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.

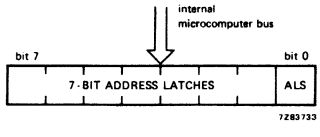


Fig. 7 Address register in the SIO interface.

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 (H'00'). 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 seperate 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.

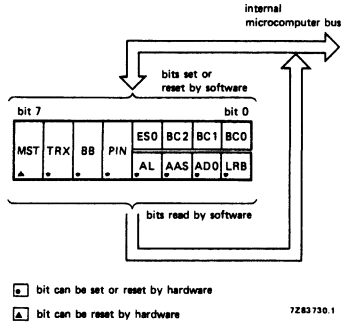


Fig. 8 Bit allocation of the status word in register S1.

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 summarised 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
```

ASS: 'Addressed as slave' bit. ASS 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.

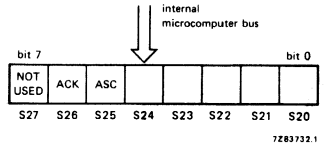


Fig. 9 Bit allocation in clock control register S2

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 µs 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 µs and the LOW time is 130 µs.

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 devisor	4,43 MHz crystal approx. f_{clock}(kHz)	6 MHz crystal 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

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.

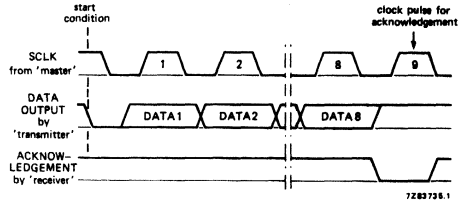


Fig. 10 Acknowledgement between 'receiver' and 'transmitter'.

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 generated for the first 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 well 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 it's output off so that the master generating DATA 2 wins the arbitration and it's 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.

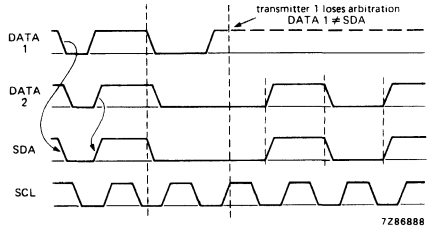


Fig. 11 Arbitration procedure between two master transmitters.

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 proceedure to take place. Clock synchronisation 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 it's 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 it's 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.

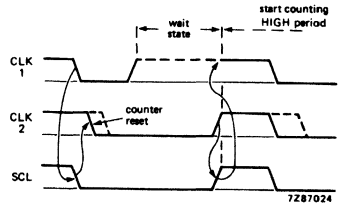


Fig. 12 Synchronization of two SIO clock generators.

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.13. 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 assymetrical 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 S0'

The next step is to load address register S0'. 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 S0' 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 S0' 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 S0,# H'84'     ; LOAD SLAVE ADD HEX '42'(bit7-1) IN S0' 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								Mode	Data received or transmitted	Software response
MST	TRX	BB	PIN	AL	AAS	ADO	LRB			
0	0	1	0	0	1	1/0	X	SLV/REC	Own slave/all 0´s address received with R/W = 0	A dummy read of S1
0	0	1	0	0	0	1/0	X	SLV/REC	Data received in S0	00101XXX to S1 and/or read S0
0	1	1	0	0	1	0	X	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 10111XX to S1 or a dummy read of S0
1	0	1	0	0	0	0	X	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	0	X	MST/TRX lost arbit. now SLV/REC1	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	X	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)	

X = 1 or 0, or underlined-don´t care

XXX = BC2, BC1, BC0 according to table 2.

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 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 inlto 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 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  SIL 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           ; TSET 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 to be preset as they can 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
```

9.6 Generation of a repeated 'start' condition

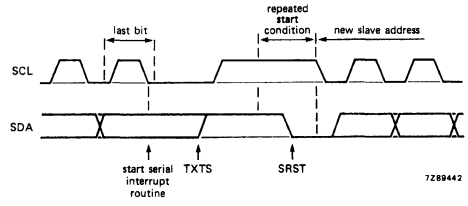


Fig. 13 Repeated 'start' condition on the serial bus.

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 4(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              ; COMPLETE 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' generates 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'49'      ; 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 TSP1           ; 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.

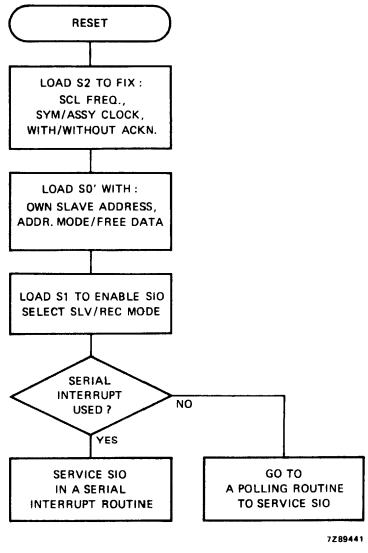


Fig. 14. Initialization flow chart. A prior RESET signal has cleared all bits of the SIO registers except PIN = 1 in status register S1.

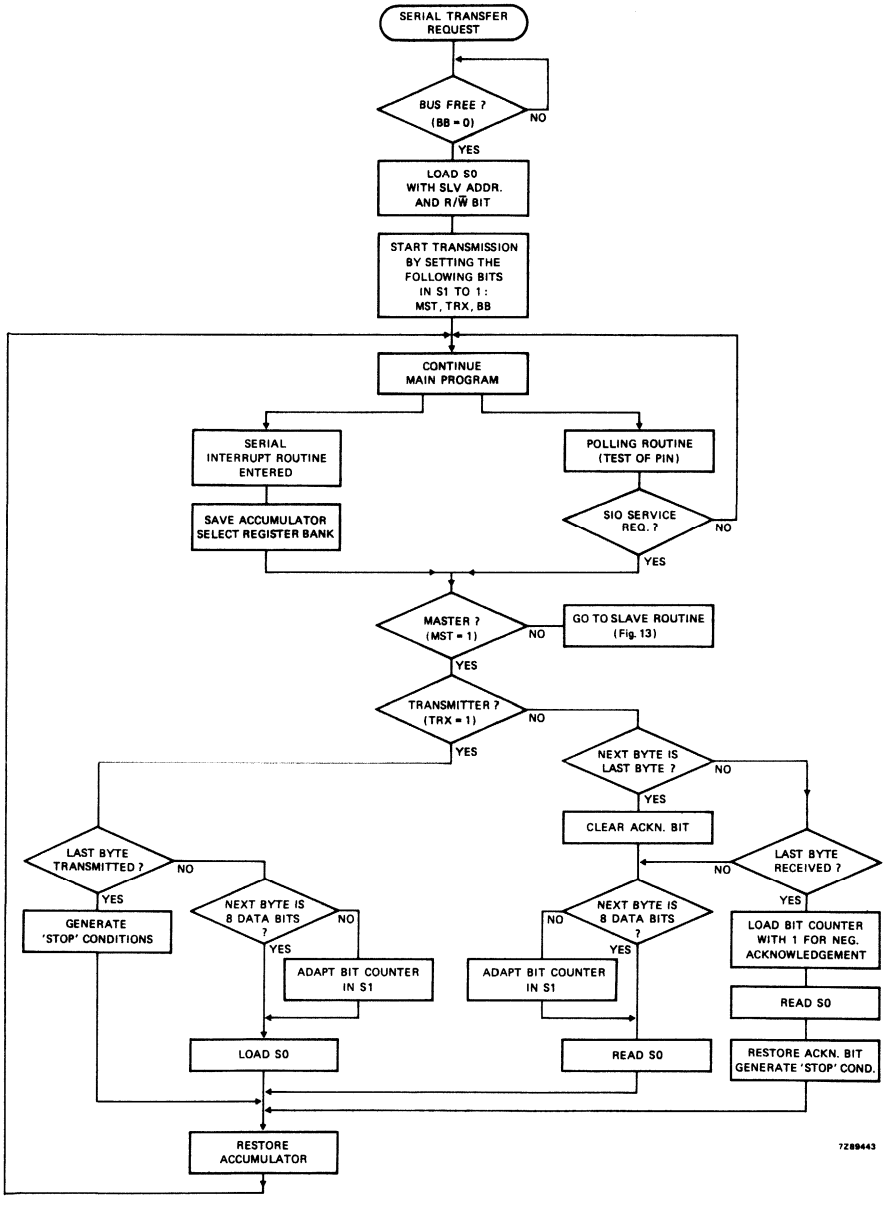


Fig. 15. Flow chart of the 'master' mode after initialization.

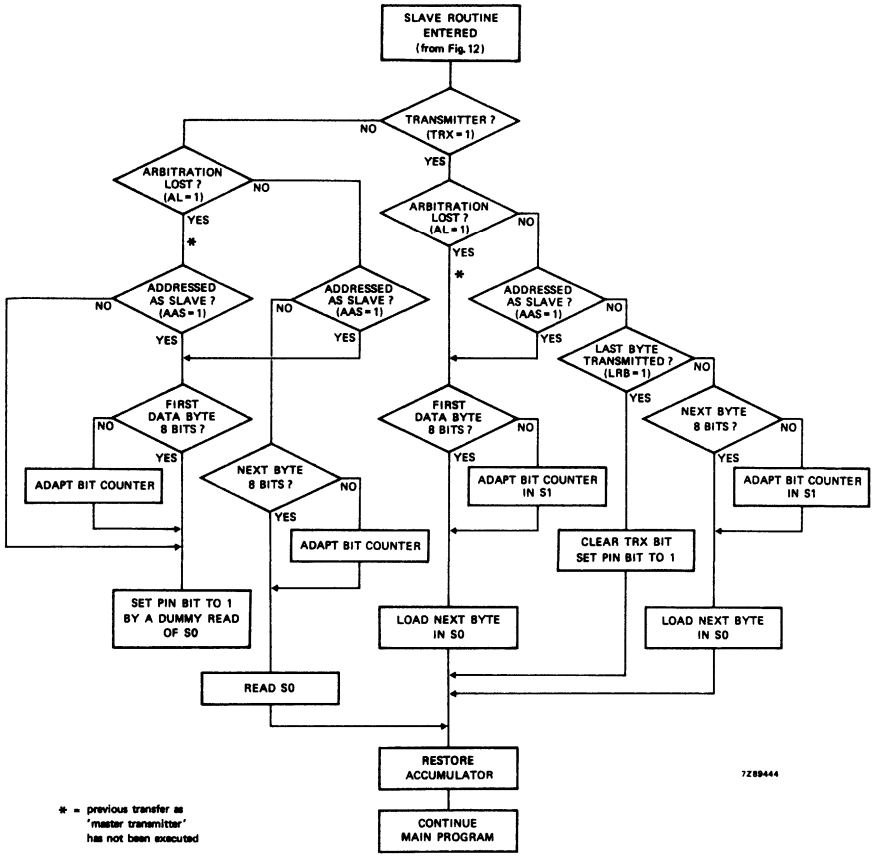


Fig. 16. Flow chart of the 'slave' mode after initialization.

9.9 Solutions to temporary software problems

The following are solutions to possible errors the SIO interface may suffer while operating in the I^2C mode.

1. Problem:

   In multi-master operation it is feared that the S2 register may be corrupted by the clock synchronisation failing.

   Solution:

   This can be solved by rewriting register S2:

```
MOV S2, #H'02'      ; Set ACK mode
MOV S2, #H'42'      ; Set ACK mode and frequency
```

2. Problem:

   When clearing the Bus-busy bit in register S1, not all the bits are set as expected.

   Solution:

   Write to S1 twice:

```
MOV S1, #ESO+PIN
MOV S1, #ESO+PIN
```

* While operating in the 'Master transmitter/receiver' mode, the following subroutine illustrates a possible bus-error correction program incorporating the above macros.

```
MERROR   MOV S2, #H'42'      ; Set acknowledge mode
         MOV R1, A           ; Save error code
         MOV A, S1           ; Get status
         JB2 ADDSLV          ; Jump if addressed as slave
         JB7 I2CSST          ; Jump if MST='1'
         JB3 MERR4           ; Jump if AL='1'
         MOV S2, #H'02'      ; Clear bus method if AL='0'
         MOV S2, #H'42'      ;
         MOV S1, #ESO+PIN    ;
         MOV S1, #ESO+PIN    ;
         JMP MERR1           ;
MERR4    MOV A,SO            ; Clear bus method if AL='1'
         JMP MERR1           ;
I2CSST   MOV S1, #STOPC      ; Send stop condition
MERR1    MOV RO, #BUSHELD    ; Test whether bus was held
         MOV A, @RO          ;
         JNZ MERR2           ; Jump if hold required
         MOV A, R1           ; Get back error code
         XRL A, #LRB         ; Test for no ACK error only
         JZ  MERR3           ; Jump if no ACK
         JMP I2CSUP          ; Try again without decrementing counter
```

```
MERR3     DJNZ R5, I2CSUP      ; Jump if tried ` R5 times (try again)
MERR2     MOV R7, #255         ; Set failed flag
          CLR A                ; Set not held flag
          RET                  ; Leave
ADDSLV    CALL SLAVE           ; Call slave routine
          JMP  MERR1           ; Test whether to try again or give up
```

8. The I^2C bus concept

CONTENTS – THE I²C BUS CONCEPT

1.0 INTRODUCTION

For 8-bit applications, such as those requiring single-chip microcomputers, certain design criteria can be established:

- A complete system usually consists of at least one microcomputer 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.

In order to produce a system to satisfy 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 CONCEPT

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 microcomputer, 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.

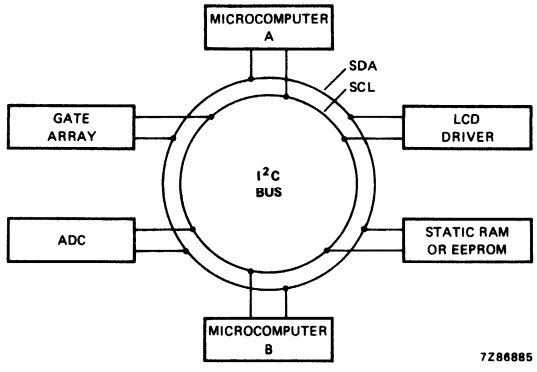


Fig.1 Typical I^2C bus configuration.

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 microcomputers, let's consider the case of a data transfer between two microcomputers 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 microcomputer A wants to send information to microcomputer B
   - microcomputer A (master) addresses microcomputer B (slave)
   - microcomputer A (master transmitter) sends data to microcomputer B (slave receiver)
   - microcomputer A terminates the transfer.

2) If microcomputer A wants to receive information from microcomputer B

   - microcomputer A (master) addresses microcomputer B (slave)
   - microcomputer A (master receiver) receives data from microcomputer B (slave transmitter)
   - microcomputer 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 microcomputer 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 an 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 synchronised 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
Synchronisation	- Procedure to synchronise 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 logical '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.

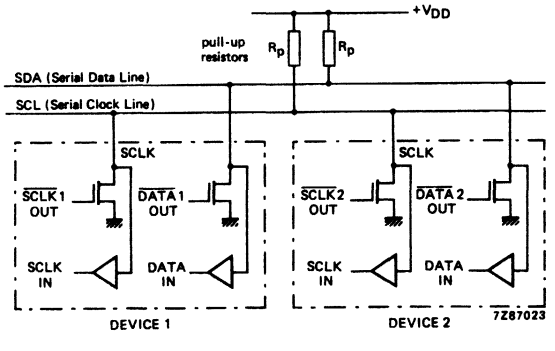


Fig.2 Connection of devices to the I^2C bus.

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).

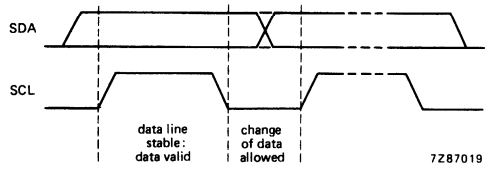


Fig.3 Bit transfer on the I^2C bus.

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.

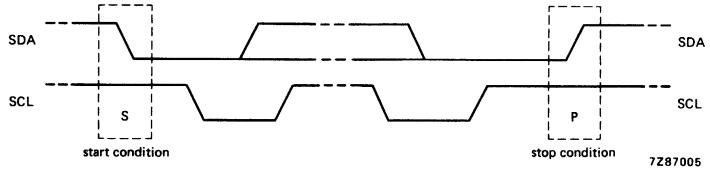


Fig.4 Start and stop conditions.

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 (in section 10.0).

Detection of start and stop conditions by devices connected to the bus is easy if they possess the necessary interfacing hardware. However, microcomputers with no such interface have to sample the SDA line at least twice per clock period in order to sense the transition.

5.0 TRANSFERING 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.

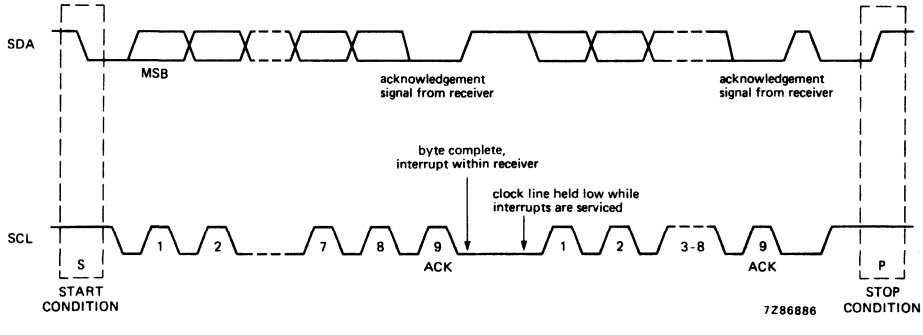


Fig.5 Data transfer on the I^2C bus.

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 case, no acknowledge is generated (see section 8.4).

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 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.

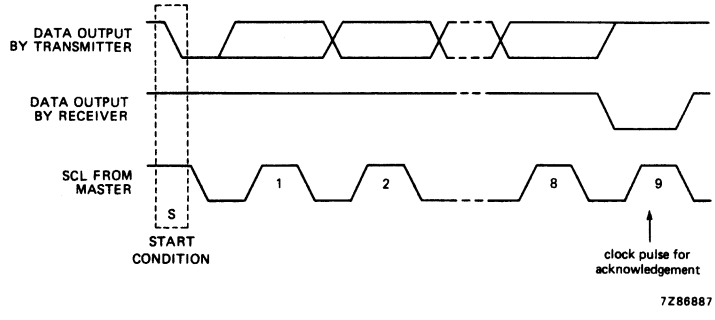


Fig.6 Acknowledge on the I^2C bus.

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 RC-5 or CBUS address - see section 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 acknowledge on the first byte following. The slave leaves the data line HIGH and the master generates the STOP condition.

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 Synchronisation

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 synchronisation 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 synchronised 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.

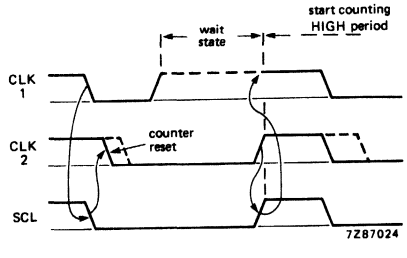


Fig.7 Clock synchronisation during the arbitration procedure.

6.2 Arbitration

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.

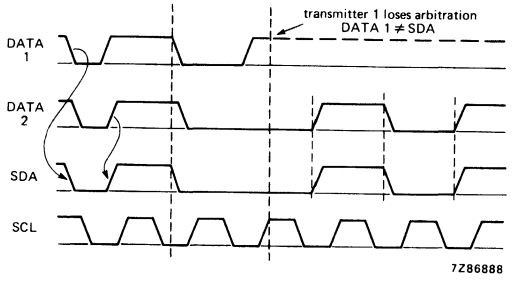


Fig.8 Arbitration procedure of two masters.

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 on 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.

6.3 Use of the clock synchronising mechanism as a handshake

In addition to being used during the arbitration procedure, the clock synchronisation 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 microcomputer 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 (See also Appendix B)

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 ($R/\overline{W}$) - 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, and address another slave without first generating a stop condition. Various combinations of read/write formats are then possible within such a transfer.

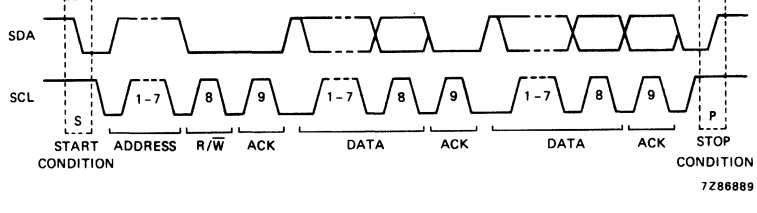
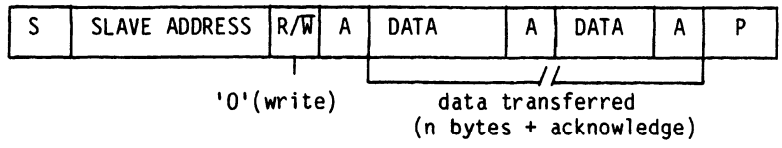


Fig.9 A complete data transfer.

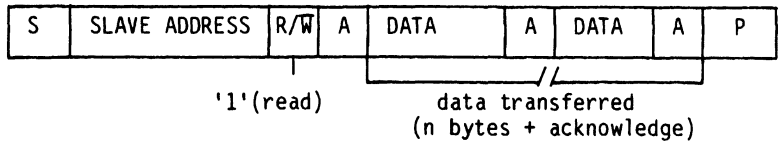
Possible data transfer formats are:

a) Master transmitter transmits to slave receiver. Direction is not changed.



A = Acknowledge
S = Start
P = Stop

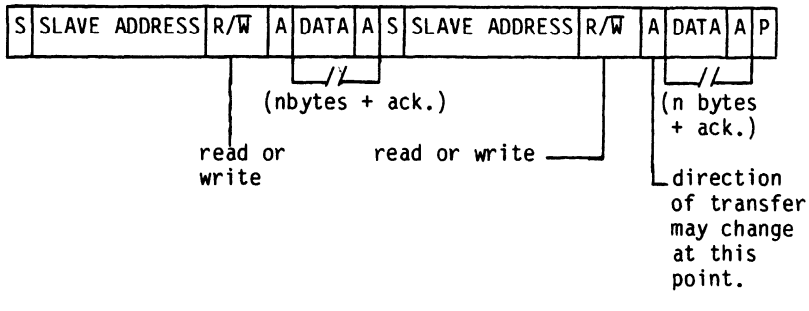
b) Master reads slave immediately after first byte.



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.

   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 as indicated by the [A] blocks 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

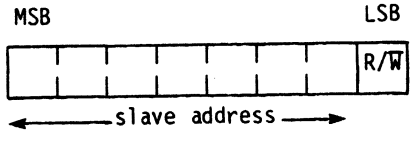


Fig.10 The first byte after the start procedure.

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/$\overline{W}$ 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:

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	
0000	100	X	to be defined
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 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).

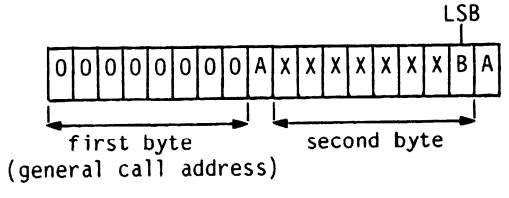


Fig.11 General call address format.

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)

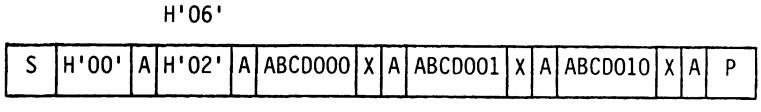


Fig.12 Sequence of a programming master.

00000100 (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).

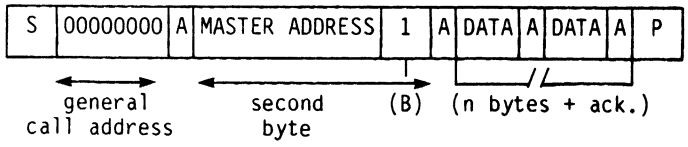


Fig.13 Data transfer from hardware master transmitter.

The seven bits remaining in the second byte contain the device address of the hardware master. This address is recognised by an intelligent device, such as a microcomputer, 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.

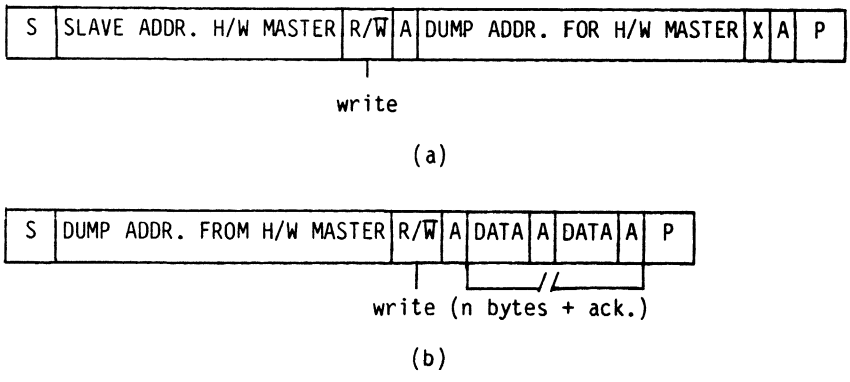


Fig.14 Data transfer of hardware master transmitter capable of dumping data directly to slave devices, (a) configuring master sends dump address to hardware master, (b) hardware master dumps data to selected slave device.

8.1.2 Start byte

Microcomputers can be connected to the I^2C bus in two ways. If an on-chip hardware I^2C bus interface is present, the microcomputer 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 microcomputer 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 microcomputer which relies on software polling.

In this case, data transfer can be preceded by a start procedure which 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

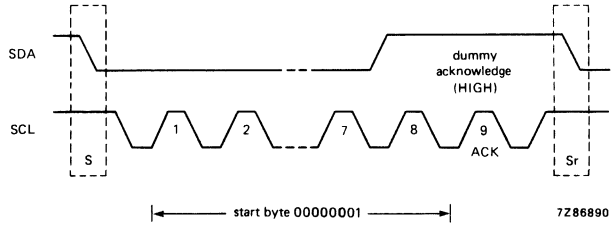


Fig.15 Start byte procedure.

After the start condition S has been transmitted by a master requiring bus access, the start byte (00000001) is transmitted. Another microcomputer 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 microcomputer is then able to switch to a higher sampling rate in order to find the second start condition Sr which is then used for synchronisation.

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).

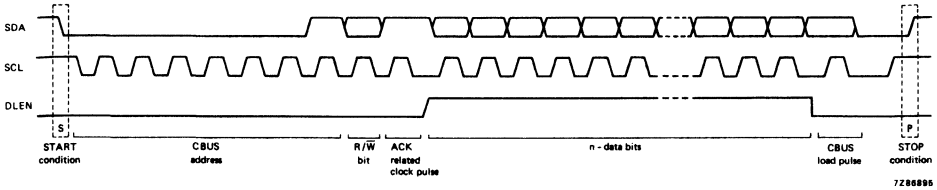


Fig.16 Data format of transmissions with CBUS receiver/transmitter.

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, recognised by all devices. In the 'low speed' mode (see Section 10.0), full 8-bit bytes must always be transmitted and the timing of the DLEN signal adapted.

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 ± 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_{ILmax} = 0,3 V_{DD} (maximum input LOW voltage)
V_{IHmin} = 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 level input current at V_{OLmax} of both the SDA pin and the SCL pin of an I^2C device is -10 μA, including the leakage current of a possible output stage.

The maximum high-level input current at 0,9 V_{DD} of both the SDA pin and SCL pin of an I^2C device is 10 μA, including the leakage current of a possible output stage.

The maximum capacitance of both the SDA pin and the SCL pin of an I^2C device is 10 pF.

Devices with fixed input levels can each have their own power supply of 5 V ± 10%. Pull-up resistors can be connected to any supply (Fig.17).

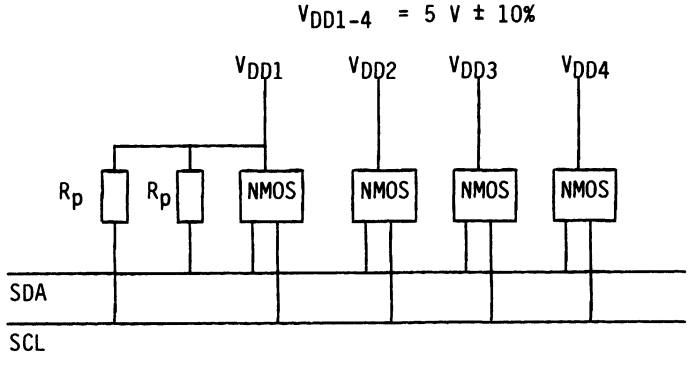


Fig.17 Fixed input level devices connected to the I^2C bus.

However, the devices with input levels related to V_{DD} must have one common supply line to which the pull-up resistor is also connected (Fig.18).

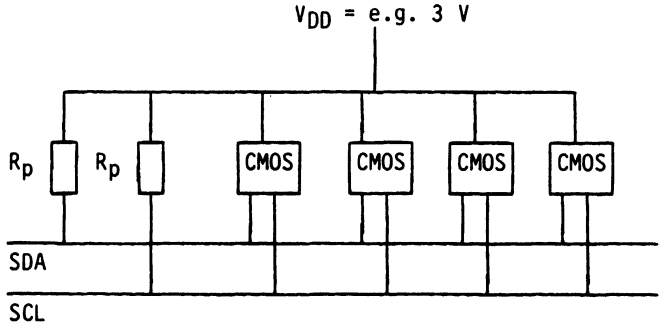


Fig.18 Devices with a wide range of supply voltages connected to the I^2C bus.

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)

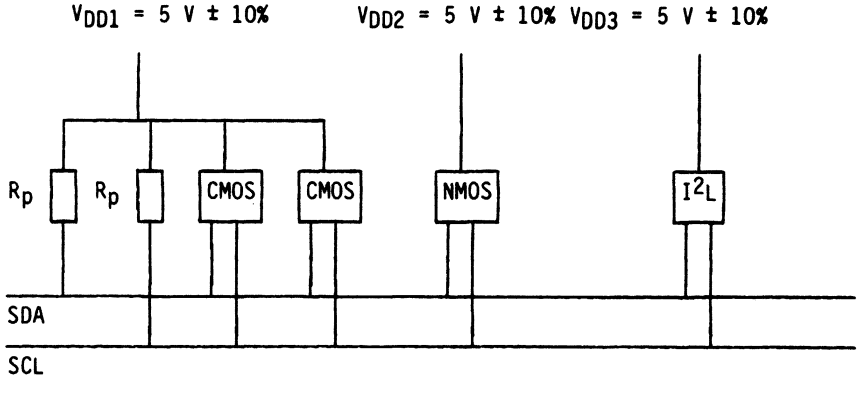


Fig.19 Devices with V_{DD}-related levels mixed with fixed input level devices on the I^2C bus.

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 flash-over protection against high voltage spikes on the SDA and SCL line due to flash-over 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.

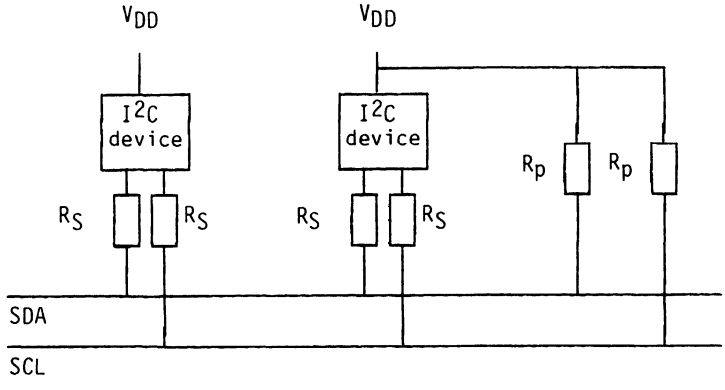


Fig.20 Serial resistors (R_S) for protection against high voltage spikes.

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. Masters in this mode can generate a bus clock with a frequency from 0 up to 100 kHz.

All devices connected to the bus must be able to follow transfers with frequencies up to 100 kHz, either by being able to transmit or receive at that speed or by applying the clock synchronisation procedure which will force the master into a 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 shown in the table following. All timing references are at V_{ILmax} and V_{ILmin}.

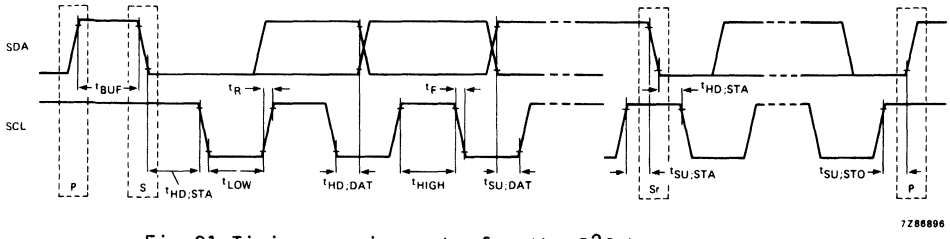


Fig.21 Timing requirements for the I²C bus.

parameter	symbol	min.	max.	units
SCL clock frequency	fSCL	0	100	kHz
Time the bus must be free before a new transmission can start	tBUF	4,7	-	μs
Hold time start condition. After this period the first clock pulse is generated	tHD;STA	4,0	-	μs
The LOW period of the clock	tLOW	4,7	-	μs
The HIGH period of the clock	tHIGH	4,0	-	μs
Set up time for start condition (Only relevant for a repeated start condition)	tSU;STA	4,7	-	μs
Hold time DATA	tHD;DAT			
for CBUS compatible masters (see also NOTE, Section 8.1.3)		5	-	μs
for I²C devices		0*	-	μs
Set-up time DATA	tSU;DAT	250	-	ns
Rise time of both SDA and SCL lines	tR	-	1	μs
Fall time of both SDA and SCL lines	tF	-	300	ns
Set-up time for stop condition	tSU;STO	4,7	-	μs

All values referred to VIH and VIL levels (see section 9.0).

* Note that a transmitter must internally provide at least a hold time to bridge the undefined region (max. 300 ns) of the falling edge of SCL.

11.0 'LOW SPEED' MODE

As explained in section 8.1.2, there is a difference in speed on the I^2C bus between fast hardware devices and the relatively slow microcomputer which relies on software polling. For this reason a 'low speed' mode is available on the I^2C bus to allow these microcomputers to poll the bus less often.

11.1 Start and stop conditions

In the 'low speed' mode, data transfer is preceded by the start procedure of section 8.1.2

11.2 Data format and timing

The bus clock in this mode has a LOW period of 130 μs ± 25 μs and a HIGH period of 390 μs ± 25μs, resulting in a clock frequency of 2 kHz approx. The duty cycle of the clock has this LOW to HIGH ratio to allow for more efficient use of microcomputers without an on-chip hardware I^2C bus interface. In this mode also, data transfer with acknowledge is obligatory. The maximum number of bytes transferred is not limited (Fig.22).

In this mode, a transfer cannot be terminated during the transmission of a byte.

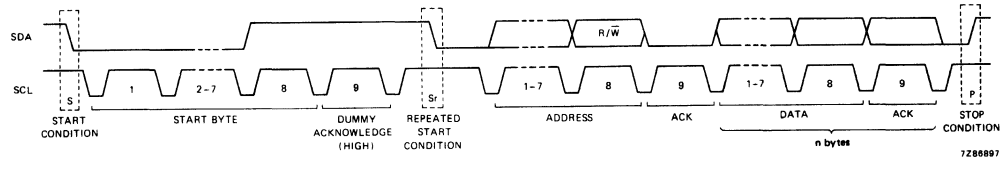


Fig.22 Data transfer 'low speed' mode.

CLOCK	: tLOW = 130 μs ± 25 μs
	: tHIGH = 390 μs ± 25 μs
DUTY CYCLE	: 1:3 LOW to HIGH (Duty cycle of clock generator)
START BYTE	: 0000 0001
MAX. NO. OF BYTES	: UNRESTRICTED
PREMATURE TERMINATION OF TRANSFER	: NOT ALLOWED
ACKNOWLEDGE CLOCK BIT	: ALWAYS PROVIDED
ACKNOWLEDGEMENT OF SLAVES	: OBLIGATORY

The bus is considered busy after the first start condition. It is considered free again one minimum clock LOW period, 105 μs, after the detection of the stop condition. Fig.23 shows the timing requirements in detail, the table following explains the abbreviations.

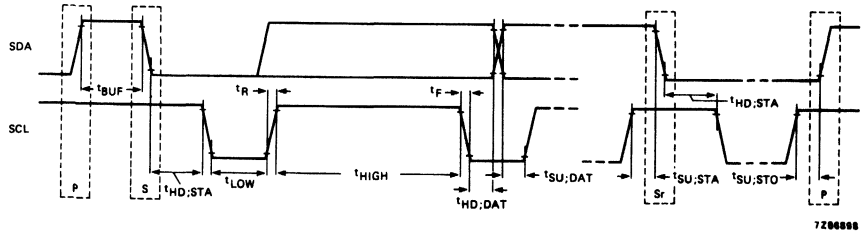


Fig.23 Timing 'low speed' mode.

parameter	symbol	min.	max.	units
Time the bus must be free before a new transmission can start	tBUF	105	-	μs
Hold time start condition. After this period the first clock pulse is generated	tHD;STA	365	-	μs
Hold time (repeated start condition only)	tHD;STA	210	-	μs
The LOW period of the clock	tLOW	105	155	μs
The HIGH period of the clock	tHIGH	365	415	μs
Set up time for start condition (Only relevant for a repeated start condition)	tSU;STA	105	155	μs
Hold time DATA	tHD;DAT			
for CBUS compatible masters (see also NOTE, Section 8.1.3)		5	-	μs
for I^2C devices		0*	-	μs
Set-up time DATA	tSU;DAT	250	-	ns
Rise time of both SDA and SCL lines	tR	-	1	μs
Fall time of both SDA and SCL lines	tF	-	300	ns
Set-up time for stop condition	tSU;STO	105	155	μs

All values referred to VIH and VIL levels (see section 9.0).

* Note that a transmitter must internally provide at least a hold time to bridge the undefined region (max. 300 ns) of the falling edge of SCL.

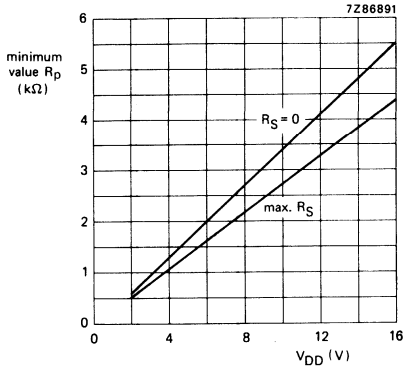
APPENDIX A

Maximum and minimum values of the pull-up resistors R_p and series resistors R_S (See Fig.20).

In a I^2C bus system these values depend on the following parameters:

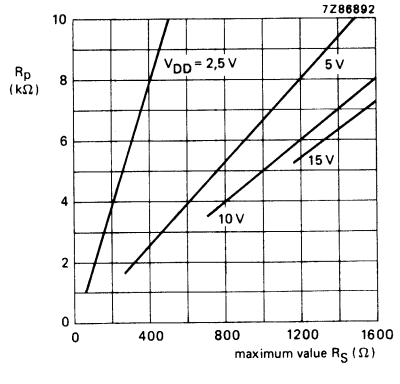
- Supply voltage
- Bus capacitance
- Number of devices (input current + leakage current)

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. In graph 1, V_{DD} against $R_{p\ min}$ is shown.



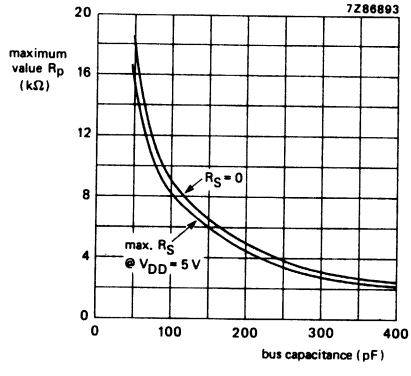
The desired noise margin of 0,1 V_{DD} for the low level limits the maximum value of R_S.

In graph 2, $R_{S\ max}$ against R_p is shown.



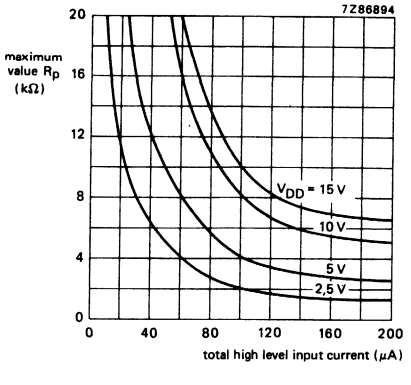
2) The bus capacitance is the total capacitance of wire, connections and pins. This capacitance limits the maximum value of R_p due to the specified rise time of 1 μsec.

In graph 3, the bus capacitance - $R_{p\ max}$ relationship is shown.



3) The maximum high-level input current of each input/output connection has a specified value of max. 10 μA. Due 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 is dependent on V_{DD}.

In graph 4 the total high-level input current - $R_{p\ max}$ relationship is shown.



APPENDIX B

Note to chapter 7.

In an I^2C bus system, the arbitration procedure between two or more masters may continue without decision, until a repeated START condition from one of the masters is transmitted. It is essential therefore, that each of the masters must contain this condition in its format at exactly the same location.

In an I^2C bus system, again the arbitration procedure between two or more masters may continue without decision, until a STOP condition from one of the masters is transmitted. It is essential therefore, that each of the masters must contain this condition in its format at exactly the same location.

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 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 appropriatly.

The format of the chapter follows the instruction set of the MAB8048 arranged alphabetically. It is suplemented with additional instructions not implemented on the 8048 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. For the 8051/C51 instruction set, see data sheet or seperate user manual section.

INSTRUCTION SET

Symbols and Abbreviations Used

A	Accumulator
AC	Auxillary Carry
addr	12-bit Program Memory Address
Bd	Bit Designator (b=0-7)
BUS	BUS Port
C	Carry (CY)
CLK	Clock
CNT	Event Counter
D	Mnemonic for 4-Bit Digit (Nibble)
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. 8048 only)
8048 refers to MAB8048, PCB 80C49 family of microcomputers
84XX " " MAB84XX
84CXX " " PCB84CXX

*) is used in this section to signify:
bits 0-6 in the 84XX, 84X1 family incl. 8422/42, 80XX family
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 RO,#H'3F'        MOVE '3F' HEX TO REG 0
      ANL A,@RO            '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 Pp,#data — Logical AND Port 1-2 With Immediate Mask (Not in 8021)

Opcodes 98H to 9AH

1 0 0 1 0 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 84XX,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 Immediate Mask (only in 8021, 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

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 84XX, 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 84XX, 84CXX) is determined by the most recent SEL MB instruction.

Execution continues at the instruction following the CALL upon return from the subroutine.

$((SP)) \leftarrow (PC), (PSW_{4-7})$
$(SP) \leftarrow (SP)+1$
$(PC_{8-10}) \leftarrow (addr_{8-10})$
$(PC_{0-7}) \leftarrow addr_{0-7}$
$(PC_{11}) \leftarrow MBFF_0$

$((SP)) \leftarrow (PC_{0-7})$
$((SP)+1) \leftarrow (PC_{8-12}),$
$(PSW\ 4,6,7)$
$(SP \leftarrow (SP)+1$
$(PC_{8-10}) \leftarrow addr_{8-10}$ 8400
$(PC_{0-7}) \leftarrow addr_{0-7}$ 8500
$(PC_{11-12}) \leftarrow MBFF_{0-1}$

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
        MOV RO,#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'
        ADD A,R3            ADD REG 3 TO ACC
        ADD A,R4            ADD REG 4 TO ACC
        CALL SUBTOT         CALL SUBROUTINE 'SUBTOT'
        ADD A,R5            ADD REG 5 TO ACC
        ADD A,R6            ADD REG 6 TO ACC
        CALL SUBTOT         CALL SUBROUTINE 'SUBTOT'

SUBTOT MOV @RO,A            MOVE CONTENTS OF ACC TO
*                           LOCATION ADDRESSED BY
*                           REG 0
        INC RO              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 R0,#H'3F'          MOVE '3F' HEX TO REG 0
        MOV A,@R0              MOVE CONTENTS OF LOCATION
*                              63 TO ACC
        DEC A                  DECREMENT ACC
        MOV @R0,A              MOVE CONTENTS OF ACC TO
*                              LOCATION 63 IN INTERNAL
*                              MEMORY
```

DEC @Rr — Decrement Data Memory Location (8422/42, 84XX, 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 (Not in 8021)

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 (Not in 8021)

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 8021,8048 or 84CXX)

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 WAIT mode (84CXX), when the SIO interrupt is disabled, it cannot be restarted by this interrupt.

(SIFF) ← 0

DIS TCNTI — Disable Timer/Counter Interrupt (Not in 8021)

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 8021 8048 or 84CXX)

Opcode E0H, 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 R0,#32           MOVE '32' DEC TO ADDRESS
*                           REG 0
       MOV @R0,#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 @R0,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 (Not in 8021)

Opcode 05H

0 0 0 0 0 1 0 1

External interrupts are enabled. A LOW signal (8048,84CXX) or a HIGH-to-LOW transition (84XX, 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, 84XX 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 WAIT mode.

(SIFF) ←1

EN TCNTI — Enable Timer/Counter Interrupt (Not in 8021)

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 test 0 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

IDL — Select Idle operation (84CXX, 80C48, C49, C31)

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.

IN A,Pp — Input Port or Data to Accumulator

Opcodes 08H to 0AH (for 84XX, 84CXX and 8021)
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 8021, 84XX, 84CXX IN A,P2 inputs P20-P23 to A0-A3

while A4-A7 is set to zero. In the 84CXX, IN A, P2 transfers data on P20-P24 to A0-A4 while A5-A7 is set to zero.

(A) ← (Pp) p=1-2 (p=0-2 for 84XX, 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
```

IN A,P0 — Input of Port 0 Data to Accumulator (84XX only)

Opcode 08H, 09H

Same as INS A,BUS except no ~~RD~~ pulse generated

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.

$(A) \leftarrow (BUS)$

```
Example:        INS A,BUS           INPUT BUS CONTENTS
                *                   TO ACC
```

JBb address — Jump If Accumulator Bit is Set (Not in 8021)

Opcode 12H

b_2 b_2 b_0 1 0 0 p p 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.

$(PC_{0-7}) \leftarrow$ addr	If Bb=1
$(PC) \leftarrow (PC)+2$	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.

$(PC_{0-7}) \leftarrow$ addr	If C=1
$(PC) \leftarrow (PC)+2$	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

1 0 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 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

0 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 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, C4, E4

a_{10} 0_9 0_8 0 0 1 0 0	a_7 a_6 a_5 a_4 a_3 a_2

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 84XX, 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
$(PC_{11}) \leftarrow$ MBFF	$(PC_{11,12}) \leftarrow$ MBFF0-1	

Example:

```
        JMB 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.

$(PC_{0-7}) \leftarrow$ addr — If C=0
$(PC) \leftarrow (PC)+2$ — 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).

$(PC_{0-7}) \leftarrow$ addr — If I=0
$(PC) \leftarrow (PC)+2$ — 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, 8021,)

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 (Not in 8021)

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

0 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 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

1 0 0 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 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

0 0 0 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 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 (Not in 8021, 84CXX)

Opcode 36H

0 0 1 1 0 1 1 0	a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

(See Instruction JI)

JT1 address — Jump If Test 1 is HIGH

Opcode 56H

0 1 0 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 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:        JT 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,#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) ← data

```
Example:        MOV A,#H'A3'        MOV 'A3' HEX TO ACC
```

MOV A,PSW — Move PSW Contents to Accumulator (Not in 8021)

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 F0H, 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 0CH, 0DH (84XX, 84CXX only not 8422/42)

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,S0        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 (84XX, 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 Rr,A — Move Accumulator Contents to Register

Opcode A8H to AFH

1 0 1 0 1 r r r

The contents of the accumulator are moved to the working register 'r'.

(Rr) ← (A) r=0-7

Example:

```
        MOV RO,A          MOVE CONTENTS OF ACC TO
*                         REG 0
```

MOV Rr,#data — Move Accumulator Contents to Register

Opcode B8H to BFH

1 0 1 1 1 r r 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 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 AOH, 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 (84XX, 84CXX only not 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 (84XX, 84CXX only not 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 (8021 and 8048 only)

Opcode OCF to OFH

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 (Not in 84XX)

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
*                               118 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 199 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 3
```

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 RO,#H'3F'       MOVE '3F' HEX TO REG 0
        ORL A,@RO           '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 Pp,#data — Logical OR Port With Immediate Mask (Not in 8021)

Opcodes 89H to 8A for 8048
88H to 8A for 84XX, 84CXX

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 84XX, 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 84XX 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 P0,A — Output Accumulator Data to Port 0 (8021, 84XX only)

OUTL BUS,A — Output Accumulator Data to BUS (8048 only)

Opcode 02H (8048), 39H (8021), 38H (84XX), 90H (84XX, 8021)

0 0 0 0	0 0 1 0

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          OUTPUT ACC CONTENTS TO BUS
```

OUTL Pp,A — Output Accumulator Data to Port

Opcodes 39H, 3AH for 8048
38H, 3AH for 84XX

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 (Not in 8021)

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 are not restored (bits 4,6,7 in 84XX, 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
(PC) ← ((SP))
(PSW 4-7) ← ((SP))

(SP) ← (SP)-1
(PC) ← ((SP)) - 84XX
$(PSW_{4,6,7})$ ← ((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 in 8021)

Opcode E5H

1 1 1 0 0 1 0 1

PC bit 11 (and bit 12 for 84XX, 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]- 84XX, 84CXX

SEL MB1

Select Memory Bank 1 (Not in 8021)

Opcode F5

1 1 1 1 0 1 0 1

PC bit 11 (bit 12 also set to zero for 84XX, 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 `- 0,0]- 84XX, 84CXX

SEL MB2 — Select Memory Bank 2 (84XX, 84CXX only)

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) ←— 10

SEL MB3 — Select Memory Bank 3 (84XX, 84CXX only)

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) ←— 11

SEL RB0 — Select Register Bank 0 (Not in 8021)

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 (Not in 8021)

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 properly.

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 input pin LOW, an interrupt sequence is enabled. In this case, the processor resumes the normal program sequence after returning from the interrupt routine, as in the normal mode. Otherwise the processor continues the normal program sequence, executing the instruction following the STOP instruction.

Note: The processor is restarted by a LOW level applied at the INT/TO pin, and not by a HIGH-to-LOW transition as in a normal interrupt mechanism.

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 HIH-to-LOW (LOW-to-HIGH for 84XX, 84CXX) transition on the T1 pin.

Example:

Initialize and start event counter. Assume overflow is desired with first T1 transition.

```
EN    TCNT1             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 84XX-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.

$(A_{4-7}) \leftarrow (A_{0-3})$

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
```

IDL — Idle Processor (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.

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) ⟵ (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 RO,#52          MOVE 52 DEC TO ADDRESS
*                           REG 0
        XCH A,@RO           EXCHANGE CONTENTS OF ACC
*                           AND LOCATION 52
        DEC A               DECREMENT ACC CONTENTS
        XCH A,@RO           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 to 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

CONTENTS – SOFTWARE EXAMPLES

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 MAB8400 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 84XX and 8048 operate using the I^2C bus at system level. Each example is taken from a Philips Microcomputer Development System (PMDS II) using 8400 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 84XX 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 E80C           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 83             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 HER
000018                       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 84XX FAMILY

The serial I/O equate list given below, is used throughout the programs in section 3.0 to 3.2

```
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 (MAB84XX - SAA1300)

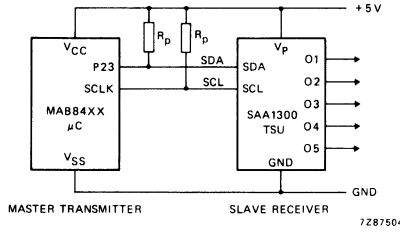


Fig. 3.1 Block diagram of MAB84XX - SAA1300 configuration.

INITIALIZATION:

Normally the serial I/O (SIO) of the MAB84XX family has to be initialized as follows:

- program the SCL frequency in reg. S2
- select the "with acknowledge" mode in reg. S2
- load its own slave address in reg. SO'
- enable the SIO logic with the ESO bit in reg. S1
- select the slave receiver mode in reg. 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 MAB84XX 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.

Symbol definition

```
                          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
000108 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 - must be loaded into the SIO data shift register S0. The transmission 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 acheivedwith the "+" statement. In the SIO equate list, each SIO status bit is defined individually.

The MAB84XX 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 succeeded.

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.

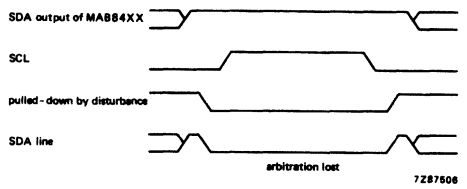


Fig. 3.3

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.

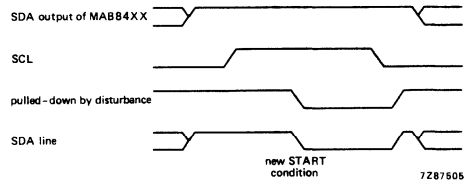


Fig. 3.4

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.

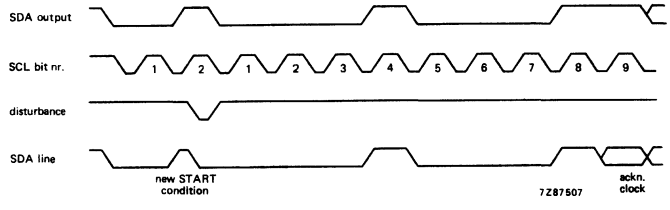


Fig. 3.5

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 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.

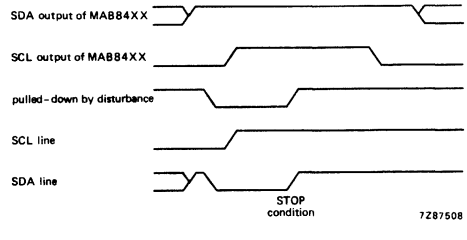


Fig. 3.6

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.

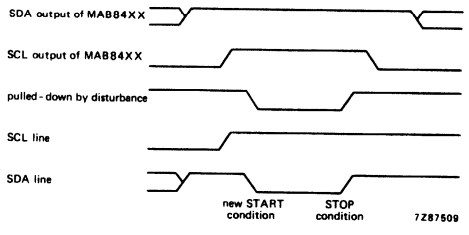


Fig. 3.7

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*.

By testing the contents of the accumulator, the main program can repeat the transmission until it succeeds, if necessary.

*In the second case the bus is released by returning to the slave mode.

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.

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 BI
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 routine has byte economy when compared 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 generate a STOP condition if a NO 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 (MAB84XX - SAB3028)

These routines can only be used if no other masters are connected to the I²C BUS.

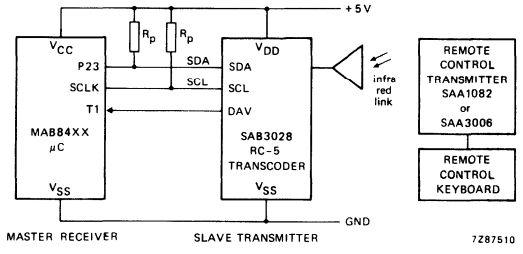


Fig. 3.8 Block diagram of MAB84XX - SAB3028 configuration.

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 I²C 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 ≠ 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 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+BC0 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 *
000158              53   163         END
```

3.1.3 Master transmitter/receiver routine including general call reset (MAB84XX - SAB3035/36/37)

These routines can only be used if no other masters are connected to the I^2C BUS.

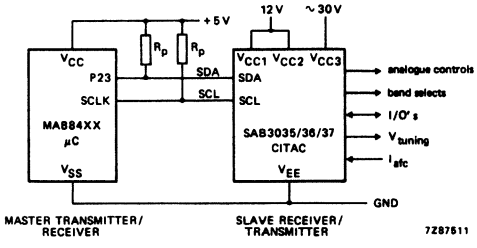


Fig. 3.10 Block diagram MAB84XX - SAB3035/36/37 CITAC configuration.

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 *
```

MASTER 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+BC0 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 (MAB84XX - PCD8571)

These routines can only be used if no other masters are connected to the I^2C BUS.

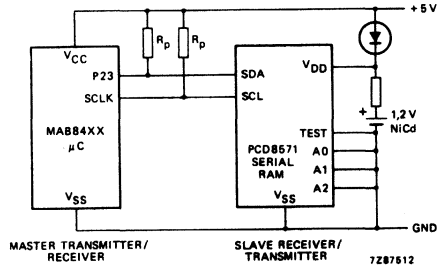


Fig. 3.14 Block diagram MAB84XX - PCD8571 serial RAM configuration.

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 MAB84XX 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 MAB84XX 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).

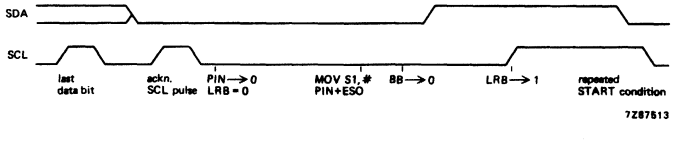


Fig. 3.17 Testing status bits BB and LRB to see whether the SCL line has been released

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 BI
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 (MAB84XX - 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.

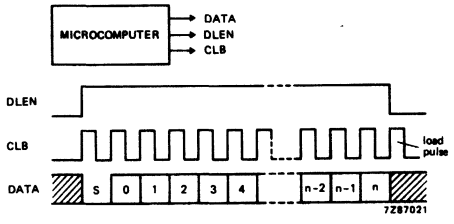


Fig. 3.18 The CBUS structure.

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.

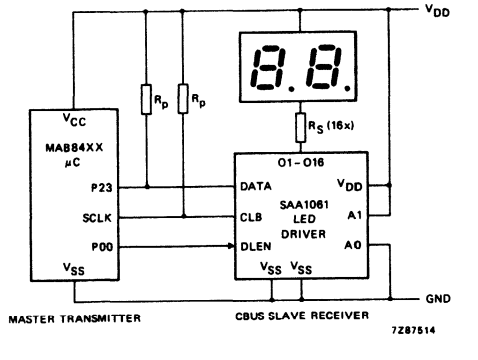


Fig. 3.19 Block diagram MAB84XX - SAA1061 LED driver configuration.

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:

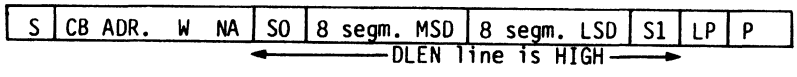


Fig. 3.20 CBUS format for writing to an LED display driver

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 SO 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 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 (MAB84XX - 2x 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.

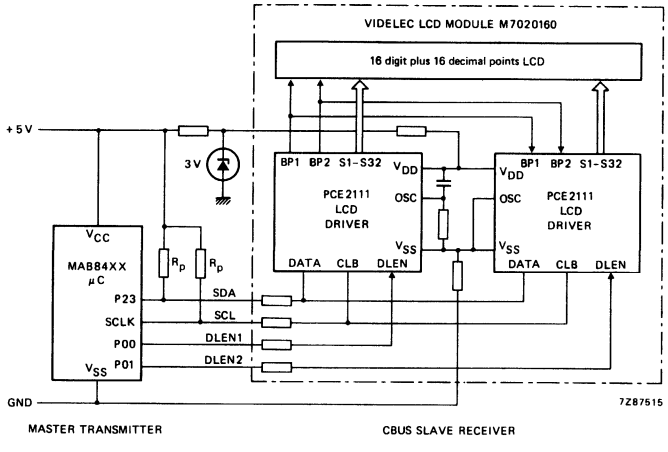


Fig. 3.21 Block diagram MAB84XX - With two PCE2111 LCD drivers configuration.

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:

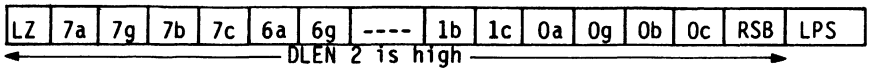


Fig. 3.25 Format of register 2B.

Register 2A is data to register A of PCE2111-2.
Format of register 2A is shown in Fig. 3.26:

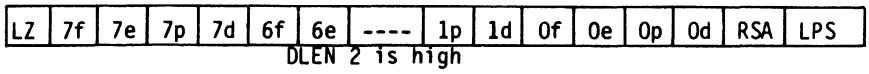


Fig. 3.26 Format of register 2A.

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 L

```
                                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 80                       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.

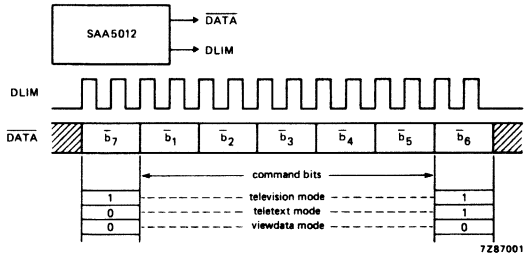


Fig. 3.27 The 2-line IBUS structure

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

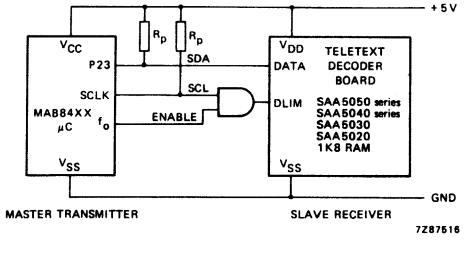


Fig. 3.28 Block diagram MAB84XX - Teletex decoder configuration.

```
                                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 BD00          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 FORMAT 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 = $\overline{B7}$ $\overline{B7}$ $\overline{B1}$ $\overline{B1}$ $\overline{B2}$ $\overline{B2}$ $\overline{B3}$ $\overline{B3}$
R3 = $\overline{B4}$ $\overline{B4}$ $\overline{B5}$ $\overline{B5}$ $\overline{B6}$ $\overline{B6}$ 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 (MAB84XX - master)

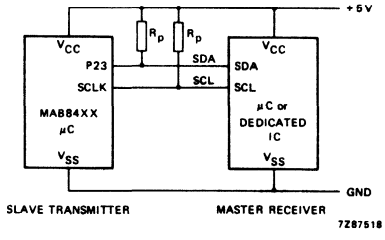


Fig. 3.30 Block diagram MAB84XX - master configuration.

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'18'   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+ES0     INITALIZE SIO STATUS
00010C 05            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 microcomputer 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 microcomputer 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 S0. 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 H'FF'.

```
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 9D18          59     161 STERR1 MOV     S1,#PIN+ESO      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)

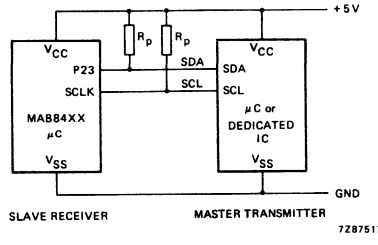


Fig. 3.32 Block diagram MAB84XX - master configuration.

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 subroutine
```

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 *
000100 00           11  84 MAIN   MOV     A,S1
000100 B820         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 9D18         36 117        MOV     S1,#PIN+ESO     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 (MAB84XX – master)

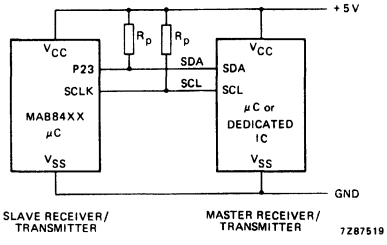


Fig. 3.34 Block diagram MAB84XX - master configuration.

```
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 05            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 Salve 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 devices own slave address
GC ADR	is the general call address
W	is the read/write bit in write state
R	is the read/write bit in read state
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.

```
000100 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

The following is an example of an I^2C Multi-master routine. The examples shown are divided into two sets of routines:

1) The first set is the basic Multi-master communication routine used by successful masters and called for I^2C Bus communications

2) The second set of routines 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.

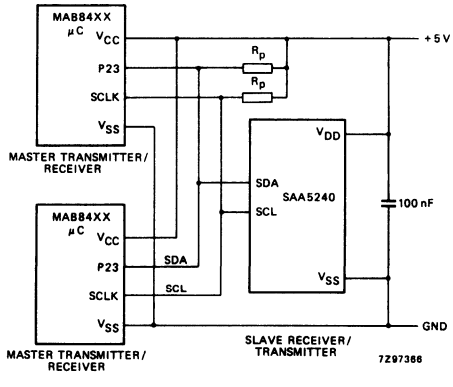


Fig. 3.38 Block diagram of multi-master MAB84XX - SAA5240 CCT

All equate listings are shown at the rear of section 3.3

3.3.1 I^2C 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 enviroment. 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-PASCAL description preceeds the program.

```
12          ENTRY   I2C,SIOIN                       Bus Comms
13          ENTRY   ICCSB                           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 + ESO 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
```

```
                                109
                                110 I2CSEC  RSECT   ROM,INPAGE                      eject
                                111         PAGE    256
                                112 *
                                113 *-------------------------------
                                114 * Save R0-R6 ; Not R7
                                115 *-------------------------------
                                116 *
000000 AF              1        117 I2C     MOV     R7,A                    R7 = address & R/W
000001 F8              2        118         MOV     A,R0                    Hold R0
000002 B868            3        119         MOV     R0,#SAVRG0              Buffer pointer
000004 A0              4        120         MOV     @R0,A                   Save R0 - SAVRG0
000005 F9              5        121         MOV     A,R1                    Get R1
000006 B906            6        122         MOV     R1,#6                   Set count to 6 regs
                                123 *
                                124 *-------------------------------
                                125 * Save R1 1st time in, and then
                                126 * R6 to R2
                                127 *-------------------------------
                                128 *
000008 18              7        129 ISRL1   INC     R0                      Next SAVRG
000009 A0              8        130         MOV     @R0,A                   Save Rn (1,6-2)
00000A F1              9        131         MOV     A,@R1                   Get next reg
00000B E908      R    10        132         DJNZ    R1,ISRL1                Do 6 times
                                133 *
                                134 *-------------------------------
                                135 * Set up data used for restarts
                                136 *-------------------------------
                                137 *
00000D B87F            11       138         MOV     R0,#TEMP0
00000F F0              12       139         MOV     A,@R0
000010 AE              13       140         MOV     R6,A                    R6 = Hold & no. of bytes
000011 BD05            14       141         MOV     R5,#5                   R5 = No. of retries
                                142 *
                                143 *-------------------------------
                                144 * Set up data for each try
                                145 *-------------------------------
                                146 *
000013 FE              15       147 I2CSUP  MOV     A,R6
000014 537F            16       148         ANL     A,#.not.HLD
000016 AC              17       149         MOV     R4,A                    R4 = No. of bytes
000017 B87E            18       150         MOV     R0,#TEMP1               R0 -> Start of buffer for data

                                153 *-------------------------------
                                154 * Test for bus being free
                                155 *-------------------------------
                                156 *
000019 9E42            19       157         MOV     S2,#H'42'               Set ack mode
00001B B920            20       158         MOV     R1,#BUSHLD              Test whether bus was held
00001D F1              21       159         MOV     A,@R1
00001E 9622      R     22       160         JNZ     I2CSFR                  Jump if was held
000020 1400      R     23       161 I2C1    CALL    FREBUS                  Free bus if busy
                                162 *
                                163 *-------------------------------
                                164 * Send slave address & start
                                165 * condition. Test arbitration.
                                166 *-------------------------------
                                167 *
000022 95              24       168 I2CSFR  DIS     SI                      Disable SIO interrupt
000023 FF              25       169         MOV     A,R7                    Get device address
000024 3C              26       170         MOV     S0,A                    Send address
000025 9DF8            27       171         MOV     S1,#STARTC              Send start condition
000027 125B      R     28       172 I2C2    JB0     I2CRDT                  Jump if reading
000029 1400      R     29       173         CALL    TBSMT                   Test bus status
00002B 968E      R     30       174         JNZ     MERROR                  JMP if error
                                175 *
                                176 *-------------------------------
                                177 * Send data block
                                178 *-------------------------------
                                179 *
00002D F0              31       180 I2CWR   MOV     A,@R0                   Get data byte
00002E 3C              32       181         MOV     S0,A                    Send byte
00002F C8              33       182         DEC     R0                      Point to next data byte
000030 1400      R     34       183 I2CWR1  CALL    TBSMT                   Test bus status
000032 968E      R     35       184         JNZ     MERROR                  JMP if error
000034 EC2D      R     36       185         DJNZ    R4,I2CWR                Loop until all data bytes gone

                                188 *-------------------------------
                                189 * Comms successful.
                                190 * Control bus holding.
                                191 *-------------------------------
                                192 *
000036 BF00            37       193 I2CCHL  MOV     R7,#0                   Set successful flag
000038 FE              38       194         MOV     A,R6                    Get byte with hold flag
000039 5380            39       195         ANL     A,#HLD                  Keep hold bit only
00003B 9641      R     40       196         JNZ     I2CBHG                  Skip bus closing if hold needed
                                197 *
                                198 *-------------------------------
                                199 * Close bus down
                                200 *-------------------------------
                                201 *
00003D 9DD8            41       202 I2CEX   MOV     S1,#STOPC               Send stop condition
00003F 0443      R     42       203         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

" = 0, Test MST bit

2) MST = 1, Send a stop condition (hardware reset MST=0)

If MST=0, then the program will 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 528A      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 9D08         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
000081 0413      R  114          354         JMP     I2CSUP             Try again without decrementing count
000083 ED13      R  115          355 MERR3   DJNZ    R5,I2CSUP          Jump if tried < R5 times (try again)
000085 BFFF         116          356 MERR2   MOV     R7,#255            Set failed flag
000087 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 *
```

I^2C 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         395 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 E903      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 776 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 RB
```

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-PASCAL description of the routine preceeds 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'
000002 0D            160        592         MOV     A,S1
000003 B206      R   161        593         JB5     SLABB
000005 83            162        594         RET
000006 520A      R   163        595 SLABB   JB2     SLABBL
000008 0C            164        596         MOV     A,S0
000009 83            165        597         RET
                                598 *
00000A 0D            166        599 SLABBL  MOV     A,S1
00000B D227      R   167        600         JB6     SLATX
00000D 5220      R   168        601         JB2     SLAASR
00000F 321D      R   169        602         JB1     SLARX1
000011 FC            170        603         MOV     A,R4
000012 C61D      R   171        604         JZ      SLARX1
000014 0C            172        605         MOV     A,S0
000015 A1            173        606         MOV     @R1,A
000016 CC            174        607         DEC     R4
000017 C9            175        608         DEC     R1
000018 FC            176        609         MOV     A,R4
000019 9635      R   177        610         JNZ     SLAEVL
00001B 0400      R   178        611         JMP     SLAALL
                                612 *
00001D 0C            179        613 SLARX1  MOV     A,S0
00001E 0435      R   180        614         JMP     SLAEVL
                                615 *
000020 0C            181        616 SLAASR  MOV     A,S0
000021 BC03          182        617         MOV     R4,#IBUFL
000023 B972          183        618         MOV     R1,#RXBUF0
000025 0435      R   184        619         JMP     SLAEVL
                                620 *
000027 1232      R   185        621 SLATX   JB0     SLATX1
000029 B92B          186        622         MOV     R1,#TXTREQ
00002B F1            187        623         MOV     A,@R1
00002C 3C            188        624         MOV     S0,A
00002D 5305          189        625         ANL     A,#SGNPRE+TXONLY
00002F A1            190        626         MOV     @R1,A
000030 0435      R   191        627         JMP     SLAEVL
                                628 *
000032 9D28          192        629 SLATX1  MOV     S1,#BB+ESO
000034 0C            193        630         MOV     A,S0
                                631 *
000035 0D            194        632 SLAEVL  MOV     A,S1
000036 37            195        633         CPL     A
000037 B23D      R   196        634         JB5     SLAEXI
000039 920A      R   197        635         JB4     SLABBL
00003B 0435      R   198        636         JMP     SLAEVL
                                637 *
00003D B904          199        638 SLAEXI  MOV     R1,#AAS
00003F 83            200        639         RET
```

```
Set ack mode
Read status
if (BB=0) then
   Self generated interrupt
else if (AAS=0) then
   Release bus

else repeat
   Read status
   if (TRX=0) {receiver} then
      if (AABS=0) then
         if (AD0=0) and

            and (no. wanted > 0) the
               Read byte
               Save byte
               Decrement no. wanted
               Update destination

               if (No. wanted = 0) t
                  Store packet
         else
            Read byte to release SCL

      else
         Read byte to start receptio
         Initialise no. wanted
         Initialise destination

   else {tranBsmitter}
      if (LRB=0) {ack} then


         Send byte from TXTREQ
         Clear transient bits


      else {no ack}
         Quit Busing bus

   repeat
      Read status

   until (BB=0) or
         (PIN=0)

until (BB=0)
Set error code used in I2C to AAS
```

3.3.2 I²C system level utility routines

These utility routines are representitive of I²C assembler programs at system level. Taken as examples here are routines for SAA5042 CCT (computer controlled teletext see figure 3.38) 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 I²C 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 I^2C BUFFER and POINTER FOR CCT (SAA5042)

This routine sets up the data buffer used by the I^2C 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) REGISTER 1

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 B005          245          784         MOV     @R0,#5              Write no hold & 5 byte msg
000004 C8            246          785         DEC     R0
000005 B004          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 & 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 A0            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 B00B          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 B00B          322      1005         MOV     @R0,#11                CCT sub-addr
00001C C8            323      1006         DEC     R0
00001D B00B          324      1007         MOV     @R0,#H'0B'             Open Box
00001F C8            325      1008         DEC     R0
000020 B00B          326      1009         MOV     @R0,#H'0B'             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 seperate 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 8E00      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 8848      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 1430 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
00000B 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 F8            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 F8            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'28'          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'
*
```

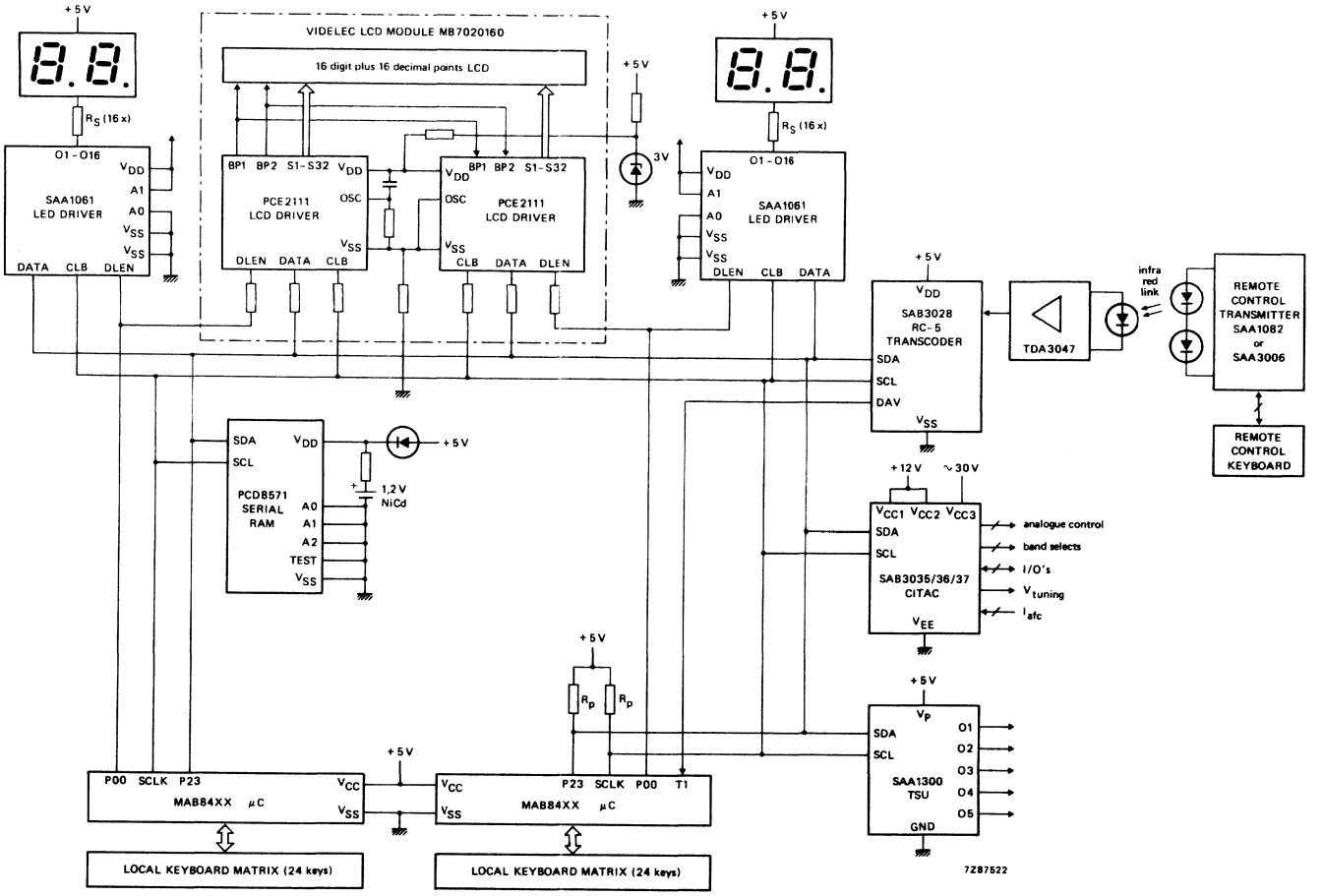


Fig. 3.48 Block diagram of an MAB84XX multi-master/multi-slave configuration

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.

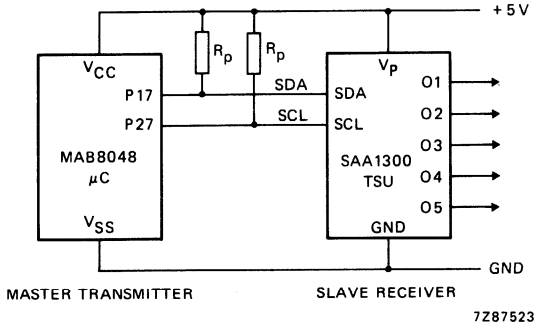


Fig. 4.1 Block diagram of MAB8048 - SAA1300 configuration

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, pins of two different I/O ports (port 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/$\overline{W}$ 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.

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.

```
                         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
000248 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
```

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 4.1.1).

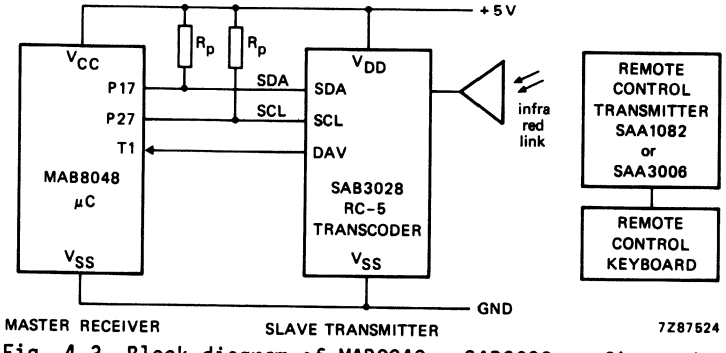


Fig. 4.3 Block diagram of MAB8048 - SAB3028 configuration

INITIALIZATION:

```
000000                      52 *        ORG     H'000'
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 MB8048 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/$\overline{W}$ 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

```
                     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 1228          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 - PCD8571)

This software is intended for members of the MAB8048 microcomputer family where there is only one master in an I²C-BUS system. In this system none of the slaves are allowed to stretch the clock low time. This software example only meets the I²C-BUS protocol functionally, electrically there are some differences. (See 4.1.1).

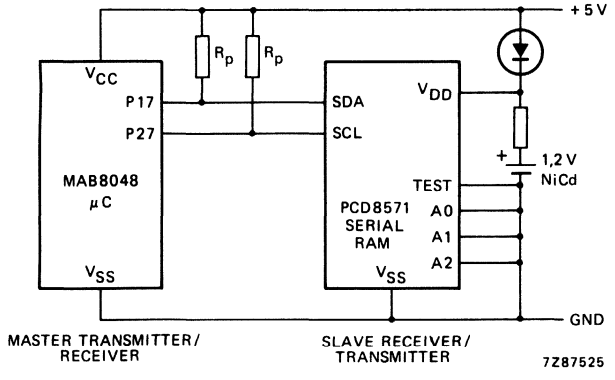


Fig. 4.5 Block diagram MAB8048 - PCD8571 configuration

```
INITIALIZATION:
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'
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     "
MAIN PROGRAM:
```

The main program calls the following two I²C-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 540D         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/$\overline{W}$ bit of the slave address, the master has to repeat the START condition, the slave address and the R/$\overline{W}$ 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/$\overline{W}$ 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 8A08       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 SUBROUTINE:

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 concept

CONTENTS – THE D²B CONCEPT

(continued on next page)

SUPPLEMENT 1

Data interpretation:

SUPPLEMENT 2

Address allocation:

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).

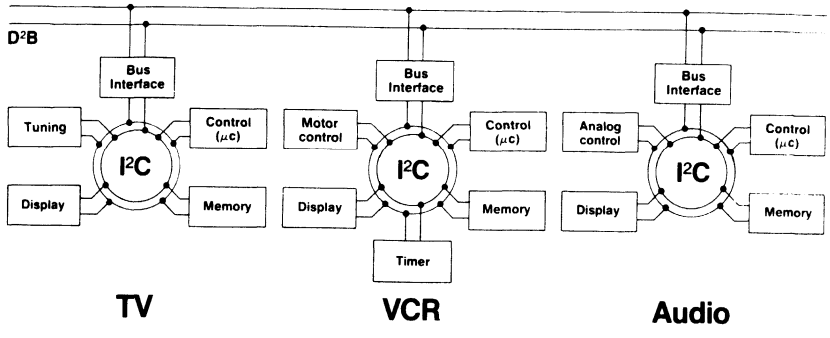


Fig. 1.1 The D^2B concept applied to an integrated Video/Audio system, each part of which employs the I^2C bus.

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 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 2457char/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 monopolize 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.

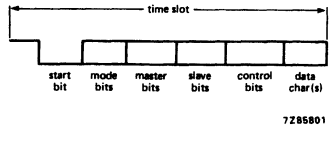


Fig. 2.1 Message format of the D^2B, showing the 6 bit-fields.

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.

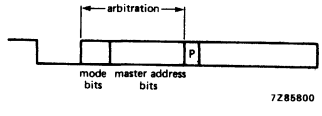


Fig. 4.1 Arbitration takes place during the mode and master fields.

- 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).

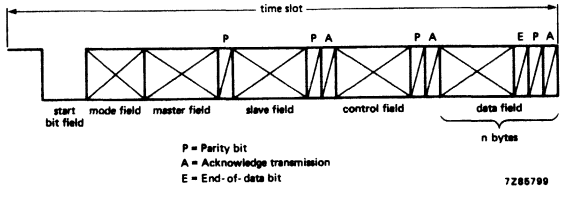


Fig. 5.1 A complete message on the D^2B.

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.

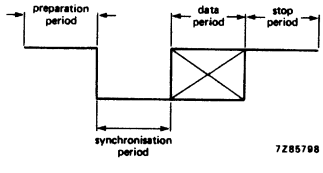


Fig. 6.1 The general bit format on the D^2B.

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.

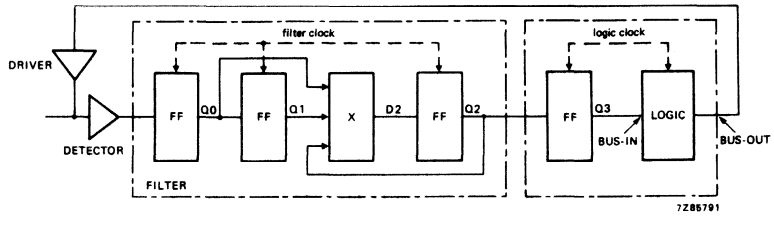


Fig. 6.2 Reference data-acquisition circuit.

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

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 (Ta) 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.

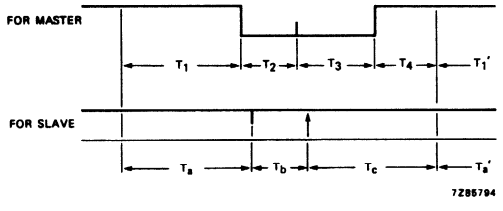


Fig. 6.3 The start bit format.

Master:

- T1:

  If a unit wishes to become master and the bus is HIGH during T1, it enters T2. 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 T1 and a LOW is detected, the unit becomes arbitrating-master and enters Tb (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 Tb and Tc to detect the long LOW period on the bus which indicates the generation of the start bit.

- T2:

  When a master enters T2, 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 T3. If no LOW is found, a timing error has occurred.

- T3:

  The bus should be kept LOW during T3, the master then enters T4 and generates a HIGH on the bus. If a HIGH is detected during T3, a timing error has occurred - since T3 is the interval which indicates the unique long LOW period of the start bit.

- T4:

  When the HIGH is detected during T4, the master enters T1'. If the bus is still LOW at the end of T4, a timing error has occurred.

Slave:

- Ta:

  This state is called the monitoring state. If a LOW is detected during Ta, the unit enters Tb as a possible slave.

- Tb:

  The bus should be LOW during Tb; the unit then enters Tc. If a HIGH is detected, the bit is not a start bit and the unit re-enters Ta.

- Tc:

  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 Tc, a HIGH is detected and the unit now wishes to become the master, it enters T1' and follows the master timing.

  If a unit does not want to be a master, it enters Ta' as a possible slave and follows slave timing

  If the bus is still LOW at the end of Tc, 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 (Ta)

    CONDITION      :master-request = 1 AND bus-in = 1

       ACTION           :bus-out = 1
                         clcnt   = t1
                         state   = STATE (T1)

    CONDITION      : bus-in = 0

        ACTION         : bus-out = 1
                         clcnt   = tb
                         state   = STATE (Tb)

    CONDITION      : master-request = 0 AND bus-in = 1

        ACTION         : bus-out = 1
                         state   = STATE (Ta)
```

For the master:

```
STATE (T1)

    CONDITION      : bus-in = 0

        ACTION         : bus-out = 1
                         clcnt   = tb
                         state   = STATE (Tb)

    CONDITION      : bus-in = 1 AND clcnt > 1

        ACTION         : bus-out = 1
                         clcnt   = clcnt - 1
                         state   = STATE (T1)

    CONDITION      : bus-in = 1 AND clcnt = 1

        ACTION         : bus-out = 0
                         clcnt   = t2
                         state   = STATE (T2)
```

```
STATE (T2)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = 0
                        clcnt   = t3
                        state   = STATE (T3)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 0
                        clcnt   = clcnt - 1
                        state   = STATE (T2)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error

STATE (T3)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 0
                        clcnt   = clcnt - 1
                        state   = STATE (T3)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out = 1
                        clcnt   = t4
                        state   = STATE (T4)

    CONDITION     : bus-in = 1

        ACTION        : bus-out = 1
                        timing error
```

```
STATE (T4)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (T4)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error

    CONDITION     : bus-in = 1

        ACTION        : bus-out = 1
                        clcnt   = t1'
                        state   = STATE (T1')
```

For the slave:

```
STATE (Tb)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tb)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out = 1
                        clcnt   = tc
                        state   = STATE (Tc)

    CONDITION     : master-request = 0 AND bus-in = 1

        ACTION        : bus-out = 1
                        state   = STATE (Ta)

    CONDITION     : master-request = 1 AND bus-in = 1

        ACTION        : bus-out = 1
                        clcnt   = t1
                        state   = STATE (T1)
```

```
STATE (Tc)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tc)

    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   = ta'
                        state   = STATE (Ta')

    CONDITION     : master-request = 1 AND bus-in = 1

        ACTION        : bus-out = 1
                        clcnt   = t1'
                        state   = STATE (T1')
```

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.

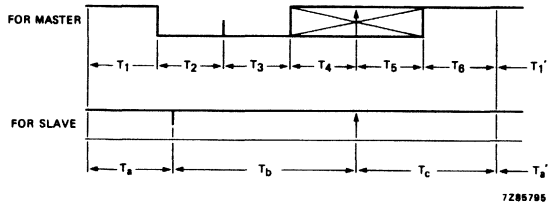


Fig. 6.4 Arbitration bit format.

- T1:

  If the bus remains HIGH during T1, the unit enters T2. If a LOW is detected, a master enters T3, keeping its output HIGH. This indicates that another master has generated the synchronization pulse for this bit.

- T2:

  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 T3 keeping its output LOW. If the LOW is not found, a timing error has occurred.

- T3:

  At the end of T3, the master enters T4.

- T4:

  At the beginning of T4, the arbitrating-master transmits the relevant bit for arbitrating with other units. At the end of T4, the unit reads the bus and enters T5.

  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.

- T5:

  If the bus is LOW at the end of T5, the unit enters T6. If the bus is HIGH, the next state is entered (T1'). A master which has lost the arbitration enters Ta' instead of T1'.

- T6:

  The master generates a HIGH and when this is detected during T6, the arbitrating-master enters the next state (T1'). A unit which has lost the arbitration enters Ta', as a possible slave.

  If the bus remains LOW, a timing error has occurred.

Slave:

- Ta:

  During Ta, a LOW should be detected on the bus. When it is, the unit enters Tb. If no LOW is detected, a timing error has occurred.

- Tb:

  At the end of Tb, the slave samples the bus. When the sample is HIGH, the next state is entered (Ta'). If not, the slave enters Tc.

- Tc:

  When a HIGH is detected during Tc, the next state is entered (Ta'). If the bus remains LOW, a timing error has occurred.

The state sequence for the arbitration bit is :

For the master:

```
STATE (T1)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = 1
                        clcnt   = t3
                        state   = STATE (T3)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (T1)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 0
                        clcnt   = t2
                        state   = STATE (T2)

STATE (T2)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = 0
                        clcnt   = t3
                        state   = STATE (T3)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 0
                        clcnt   = clcnt - 1
                        state   = STATE (T2)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error
```

STATE (T3)

CONDITION : bus-out = 0 AND clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T3)

CONDITION : bus-out = 1 and clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T3)

CONDITION : clcnt = 1

ACTION : bus-out = relevant bit
clcnt = t4
state = STATE (T4)

STATE (T4)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T4)

CONDITION : bus-in = 0 AND clcnt = 1 AND relevant bit = 0

ACTION : bus-out = relevant bit
clcnt = t5
bit value = 0
arbitration-not-lost
state = STATE (T5)

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 = t5
bit value = 0
arbitration-lost
state = STATE (T5)

CONDITION : bus-in = 1 AND clcnt = 1 AND relevant bit = 1

ACTION : bus-out = relevant bit
clcnt = t5
bit value = 1
arbitration-not-lost
state = STATE (T5)

STATE (T5)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T5)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = 1
clcnt = t6
state = STATE (T6)

CONDITION : bus-in = 1 AND clcnt = 1 AND arbitration-not-lost

ACTION : bus-out = 1
clcnt = t1'
state = STATE (T1')

CONDITION : bus-in = 1 AND clcnt = 1 AND arbitration-lost

ACTION : bus-out = 1
clcnt = ta'
state = STATE (Ta')

STATE (T6)

CONDITION : bus-in = 0 AND clcnt > 1

ACTION : bus-out = 1
clcnt = clcnt - 1
state = STATE (T6)

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 = t1'
state = STATE (T1')

CONDITION : bus-in = 1 AND arbitration-lost

ACTION : bus-out = 1
clcnt = ta'
state = STATE (Ta')

For the slave:

```
STATE (Ta)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = 1
                        clcnt   = tb
                        state   = STATE (Tb)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Ta)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error

STATE (Tb)

    CONDITION     : clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tb)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out    = 1
                        clcnt      = tc
                        bit value  = 0
                        state      = STATE (Tc)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out    = 1
                        clcnt      = ta'
                        bit value  = 1
                        state      = STATE (Ta')
```

```
STATE (Tc)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tc)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error
    CONDITION     : bus-in = 1

        ACTION        : bus-out = 1
                        clcnt   = ta'
                        state   = STATE (Ta')
```

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.

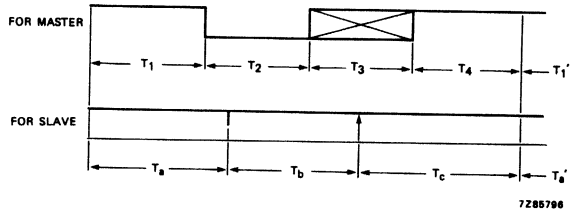


Fig. 6.5 Master to slave bit format.

Master:

- T1:

  The bus should remain HIGH during T1. If it does, the master enters T2. If a LOW is detected, a timing error has occurred.

- T2:

  The master generates the synchronization pulse by pulling the bus LOW. If the bus remains HIGH during T2, a timing error has occurred. If not, the master enters T3.

- T3:

  During T3, the master transmits the relevant data on the bus. It then enters T4.

- T4:

  During T4, 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 (T2).

Slave:

- Ta:

  A LOW should be detected during Ta. When it is, the slave enters Tb. If the bus remains HIGH, a timing error has occurred.

- Tb:

  At the end of Tb, the unit samples the bus to read the bit. If the bus is HIGH, the next state is entered (Ta'). If it is LOW, the unit enters Tc.

- Tc:

  As soon as a HIGH is detected, the next state is entered (Ta'). If the bus is still LOW at the end of Tc, a timing error has occurred.

The state sequence for master to slave bits is :

For the master:

STATE (T1)

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 (T1)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 0
clcnt = t2
state = STATE (T2)

STATE (T2)

CONDITION : clcnt > 1

ACTION : bus-out = 0
clcnt = clcnt - 1
state = STATE (T2)

CONDITION : bus-in = 0 AND clcnt = 1

ACTION : bus-out = relevant bit
clcnt = t3
state = STATE (T3)

CONDITION : bus-in = 1 AND clcnt = 1

ACTION : bus-out = 1
timing error

STATE (T3)

CONDITION : clcnt > 1

ACTION : bus-out = relevant bit
clcnt = clcnt - 1
state = STATE (T3)

CONDITION : clcnt = 1

ACTION : bus-out = 1
clcnt = t4
state = STATE (T4)

```
STATE (T4)

    CONDITION     : clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (T4)

    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 (Ta of the start bit)

    CONDITION     : bus-in = 1 AND clcnt = 1 AND not-last-bit-in-the-
                    message

        ACTION        : bus-out = 0
                        clcnt   = t2'
                        state   = STATE (T2')
```

For the slave:

```
STATE (Ta)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = 1
                        clcnt   = tb
                        state   = STATE (Tb)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Ta)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error
```

```
STATE (Tb)

    CONDITION     : clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tb)

    CONDITION     : bus-in = 0 AND clcnt = 1

        ACTION        : bus-out   = 1
                        clcnt     = tc
                        bit value = 0
                        state     = STATE (Tc)

    CONDITION     : bus-in = 1 AND clcnt = 1 AND last-bit-in-message

        ACTION        : bus-out   = 1
                        bit value = 1
                        state     = STATE (Ta of the start bit)

    CONDITION     : bus-in = 1 AND clcnt = 1 AND not-the-last-bit-in-
                    the-message

        ACTION        : bus-out   = 1
                        clcnt     = ta'
                        bit value = 1
                        state     = STATE (Ta')

STATE (Tc)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tc)

    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 (Ta of the start bit)

    CONDITION     : bus-in = 1 AND not-last-bit-in-the-message.

        ACTION        : bus-out = 1
                        clcnt   = ta'
                        state   = (Ta')
```

6.1.4 Slave to master bit

The slave to master bits form the "slave transmitter to master receiver" data transfer and acknowledge.

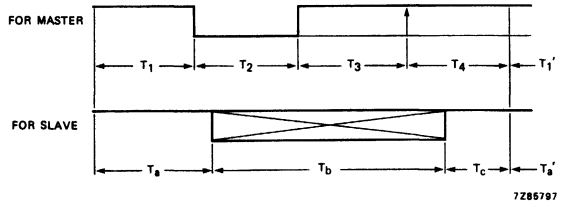


Fig. 6.6 Slave to master bit format.

Master:

- T1:

  Does not apply (T1 is always zero).

- T2:

  The master transmits the synchronization pulse by pulling the bus LOW during T2. At the end of this period, when the LOW is detected on the bus, the master enters T3. If the bus remains high during T2, a timing error has occurred.

- T3:

  During T3, the master generates a HIGH on the bus. After t3, it samples the bus to read the bit transmitted by the slave and enters T4.

- T4:

  The bus should be HIGH at the end of T4. The master then enters the next state (T2). If the bus is still LOW, a timing error has occurred.

Slave:

- Ta:

  When the LOW generated by the master is detected, the slave enters Tb. If the bus is still HIGH at the end of Ta, a timing error has occurred.

- Tb:

  During Tb, the slave transmits the relevant bit. If at the end of Tb the bus is HIGH, the slave enters (Ta'). If the bus is LOW, the slave enters Tc.

- Tc:

  During Tc, a HIGH should be detected on the bus. When it is, the next state is entered (Ta'). If the bus remains LOW, a timing error has occurred.

The state sequence for slave to master bits is:

For the master:

STATE (T1)

(Does not apply)

STATE (T2)

```
CONDITION   : clcnt > 1

    ACTION      : bus-out = 0
                  clcnt   = clcnt - 1
                  state   = STATE (T2)

CONDITION   : bus-in = 0 AND clcnt =1

    ACTION      : bus-out = 1
                  clcnt   = t3
                  state   = STATE (T3)

CONDITION   : bus-in = 1 AND clcnt = 1

    ACTION      : bus-out = 1
                  timing error
```

STATE (T3)

```
CONDITION   : clcnt > 1

    ACTION      : bus-out = 1
                  clcnt   = clcnt - 1
                  state   = STATE (T3)

CONDITION   : bus-in = 0 AND clcnt = 1

    ACTION      : bus-out   = 1
                  clcnt     = t4
                  bit value = 0
                  state     = STATE (T4)

CONDITION   : bus-in = 1 AND clcnt = 1

    ACTION      : bus-out   = 1
                  clcnt     = t4
                  bit value = 1
                  state     = STATE (T4)
```

```
STATE (T4)

    CONDITION     : clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (T4)

    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 (Ta of the start bit)

    CONDITION     : bus-in = 1, clcnt = 1 AND not-last-bit-in-the-
                    message

        ACTION        : bus-out = 0
                        clcnt   = t2'
                        state   = STATE (T2')
```

For the slave:

```
STATE (Ta)

    CONDITION     : bus-in = 0

        ACTION        : bus-out = relevant bit
                        clcnt   = tb
                        state   = STATE (Tb)

    CONDITION     : bus-in = 1 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Ta)

    CONDITION     : bus-in = 1 AND clcnt = 1

        ACTION        : bus-out = 1
                        timing error
```

```
STATE (Tb)

    CONDITION     : clcnt > 1

        ACTION        : bus-out = relevant bit
                        clcnt   = clcnt - 1
                        state   = STATE (Tb)

    CONDITION     ; bus-in = 0 AND clcnt = 1

        ACTION        : bus-out = 1
                        clcnt   = tc
                        state   = STATE (Tc)

    CONDITION     : bus-in = 1 AND clcnt = 1 AND last-bit-in-the-
                                                 message

        ACTION        : bus-out = 1
                        state   = STATE (Ta of the start bit)

    CONDITION     : bus-in = 1 AND clcnt = 1 AND not-last-bit-in-the-
                                                 message

        ACTION        : bus-out = 1
                        clcnt   = ta'
                        state   = STATE (Ta')

STATE (Tc)

    CONDITION     : bus-in = 0 AND clcnt > 1

        ACTION        : bus-out = 1
                        clcnt   = clcnt - 1
                        state   = STATE (Tc)

    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 (Ta of the start bit)

    CONDITION     : bus-in = 1 AND not-last-bit-in-the-message

        ACTION        : bus-out = 1
                        clcnt   = ta'
                        state   = STATE (Ta')
```

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

* 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.

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.

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 supplement 1) 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).

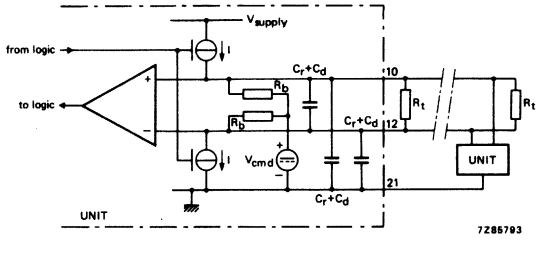


Fig. 8.1 Reference circuit for connecting units to the D^2B via a differentially driven cable.

8.1 Logical and electrical state relationship

The relationship between the logical states and the electrical levels on the data lines are given in 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:

$$2{,}75 \text{ mA} \leqslant I \leqslant 5 \text{ mA at ON}$$
$$I < 1 \text{ µA at OFF}$$

- 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:

$$C_d < 25 \text{ pF between points 10 and 12}$$
$$C_d < 25 \text{ pF between points 10 and 21}$$
$$C_d < 25 \text{ pF between points 12 and 21}$$

- The transition time values (T) are given in Fig. 8.2.

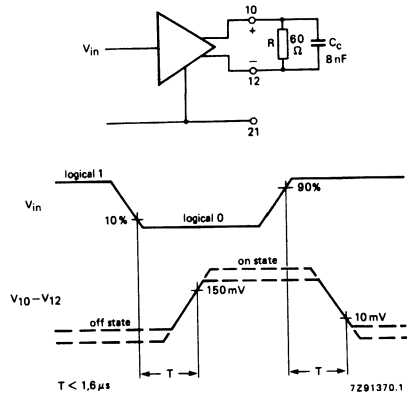


Fig. 8.2 The transition time measurement

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:

$$1\ V < Vcm_r < 3{,}75\ V \text{ for ON}$$
$$0\ V < Vcm_r < 5\ V \text{ for OFF}$$

- The input voltage ($V_{10} - V_{12}$) is:

$$V_{10} - V_{12} \geqslant 120\ mV \text{ for ON}$$
$$V_{10} - V_{12} \leqslant 20\ mV \text{ for OFF}$$

- 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.

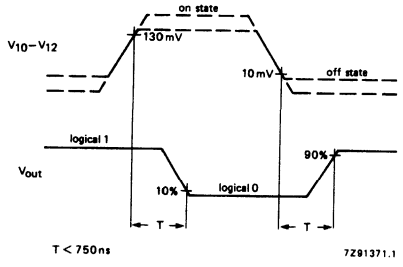


Fig. 8.3 Receiver delay measurement.

8.4 Cable characteristics

- Wire resistance $\leqslant$ 0,1 Ω/m
- Characteristic impedance (Z_0) = 120Ω ± 20%

8.5 Configuration

- Bus length = 150 m (max)
- Resistive cable termination = 120Ω ± 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 us at any terminal connected to the bus.

The propagation delay due to the bus wires has to be less than 0,9 us.

The delay of the driver circuitry must be less than 0,1 us.

The delay of the detector circuitry must be less than 0,75 us.

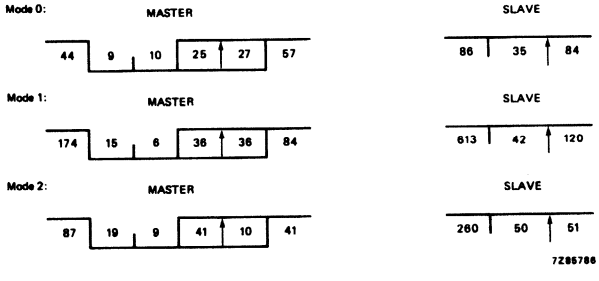
9.1 Bit timing

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.

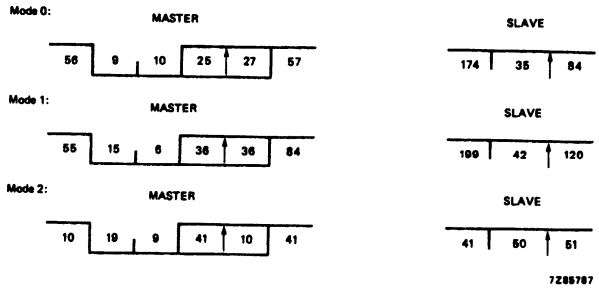
9.1.1 Start bit timing

The start bit always takes place in mode 0.

9.1.2 Mode bit timing

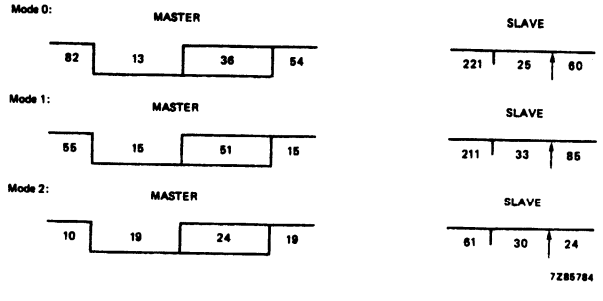


9.1.3 Master address bit timing

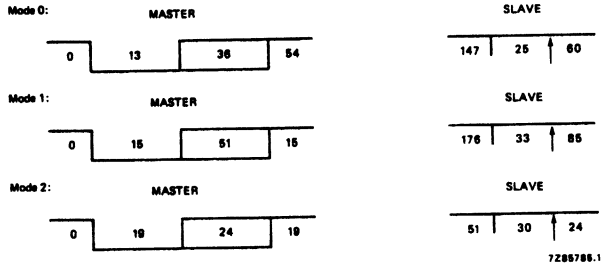


9.1.4 Master to slave bit timing

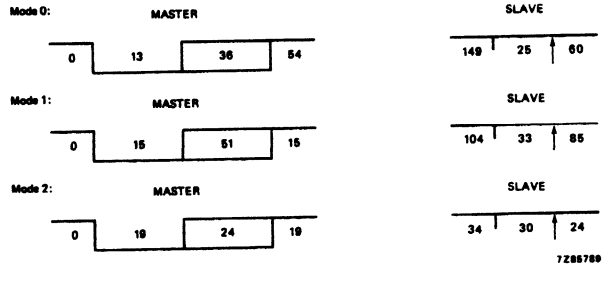
- Parity bit on master address



- 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.

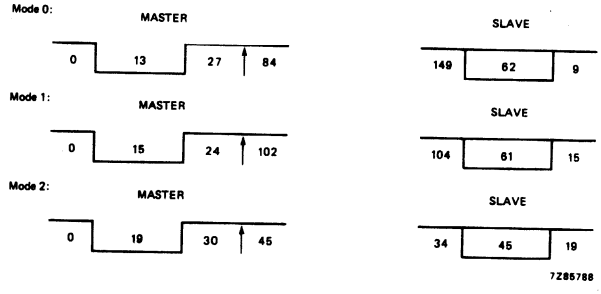


- All other master to slave bits.

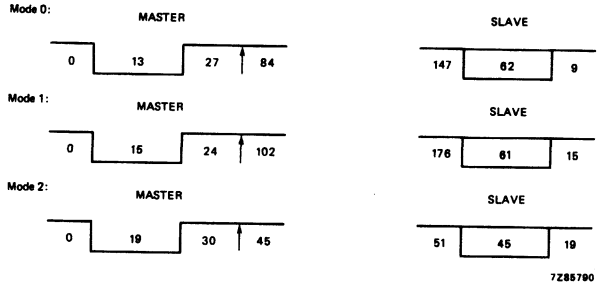


9.1.5 Slave to master bit timing

- 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

SUPPLEMENT 1

Data interpretation:

1.0 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.

2.0 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 "

3.0 MEMORY ADDRESS

Each unit for audio-video applications has to be implemented with a memory that can be accessed via the control field specification "write memory address".

When a unit receives a message and the control field indicates "write memory address", the received data bytes are processed in the following way.

- the data buffer empty bit in the slave status byte is reset to zero to avoid reading irrelevant data
- the most-significant (MS) bit (bit no. 7) of each byte, starting with the first data byte received, is checked.
  - if the value is zero, the unit will act in the following way:

    the data bits b6 - b0 are interpreted as the 7 MS bits of the memory address and are stored in the MS part of the memory address buffer (bit 6 is stored in the MS bit of the memory address);

- if the value is one, the unit will act in the following way:

  the data bits b6 - b0 are interpreted as the 7 LS bits of the memory address and are stored in the LS part of the memory address buffer;

  it will put the data (8 bits) stored in the memory on the address specified by the memory address buffer, into the data buffer and the data buffer empty bit in the slave status byte is set to '1' (the data buffer is the buffer that can be accessed by "read data" defined by the control bits);

  if a memory address is accessed that is not specified, the value "all zeros" will be put into the data buffer. The value "all zeros" will also be put into the memory address buffer.

4.0 DATA

When the control bits specify "read data", the data available in the slave's data buffer is received by the master.

The interpretation of the received data depends upon the specification of the unit and/or the contents of the memory (see this supplement, section 3.0).

When the control bits specify "write data", the interpretation of the received data depends on specification of this unit.

5.0 COMMANDS

When the control bits specify "write command", the following codes indicate the type of function the slave is required to perform. The "locking" or "unlocking" of a slave to a master is also specified in the control bits.

5.1 Command codes applicable to the address space 256-511

5.1.1 Command codes applicable to any type of equipment

Code	General commands	Audio commands	Video commands
0-9	Numerical values 0-9		
12	Unit standby		
13		Mute/demute (toggle)	
14	All analogue to nominal values		
15	Display		
16		Sound+	
17		Sound-	
18			Brightness+
19			Brightness-
20			Colour saturation+
21			Colour saturation-
22		Bass+	
23		Bass-	
24		Treble+	
25		Treble-	
26		Balance right+	
27		Balance left+	
49	Correction, clear		
53	Play		
54	Stop		
55	Record		
56	Connect (audio + video)		
57	Disconnect (audio + video)		
61	System standby		
64			Contrast+
65			Contrast-
66		Medium+	
67		Medium-	
68			Hue+
69			Hue-
70-79	not yet allocated (reserved for continuous functions)		
127	Unit mains off		
128	Block no. 0		
"	" "		
159	Block no. 31		
160	Unit no. 0 within a block		
"	" " " "		
"	" " " "		
167	Unit no. 7 within a block		

Code	General commands	Audio commands	Video commands
192	Key release		
193	Command extension		
194			
195			
196		Mute	
197		Demute	
198	Set protection		
199	Reset protection		
200		Stereo sound	
201		Mono sound	
202		Two independent sound channels	
203		Select channel A	
204		Select channel B	
205	Set external (audio + video)		
206	Set external only audio		
207	Set external only video		
208	Set internal (audio + video)		
209	Set internal only audio		
210	Set internal only video		

5.1.2 Command codes applicable to specified types of equipment

5.1.2.1 Recorders and players

Code	Command
30	Search forward to next marker
31	Search reverse to next marker
34	Play slow reverse
37	Step picture reverse
38	Slow motion speed+
39	Slow motion speed-
40	Play slow forward
41	Step picture forward
42	Play fast forward
44	Picture search reverse
45	Eject
46	Picture search forward
47	Play normal reverse
48	Pause
50	Tape rewind
52	Tape wind
59	Arm up
60	Arm down

5.1.2.2 Tuners (Video & Audio)

Code	Command
30	Search forward
31	Search reverse

5.1.2.3 Source selector

Code	Command
211	Connect audio
212	Connect video
213	Disconnect audio
214	Disconnect video

5.1.3 Reserved codes for future standardization

All command codes >215 and $\leqslant 255$ are reserved for future standardization

5.1.4 Command procedures

Commands can be transmitted to a unit via the D^2B in two ways:

1. Only the code belonging to the command is transmitted.
2. A sequence of two commands is transmitted as follows:

   - the code representing the command
   - the command "key-release" (code 192).

   The following command codes should always be followed by a key-release command:

   - the codes applicable to any type of equipment:

     codes 16 - 27
     codes 64 - 79

   - and the video recorder, video player command codes:

     code 38
     code 39.

If a certain command code has to be interpreted with another non-standardized meaning, such a command has to be preceded by the command code 193 (command extension).

For a source selector, it is necessary to define a procedure for making/ breaking a connection between two units. This procedure is defined by the following sequence of commands :

- command indicating a block number (128-159)
- command indicating a unit number within a block (160-167)
- connect/disconnect command (56, 211 or 212)
- command indicating a block number (128-159)
- command indicating a unit number within a block (160-167).

A block is a part of the address space set aside for a particular type of equipment (see Supplement 2).

The sequence of the first two commands uniquely defines a unit designated as the "source unit".

The sequence of the last two commands uniquely defines a unit designated as the "destination unit".

SUPPLEMENT 2

Address allocation:

1.0 ADDRESS SPACE DEFINITION

Different units are defined by the 12 address number bits providing a sequence of codes numbered 0-4095 with the following interpretation :

0 - 255	not defined
256 - 511	reserved for peripheral equipment and audio-video source selectors
512 - 4095	not defined.

2.0 ALLOCATION OF ADDRESS CODES

The address space 256-511 is divided into 32 blocks, each block indicating the type of equipment. Each block can define 8 units.

If only one unit in a certain block is present in the system, it will have the lowest address within the block. If a second unit in the same block is present in the system, it will have the address "lowest + 1" within the block, and so on.

Bit 11 is the MSB of the address and bit 0, the LSB. The following allocation is defined :

- The most significant 4 address bits have the fixed value:

bit 11 = 0
bit 10 = 0
bit 9 = 0
bit 8 = 1

- The block numbers indicated by bits 7, 6, 5, 4 and 3 are allocated to the following types of equipment:

bit no. 7	6	5	4	3	allocated type of equipment
0	0	0	0	0	video monitor
0	0	0	0	1	audio pre-amp
0	0	0	1	0	video source selector
0	0	0	1	1	audio source selector
0	0	1	0	0	video recorder
0	0	1	0	1	video tuner
0	0	1	1	0	video player
0	0	1	1	1	video camera
0	1	0	0	0	teletext decoder
0	1	0	0	1	videotex decoder
0	1	0	1	0	reserved for future standardization
0	1	0	1	1	reserved for future standardization
0	1	1	0	0	reserved for future standardization
0	1	1	0	1	reserved for future standardization
0	1	1	1	0	reserved for future standardization
0	1	1	1	1	free for use selector
1	0	0	0	0	audio tuner
1	0	0	0	1	audio recorder
1	0	0	1	0	compact disc
1	0	0	1	1	phono
1	0	1	0	0	reserved for future standardization
1	0	1	0	1	reserved for future standardization
1	0	1	1	0	reserved for future standardization
"		"			" " " "
1	1	1	1	0	reserved for future standardization
1	1	1	1	1	free for use

If a unit is defined which consists of several type of equipment then this unit will have a block number equal to the lowest block number of the equipment implemented within it. For example, a television set will have the block number 00000 because it contains a video monitor.

12. Microcontroller development support

CONTENTS – MICROCONTROLLER DEVELOPMENT SUPPORT

1.0 INTRODUCTION

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 requre 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.

Program development, emulation and debugging facilities are the major components of a Microcontroller Development System, or MDS.
We offer a number of alternatives in this field :

- Microcomputer development System (PMDS 1) - PM4421
- Microcomputer development System (PMDS 2) - PM4422
- Engieering development Systems (PEDS) - PM48XX
- Stand alone Development System (SDS)
- Microcontroller Instructor - PM4300

The PM4421/2 (PMDS 1/2) 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 PMDS configured for different types of target CPU. In both systems facilities such as Text Editing, Assembly, Linking, Emulation and Debugging aid the user from initial concept to final testing.

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 PM4300 is a low cost solution for microcontroller emulation and design work. It allows dynamic testing of a microcontroller system using emulation to give access to the internal operation of the target microcontroller. Memory contents or I/O devices can also be actively modified through PM4300 commands. An RS-232 interface also permits up and downloading of programs in conjunction with a host computer system.

2.0 MICROCONTROLLER DEVELOPMENT SYSTEMS (PMDS 1/2) - PM4421/2

2.1 Microcontroller development system PMDS 1

A universal microcontroller system, such as the PM4421, is a non-dedicated tool capable of supporting many different microcontrollers and microprocessors. It has the capability of handling 8- and 16-bit microcontrollers as well as multi-processor configurations. Modular programs can be written and debugged individually before linking. Debug facilities include real time emulation at the full rated speed of the target microcontroller, in all hardware environments and from an early stage of product development.

The key to the versatility of PMDS (1/2) is a 'black-box' adaptor that fits between it and the target microcontroller. Only this adaptor needs to be exchanged when changing over to a new microcontroller, resulting in large time and cost savings. Additional savings are obtained through the microcontroller independent hardware and software of PMDS (1/2) no extra familiarisation time being required when using different target microcontrollers.

For type numbers of the particular MAB (Microcomputer adaptor box) required and hardware/software options available for your application, see section 8.0 - a Technical Data Sheet is also available from our Industrial and Electro-acoustic Systems division.

2.2 Microcontroller development system PMDS 2

Upon entering the microcontroller market back in 1979, We immediatly introduced a single-user universal development system, the PM4421 -PMDS 1. The need soon became apparent for a multi-user development system, and this was introduced as the PM4422 -PMDS 2 in 1982.

PMDS 2 is a powerful development tool. The system possesses many improvements on the PMDS 1. Such improvements include software manipulation through a versatile UNIX based operating system, a time sharing multi-user operation and support for a variety of high level languages.

PMDS 2 supports development for a wide range of both 8 and 16 bit microcontrollers and microprocessors.

2.3 System architecture PMDS 1 - PM4421

PMDS 1 features a multibus-multiprocessor architecture, in which system and emulation functions are totally separate (Fig.1). Various benefits are obtained from such a set-up:

- Emulation is carried out in real-time as no system management constraints exist on the emulation facilities.
- There is no chance of master system software being damaged by program faults.

The 'nerve-centre' of the PMDS is the PM4401 computer. In its PMDS configuration, the PM4401 consists of a 'master system' plus a Universal Debug Unit (UDU). The master system includes the CPU, a P851 LSI minicomputer with 16-bit word-length, 14 program registers, 160 micro-programmed instructions and 64 K byte address range. The System Memory is a 64 K byte fast dynamic RAM, access time 450 ns for 16 bits. Mass storage is provided by two built-in mini-floppy disk drives. The drives provide double-sided double density recording, resulting in a capacity of 320 K bytes (formatted) per diskette. A video display and detachable keyboard are also part of the main cabinet. Figure 1. illustrates PMDS 1 architecture.

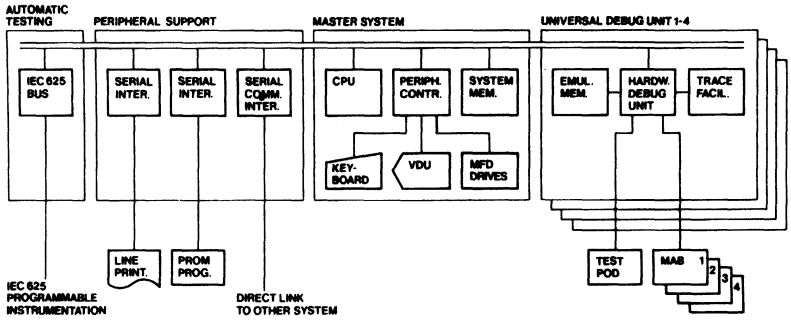


Fig. 1. PMDS 1 architecture

2.4 System software PMDS 1

Software for PMDS 1 is provided pre-recorded on diskette. It consists of a system control program called Monitor plus a set of software programs known as PMDS Processors. Monitor must be present in the system all the time and works in conjunction with any of the Processors (Fig. 2).

Monitor - has overall control of the system. It controls transfer of data between the PM4401 and input/output devices and the loading of software into system memory. All Monitor activities happen automatically but some tasks can be dealt with 'on request' - in response to keyboard activated commands. Most control commands are handled directly via the Control Command Interpreter or CCI.

PMDS 1 Processors include the CCI, Text Editor, Assembler, Linker, PROM processor, Debugger and a general purpose communication package. The communications package allows software communication with mainframes, mini-computers or other external sources. Figure 2. overleaf, illustrates PMDS 1 software structure.

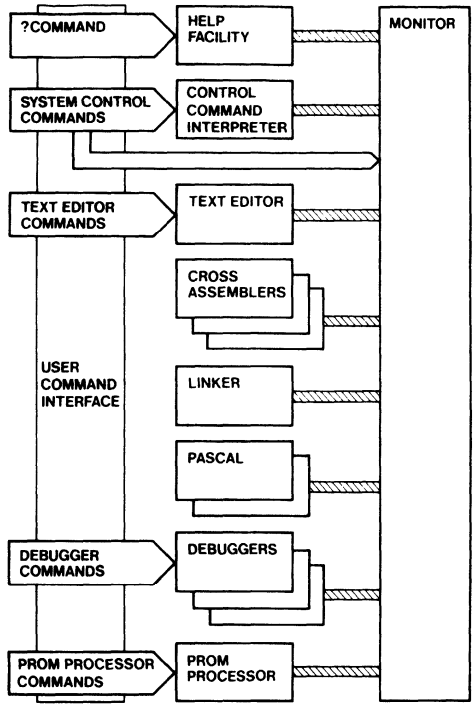


Fig. 2 PMDS 1 - Software structure

Control Command Interpreter - includes a set of commands to activate other processors. It also carries out diskette-file management and performs other system control tasks. As with all PMDS commands, CCI commands are interactive - entry from the keyboard resulting in feedback on the screen. Command sequences can also be stored as 'command procedures' on diskette, if required, and activated with a single CCI command.

Text Editor - is used for keyboard entry of source programs. The data entered is stored on diskette for future assembly. Again the interactive command language is used for all editing operations.

Cross-Assemblers - are designed for use with a single microcontroller or for a 'family' of similar microcontrollers. They are designed so that the source language is the same as the manufacturers, but all label and comment conventions and assembler directives are standard. Code can be relocatable or absolute. The Cross Assembler Processor includes all facilities for expression evaluation, macro processing, and conditional assembly.

Linker - combines translated program modules (object modules) into a complete program. In doing this, it resolves intermodular cross-references and is able to map the program into the address space of the target microcontroller. The resulting complete program, 'the load module' is stored on diskette and can be loaded straight into a PROM or the UDU emulation memory for an emulation run on the target microcontroller.

PROM Processor - drives the external hardware PROM Programming Unit. When using this processor, keyboard commands can initiate machine code Program and PROM operations. As well as 'burning-in' PROMS, command functions include: copying, verifying and blank-checking PROMS; inserting PROM checksums; and examining and patching machine code programs.

Debugger - controls all facilities of the UDU in order to carry out emulation and debugging. Different versions exist for the various types of microcontroller to be tested, but commands and facilities available are largely identical for each version. Program emulation can be carried out on various prototype and PMDS hardware configurations.

PMDS PASCAL compilers may be used to develop programs for various microprocessors supported by PMDS 1. All compilers are written to guarantee real portability of PASCAL programs within the PM4421. The language is implemented as described in the ISO report ISO/TC97/SC5 N565 with some important extensions.

The PASCAL compiler generates from the source code a standard PM4421 relocatable object code file. This may be linked with other program parts by the linker to produce the microprocessor load file. Further to this, the interpreter available on the PM4421 has extensive debug facilities and the same source language specification as the PASCAL compiler.

PMDS PASCAL provides a calling sequence which allows interfacing with ASSEMBLER language written routines, these are then incorporated into the loadfile for hardware dependent, real time or I/O operations.

2.5 System architecture PMDS 2

Like PMDS 1, PMDS 2 features a multibus-multiprocessor architecture, system and emulation functions are again totally seperate. 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 M byte addressing range.

Multi-user operation under a time sharing scheme is acheived by using up to 4 Tcom cards, each card housing 1 SP16C/10 slave CPU. In addition to the slave processor, the Tcom card also carries 128 K bytes 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.

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 K bytes 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 K bytes (formatted) per diskette with the winchester providing from 5 to 21 Mbyte of storage area. Figure 3. illustrates PMDS 2 -System architecture.

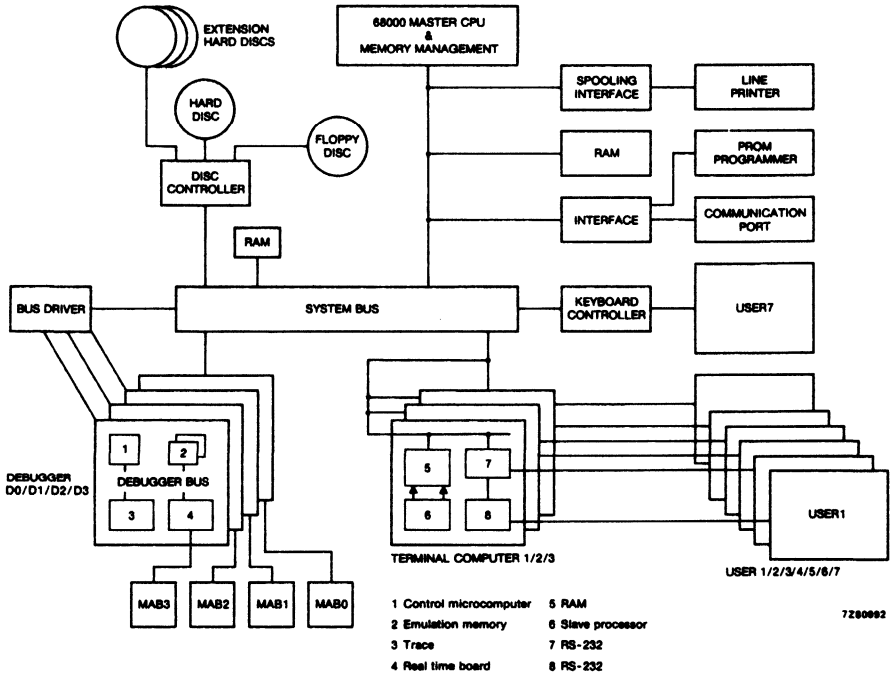


Fig. 3 PMDS 2 System architecture.

2.6 System software 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 4 shows the software structure of PMDS 2.

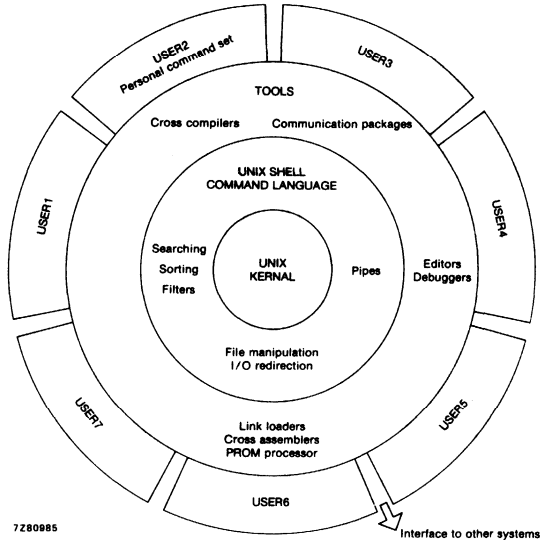


Fig. 4 PMDS 2 software structure

*UNIX is a product of Bell laboratories

PMDS 2 system processes are listed below (for description see PMDS 1):

1) The text editor

2) The cross-assemblers*

3) The linker

4) Debugging

5) The PROM processor

6) The PASCAL compiler and interpreter

*Some of the many advanced features common to PMDS 2 assemblers are:

- Modular support to allow structured programming
- Macro-expansion facilities, including nesting
- Object code optimisation
- Conditional code generation
- Memory type specification
- Cross-reference listing
- Extensive set of error messages

Note: A 16 bit compiler is also available for development in 'C' language.

2.7 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. Each major unit within the UDU will now be described in turn:-

The Control Microcontroller - interfaces debugging hardware to the PMDS system bus, allowing control and interrogation of the Universal Debug Unit by the user system via the Debug Unit controlling software.

The Real Time Control unit - supervises the real-time switching function of the MAB and synchronizes the flow of information during real-time emulation between emulation memory, trace memory, MAB and Test Pods.

Emulation Memory - is used to simulate all, or selected parts of, target memory during software debugging or system integration. Since system memory is functionally separated from the Emulation Memory, it cannot be corrupted by target system errors.

Trace Memory - is a 256 x 48 bit memory used to collect 'history' data during a real-time emulation run. It is provided together with an Event Counter and 24-bit register to count a wide variety of hardware and software events and to count down to trigger emulation breaks.

Microcontroller Adaptor Box - is available in different versions for different microcontrollers. It's main function is to act as a 3-way switch to connect the target microcontroller, which is normally plugged in the top, to either the PMDS or a prototype circuit at real-time speeds. The MAB also provides software-selectable clock frequencies for use in emulation and has a BNC connector for connection of an external clock generator and for external monitoring of trigger signals.

Test Pod - carries a set of 8 probes to monitor signals at selected points in the prototype circuit. The probes are referred to as 'user probes' to distinguish them from emulation probes. The signals detected, as well as being used to control emulation, may be captured as history data by the Trace Memory. The pod is connected via a ribbon cable to the rear of the PM4401 cabinet and from there to the Real Time Unit. A second Test Pod can also be connected via the MAB (for 8-bit emulators).

2.8 PMDS 1/2 debugging facilities

Debugging commands can examine and change the contents of memory, microcontroller registers or I/O ports.

Control groups available include:

- Mapping commands
- Parameter setting
- Emulation control
- System status
- File Manipulation

Emulation runs can be made in real-time with breakpoints at strategic locations, or step-by-step mode with whatever increment is required. Symbolic debugging makes it possible to use symbolic names to reference memory contents, map I/O or set breakpoints. Access can therefore be achieved without the need to remember absolute hexadecimal values.

A particular advantage of PMDS 2 and it's UNIX based operating system is the facility to create a new 'shell' during debug. This means, that without losing all the parameters set during a debug session, a modified or alternative version of the program can be edited, assembled and linked. It is then possible to return to the session, load the new program and continue with the emulation.

Before starting emulation on an incomplete prototype, mapping commands can be used to indicate which fuctions are to be replaced by the facilities of the Development System. Even if all hardware is not available, debugging can still be carried out.

Target system memory is divided into 256 segments of 256 bytes. The MAP command enables each segment to be assigned to prototype memory or the UDU emulation memory, or to give it guarded status. The type of memory (RAM or ROM) in each segment can be specified. If a guarded segment is addressed during an emulation run, a program halt is initiated.

Similarly for microprocessors with memory-mapped I/O, IMAP and OMAP allow 256 input/output ports to be assigned to either the prototype system, keyboard, printer, screen, file, or system memory located queue.

When I/O and memory mapping is complete, emulation parameters can be defined by four registers:

- two trigger registers - which halt emulation (forming a 'breakpoint') when the contents of the registers are matched by conditions on the bus.

- two trigger registers - which qualify the type of data to be stored in real-time trace memory during real-time emulation or the type of step performed during step mode emulation.

Trigger registers contain 48 match condition bits which can be 0, 1 or X (don't care).

Register breakpoints can occur for:

- address lines (16)
- data lines (8 or 16)
- control lines (4)
- extended control lines (4)
- standard test probe (8)
- extra test probe (8) ..... or any combination.

A match counter may be set to specify the number of times that the match point is to occur before generating a breakpoint. Trigger registers can also be armed to trigger only on the occurence of another match. This is useful when breakpoints are required within a nested subroutine.

An optional 24-bit event counter can be used to count,

- clock cycles
- machine cycles
- instruction cycles
- real-time
- interrupt acknowledges
- break condition 0
- break condition 1

The counting process is continuous during an emulation run or can be started/stopped at break condition 0 or 1.

In order to carry out an emulation run, the program is first loaded into memory, using the LOAD command. Emulation can then be carried out in either step or real-time mode. Stepping is usually carried out first. By depressing the STEP key, each program stage is shown, in turn, on the display.

The STEP command can control various modes. In addition to single stepping, the other modes permit stepping between jump instruction or subroutine calls and returns. Another valuable stepping mode enables stepping between match points of qualifier registers.

Stepping can continue under debugger control with no need for a command for each step. Specifying STEP FOREVER means that stepping will continue until a HALT command is entered, or the HALT button is pressed, otherwise the program runs until a stop condition occurs:

Stop conditions can occur for the following:

- matched breakpoint
- write access to ROM
- guarded memory access
- guarded input/output port access
- event counter times out
- multi-emulation break
- DMA cycle detected
- target microcontroller reset
- target microcontroller halt
- target system power failure

To run a program in real-time - the Run button is pressed on the keyboard. The emulation starts running at full speed from the program start address to the last halt address. Emulation start points can be anywhere in the program and program execution can be halted at any time by pressing HALT key. Unless the key is pressed, emulation will continue until a stop condition arises.

When carrying out emulation, a combination of both step and real-time modes are generally used.

During emulation run, events at selected program points can be investigated using the Trace facility. Trace memory can hold 255 data entries including:

- address
- data and control line signals
- user probe signals

Data can be collected on every machine cycle, instruction cycle, op-code fetch or clock pulse at the test pod 0 probes. Trace facilities are triggered by matching values set in the qualifier registers. There are three types of triggering modes: pre-, post-, and centre-trigger.

By selecting the appropriate trigger, trace memory can be filled in the following three ways:

1) Emulation halts on trigger and the 255 previous traces are stored (pre-trigger mode)

2) Trace memory is filled with 127 traces before, and 128 after triggering (centre-trigger mode)

3) Program continues running for another 255 traces after triggering (post-trigger mode).

When a breakpoint occurs the program will continue running until an instruction is completed. Examination of an emulation run would be useless without some means of altering program contents. System status commands allow the contents and status of the target system to be examined and modified. In particular:

* all internal registers can be read or modified
* memory locations can be displayed on the screen and modified
* specific values can be read from or written to I/O ports - useful when physical signals cannot be connected to a prototype.

The WFILE command allows downloading of a load module resident in memory onto a diskette file.

2.9 PMDS peripherals

PMDS is equipped with the following peripherals:

- a high-speed printer
- a universal PROM Programmer
- an (optional) external mini-floppy disk drive extension (contains 2 drives)
- optional back-up tape facilities

The universal PROM Programmer is a remote-control device operated by PMDS keyboard commands. By using interchangeable slot-in 'Program-PAKS' and socket adaptors the programmer can handle a wide range of PROM types with a capacity of up to 4K x 8 bits.

3.0 THE SDS-8051/80C51 STAND-ALONE DEVELOPMENT SYSTEM

Outline description

The SDS-80C51 is a stand-alone system for debugging and emulation of the 8051/80C51 family of 8-bit microcontrollers.

Commands for required operation are entered via an RS-232C serial interface port from a single CRT terminal. The command language is INTEL based so as to reduce re-learning requirements by INTEL users.

Features of SDS-51 include:

- Self-contained architecture
- 80C51 bond-out chip
- 2 RS-232C serial ports
- INTEL based command language
- Availability for other microcontroller families
- MDS and Personal computer compatibility.

3.1 System architecture

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 based on the Intel EMV-51 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 from which address and control data may be extracted when required.

The power supply is entirely self-contained within the device. Figure 5. shows a block diagram of the SDS-80C51.

No real time trace facility is included on the SDS-80C51 system.

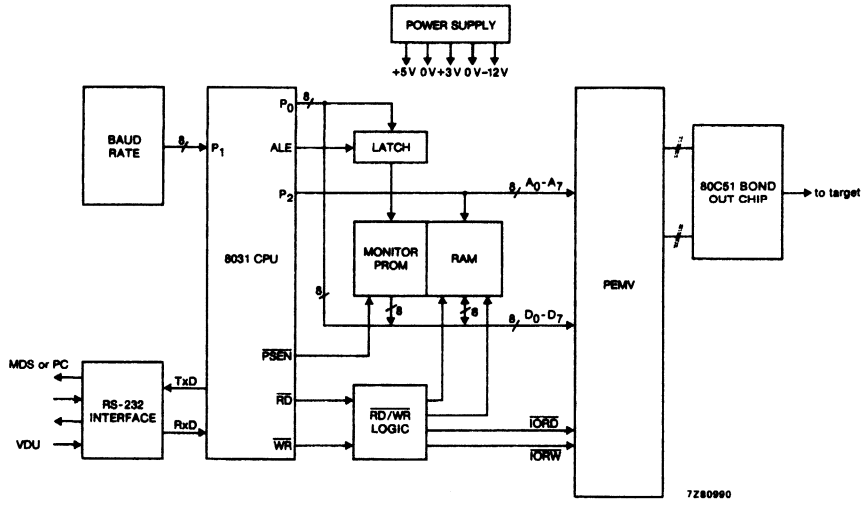


Fig. 5. Block diagram of SDS-80C51

3.2 SDS-80C51 system software

Monitor software in 12 K of EPROM enables the user to perform the following functions:

- To communicate with SDS-80C51 using a CRT terminal
- 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 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 8051/80C51's data memory and registers
- To upload or download object code programs from a microcomputer development system or a personal computer system.

3.3 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 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 instruction mnemonics.

Continuation mode allows the user to enter a list of labels 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.

4.0 PEDS - ENGINEERING DEVELOPMENT SYSTEM

PEDS was developed as an upgraded single-user version of PMDS 1. PMDS 1 has been completely re-engineered to present a multi-tasking, high through-put system. 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. Additional facilities include:

- A UNIX based operating system
- Softkeys for fast start-up and learning
- 256 K to 1 Mbyte RAM + up to 10,8 Mbyte mass storage
- Support for a wide range of MPU's and MCU's.

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 K byte memory accessible through a local bus. Using the system bus, the 68000 can access all memory which may be extended up to 1M 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 K byte 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 6. shows the PEDS system architecture.

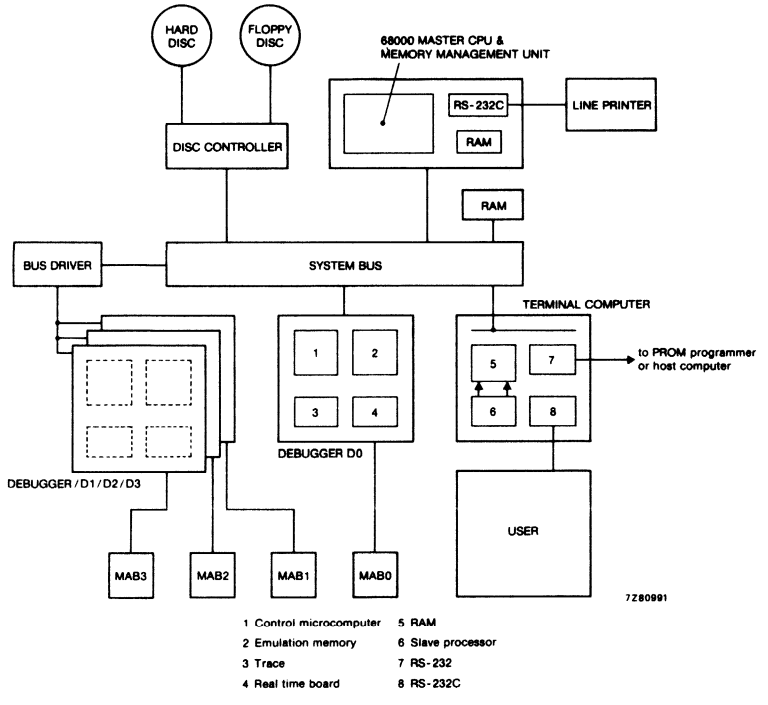


Fig. 6. PEDS system architecture

4.2 PEDS software

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 7. illustrates PEDS software organisation.

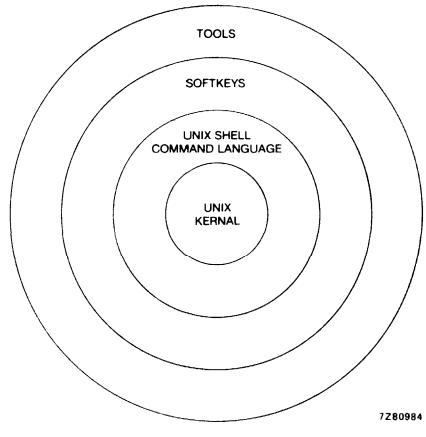


Fig. 7. PEDS software organisation.

5.0 MICROCOMPUTER INSTRUCTOR PM4300

The PM4300 Microcomputer Instructor is a universal tool for designing, prototyping and debugging microcontroller-based products. The PM4300 may be used singularly to write and debug programs in machine code or linked by an RS-232 port to a host computer or development system such as PMDS. The host may then be used to edit and assemble source program code which is later down-loaded to the PM4300 and debugged in the prototype system.

Plug-in, dedicated, personality modules permit the support of many different microcontrollers. All that is required when exchanging microcontrollers is to plug in a new module. Learning new command formats is not required as command and data-entry operations are standard for all modules. Although there are variations in hardware, all modules perform basically the same function. The MAB personality module, for example, has extra facilities for program disassembly and is also able to program EPROMs.

The personality module contains the target microcontroller. Signals are brought out through the Processor Cable which match the pin out of the test microcontroller. When the cable is connected to the microcontroller socket of a prototype system, all Instructor Commands can be used for real-time execution and single-stepping of programs in the test system. A prototype breadboard area is included on top of the Microcontroller Instructor for fast development of trial circuits. Connection to this area is made via the Processor Cable, which is attached to the rear of the unit.

The Microcomputer Instructor is completely self-contained. Control of system environment includes manual operation of I/O devices and interrupt generation. This control is provided using only the keyboard, on-board switches and LEDs.

Programs are entered through the 'hex' keyboard, or downloaded from another system or an audio cassette. Once entered, the operator can use commands to examine and change memory data or registers, set breakpoint and single-step a program.

A software breakpoint can be set at any point and the program then executed up to it. By single stepping through the program faults can be identified and corrected as they occur. In this mode, the contents of a specific memory location or a target register are continuously shown on the alphanumeric display as the program is executed. In the MAB8400 personality module the address and mnemonic of the executed instruction are shown in the alphanumeric display after each step. At the same time, the contents of all registers and the mnemonic of the instruction can be dumped via the RS-232 interface to an external device, such as a printer or CRT. An AUTO STEP function allows fast stepping with pre-programmed delay intervals from 20ms to 6 s.

Checking peripheral IC operation, and I/O devices in particular, is acheived using the Processor Cable together and breadboard panel. Microcontroller Instructor commands may read or write I/O data, permitting I/O checks to be carried out in hardware rather than time consuming software. A built-in 8-bit parallel I/O port allows input and output routines to be written without the need to build the relevant circuit. Input words can be created by using the 8 switches located on the top of the Microcomputer Instructor; when the port is used for output, the output word is shown on 8 LEDs.

Interrupt keys permit manual generation and execution of user-written interrupt routines. By using the single step command together with the interrupt facility, single stepping through interrupt routines is possible, with control eventually returning to the main program. Writing and debugging interrupt routines is simplified using these functions. When the vector interrupt switch on the Microcontroller Instructor is in the 'up' position, vector addresses can be set and changed for when the target microcontroller is responding to a vectored interrupt.

Provisions exist for expansion of the Microcomputer Instructor. An independent system bus together with the Processor Cable which brings out target microcontroller control signals, permits system facilities to be expanded without interference to the microcontroller under development.

5.1 System architecture

The PM4300 Microcomputer Instructor can be considered in two parts; the main unit and the personality module. (Fig. 3) An 8041 microcontroller drives the main unit and is treated as an intelligent peripheral by the target microcontroller. The time consuming tasks af ASCII conversion, key debounce, formatting and display scanning are all handled by the 8041. Up to 64 keys (36 are assigned at present) and the 16 digit, 14 segment vacuum display are supported by the 8041, allowing the main unit of the Microcontroller Instructor to be target microcontroller independent.

The bus structure of the main unit is a permanent universal bus containing 16 data lines, 16 address lines and control signals. All internal RAM and other functions reside on the system bus. Standard user memory is a 1 or 2Kbyte of static RAM. Two extra sockets in the back of the Microcontroller Instructor allows for an optional 1 or 2 Kbytes of additional RAM to be added. Moreover the MAB8400 personality module has an additional 2 Kbytes on-board RAM, so that up to 6 Kbytes programs can be emulated using this module. Figure 8 shows the PM4300 system architecture.

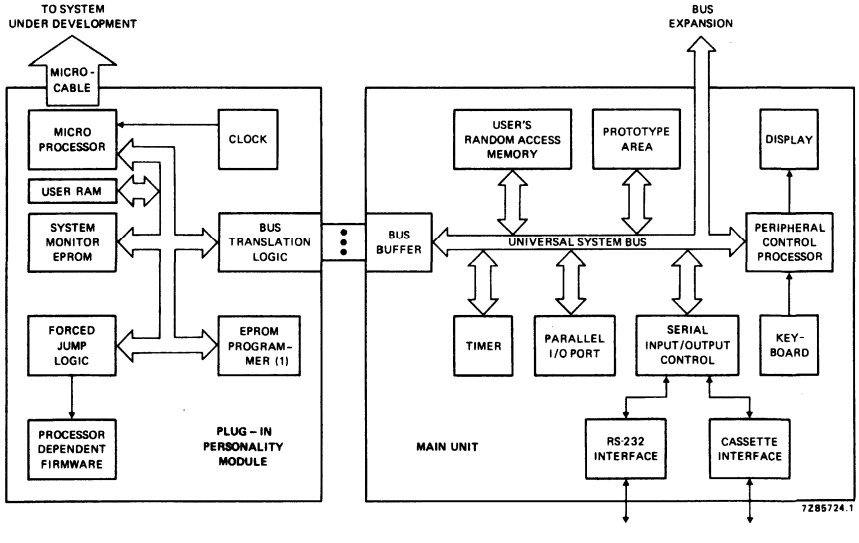


Fig. 8. PM4300 System Architecture.

In order to develop I/O programs, an 8-bit parallel I/O port is also connected to the bus. The front panel switches used to set input words can also be used to select an interrupt vector for Page 0 interrupts. Both the RS-232 and cassette interfaces are connected to the system bus via a Serial I/O control and both contain an 8251 UART. The UART converts parallel data from the target microcontroller to serial data for the I/O device and vice-versa. The baud rate used for the RS-232 interface is selected by a DIP switch located under the personality module. When using the cassette interface, the baud rate is set at 300 baud to meet the Kansas City Standard.

The personality module contains the target microcontroller, Processor Cable, system monitor, bus translation logic and forced jump sequencer. System monitor allows the user to enter and alter programs, execute programs in continuous or single step modes, the time execution of routines and other auxiliary functions. Monitor commands are entered via the control keys and hexadecimal keyboard; responses are displayed on the 16-digit display. Entry to the monitor is carried out via the forced jump logic located on the personality module. Three functions are provided by the forced jump logic;

-monitor entry after power-up or a monitor (MON) key entry

-monitor return after executing the single-step function

-return from breakpoint detection.

A disassembler and PROM programmer are also incorporated on the PM4300 and the results of a disassembly may be displayed on the alphanumeric read-out or alternatively an external CRT or printer. The PROM programmer is able to read, write and verify 2716, 2732 and 2732A EPROMS.

During a power-on sequence, the first function causes the hardware to issue a RESET instruction and inhibit RAM memory. After RESET is released, the jump sequencer is enabled and forces a jump to the start of the monitor.

5.2 Technical specifications

Operating Speeds	Full rated speeds of microcontrollers
Program Panel	16-digit, 14-segment alphanumeric display, 32-key Command and Hexadecimal keyboards
Internal Memory	User: 1 K or 2 Kbytes/words of RAM at 0000H System: 2 Kbytes/words of ROM 128 bytes/words of RAM
External Memory	Expandable up to 64 Kbytes
I/O	One serial I/O port wired for RS-232C and Kansas City Standard audio cassette interface One parallel I/O port wired to 8-input switches and 8-LED outputs

Interrupts	Manually generated by a personality module key, or a real-time clock input. I/O port switches can provide an interrupt vector.
Baud rate	Switch selectable from 110 to 9600 baud
Prototype Area	Holds up to thirteen 16-pin devices plus the Emulator Cable
Monitor	Occupies 4 Kbytes or 2 K words, depending on the personality module installed
Breakpoint	Hardware compare on address. This can be used as a non-break trigger for a storage scope or logic analyzer.
Step	Single step or automatic-step. Variable step rate
Power	100, 115, 220, 240 V a.c, 50 to 60 Hz
Temperature	Operating: 0 °C to 32 °C With user circuits using up to 0,5A at 5 volts 0 °C to 40 °C with no user circuits. Storage: -40 °C to 75 °C (-40 °F to 167 °F)
Humidity	95% RH 15 °C to 40 °C (59 °F to 104 °F)
Physical	Width: 14,25" (36,2 cm) Height: 3,75" (9,5 cm) Depth: 14,0" (35,6 cm) Weigth: approximately 10lbs. (4,5 kg)

5.3 Command keys

The command keyboard of the Microcontroller Instructor is summarised in the table and described in the following paragraphs.

Table of Functional command keys

Key	Description
AUTO STEP	Automatic single-step through user program
BREAK	Set hardware breakpoint
DUMP	Output to audio cassette or host computer or commanding the EPROM programmer
ENTER LAST	Enter and decrement
ENTER NEXT	Enter and increment
GOTO	Execute user program
I/O	Read or write to an I/O port
LOAD	Input from audio cassette or host computer
MEMORY	Display or alter memory
MON	Return to the monitor program
MOVE	Move block of data
PATCH	Quick data input
REG	Display or alter registers
RESET	Go to reset location or target processor
STEP	Single-step
TIME	Time execution of user program

KEY	DESCRIPTION
AUTO STEP	The AUTO-STEP key executes a single instruction in the user's program. After execution, the system returns to the monitor program, displays the address of the next instruction to be executed and either the last register, memory location, or I/O location examined, or time. In the MAB84XX personality module, it is also possible to display the address of the current instruction followed by the mnemonic of that instruction. After a delay, the keyboard is read and if a key has not been pressed, another instruction in the user' program is executed. The single-step function is normally performed twice a second but the speed can be varied.
BREAK	The BREAK key sets or examines a breakpoint in the user's program. The breakpoint is a hardware function of the Microcontroller Instructor that halts the execution of the user's program at a specific address, after executing the instruction at that address. The Break instruction does not alter system status.
DUMP	The DUMP key outputs a segment of user memory in Hex to the serial I/O port and the audio cassette output jack. The DUMP key allows the user to save programs on magnetic tape, or to upload them onto a larger system if the step rate is zero. If the step rate is non-zero, memory is displayed on a terminal connected to the serial port. The baud rate is variable from 110 to 9600 baud for RS-232 and fixed at 300 baud for the cassette interface. The cassette interface meets the Kansas City Standard for recording. The dump key in the MAB84XX personality module has two functions: • dumping user program memory to the serial interface (or audio cassette) in 3 ways (selectable by the step rate): - dump of program memory in Tek hex object code format - display of program memory on CRT - disassembly of program memory on CRT (not standard). • Commanding the EPROM programmer (in conjunction with the PM4300 port switches): - writing user memory to a 2716 or 2732 EPROM - comparing user memory with a 2716 or 2732 EPROM - reading 2716 ro 2732 EPROM contents into user memory.
ENTER LAST	The ENTER LAST key is used to display or alter the previous address or contents of a memory location or register. This key is always used in conjunction with another key
ENTER NEXT	The ENTER NEXT key is used to display or alter the next address, or contents of a memory location or register. This key is always used in conjunction with another key.
GOTO	The GOTO key terminates the monitor mode and transfers control to the user's program at the address supplied after the GOTO key is pressed. Program execution continues until a breakpoint is encountered or the MON key is pressed.
I/O	The I/O key allows the user to read a value from any I/O port and display it, or output a value to an I/O port. This function enables easy initialisation of peripheral support chips when used on the prototype board.

LOAD	The LOAD key loads the Tek hex formatted cassette files or data from the serial I/O port into memory, as specified in the file. An offset may also be specified with the command to allow relocation of the data in memory. Optionally the MAB84XX personality module also allows INTEL hex object code formatted data files to be loaded into memory.
MEMORY	The MEMORY key allows the user to display and alter user memory. The user specifies an address of a memory location and can then examine or change the data at that location.
MON	The MON key causes the forced jump logic in the Microcontroller Instructor to force a jump instruction sequence onto the data bus, resulting in a return to the monitor program.
MOVE	The MOVE key allows the user to move segments of code from one part of memory to another for editing or patching purposes.
PATCH	The PATCH key allows entry of sequential memory locations after supplying a starting address. Each keystroke is entered as a nibble of data.
REG	The REG key places the monitor in the examine and alter registers mode. In this mode, the user may examine and alter the contents of any of the general-purpose registers on the processor being used. The keys of the hexadecimal keyboard are used for the register selection.
RESET	The RESET key resets the target processor, causing program execution to begin at the reset address. This does not reset the Microcontroller Instructor board into the monitor mode. Use of this key overrides monitor control, and functions such as breakpoint detection are not supported.
STEP	The STEP key executes a single instruction in the user's program. After execution, the system returns to the monitor mode, displays the address of the next instruction to be executed and either the last memory location or register examined.
TIME	The TIME key allows the user to determine the instruction execution time spent in the user RAM space.

6.0 PMDS & PM4300 CONFIGURATIONS

When developing microcontroller systems, an instructor such as the PM4300 may be all that is required at the start of a project. But as the project grows, a larger and more sophisticated development system may become necessary. The intention of this section is to show how, starting with a PM4300, the function of a microcontroller development system can be improved and expanded.

6.1 PM4300

As explained in section 5.0, this unit can be used for PROM programming and debugging a prototype system in hex. Figure 9. overleaf, shows the connection of an optional CRT screen to display emulation status and disassembly (only for MAB8400).

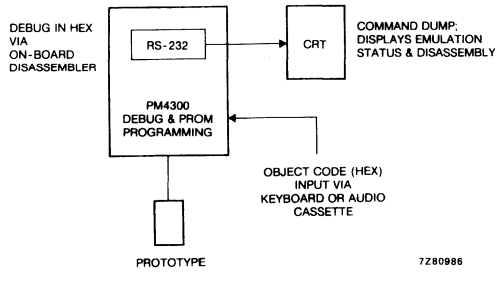


Fig. 9 Connection of PM4300 to CRT screen.

6.2 PM4300 and host computer

The connection of a host computer allows source editing and assembly on the host computer with down loading on to the PM4300 for emulation and debugging. Again, an optional CRT display can be connected to the PM4300 for monitoring emulation status and this is illustrated in figure 10.

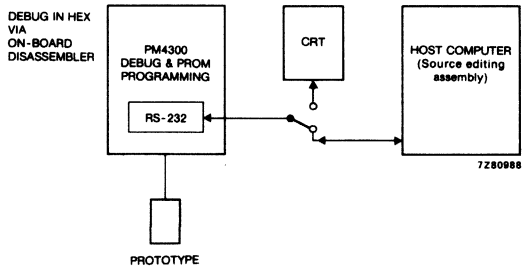


Fig. 10. Connection of PM4300 to host computer and CRT screen.

6.3 PM4300 and PMDS

Even if the development system is upgraded to include a PMDS, the PM 4300 still provides a useful function. When debugging more than one microcontroller at the same time, PMDS and PM4300 can be connected to the prototype system to perform simultaneous emulation and debugging, this is shown in figure 11.

The program is first edited, assembled and linked on the PMDS and then down loaded on to the PM4300, as in the previous example. But in this case, the PMDS is also connected to the protype system via a Microcontroller Adapter Box (MAB) and is hence also able to debug the prototype.

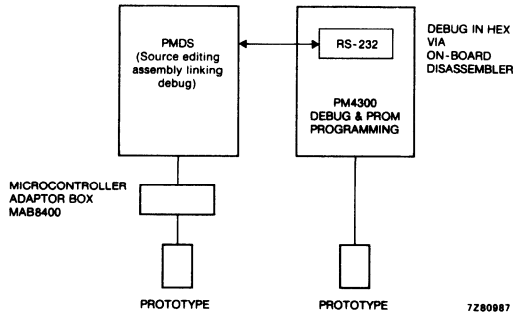


Fig. 11. Connection of PM4300 to PMDS

7.0 MULTI-DEBUGGING WITH PMDS 1

One of the major features of PMDS 1 is its ability to debug a multi-microcontroller configuration. When considering microcontrollers, such as the MAB8400, connected via an I^2C (Inter IC) bus, this facility becomes extremely necessary

Using the multi-debug facility allows PMDS 1 to monitor any component (up to a maximum of 4 components) during an emulation run. The additional hardware required is an extra MAB, debug-extension unit and Universal debug unit for each additional component (see section 8.4 on PMDS hardware support). Figure 12. illustrates a possible arrangemant.

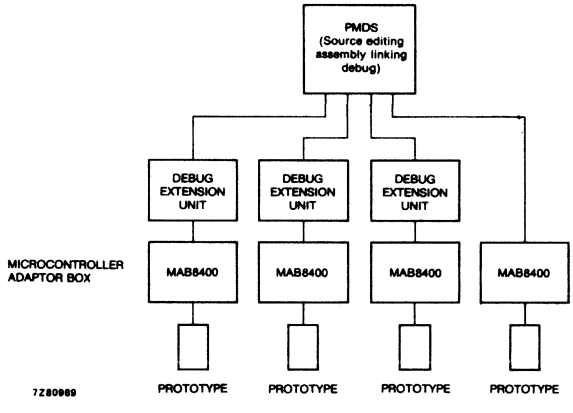


Fig. 12. Connection of PMDS 1 to up to four target components.

8.0 PEDS, PMDS & PM4300 HARDWARE AND SOFTWARE SUPPORT

8.1 PEDS - software support

The following software facilities are available to PEDS:

PM 8394 M Screen editor

PM 8395 M 'C' on host 68000 CPU

PM 8396 M Source code control system

PM 8399 M General purpose communication package.

PASCAL is supported by PEDS as a second high level lanquage. It's prime advantages are superb transportability and very strong type checking. (See data sheets on PASCAL, PL/M and 'C' compilers for PMDS 2/PEDS).

Order numbers
PM 8470 PASCAL for 8080, 8085 and Z80
PM 8476 PASCAL for 6809
PASCAL for PMDS is also included in both of these type numbers.

8.2 PEDS - hardware support

PM 8418 256 Kbyte system memory

PM 4485M 10 Minifloppies (96 tpi)

Emulator options (see universal debugger PM 8400 data sheet for details)

PM 8440 MAB No-break switch

PM 4475 bus extender card -this card provides an interface to the first extension debug unit

PM 8405/20 extension debug unit

PM 8410 Trace board -this board gives logic state trace (255 x 48 lines) and event counter possibilities

PM 8412 64 Kbyte dyn. emulation mem.

PM 8418 256 Kbyte dyn. emulation mem.

PM 8414 16 Kbyte fast emulation mem.

PM 8419 64 Kbyte fast emulation mem.

Please note, the above emulator options are also available to PMDS 1/2

8.3 PMDS - software support

With PMDS the programmer can use either assembler or various high level lanquages, or a combination of both. PMDS 1 supports PASCAL for the development of various microcontrollers (see PEDS for order numbers). In addition to PASCAL, PMDS 2 under UNIX, supports FORTRAN, 'C' and PL/M 86 ('C' and PL/M 86 are used in PMDS 2 for 16-bit microcontroller development).

For the MAB8400 & PCF8500 families the following support is available;

PM 8367 Cross-assembler
PM 8421 Microcontroller adapter box
PM 8387 Debugger

For the MAB8048 family the following support is available;

PM 8461 Cross-assembler
PM 8421 Microcontroller adaptor box
PM 8481 Debugger

8.4 PMDS - hardware support

The PM 8405 Debug extension unit

When two or more debug units are to be controled from a PMDS 1 or from the single workstation of a PMDS 2, the PM 8405 must be fitted with a PM 8400 UDU Universal debug unit, plus optionally a PM8410 Trace card and up to four PM 8411-8419 Emulation memory cards. A prerequisite for the connection of the first PM8405 is the presence of the PM 4475 Bus extender card in the master system. Connection of the first extension extension debug to the master system is by a pair of ribbon cables. Additional extension debug modules connect from the first in a daisy chain arrangement. A pair of 2m long cables are supplied with each extension module.

The PM 8411-8419 Emulation memory cards

Each UDU 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 130ns including mapping time
- Dynamic; access time 450ns.

The two types of emulation memory may be mixed, subject to certain restrictions. The following memory cards are available:

PM 8411	32	Kbytes dynamic
PM 8412	64	Kbytes dynamic
PM 8413	8	Kbytes fast
PM 8414	16	Kbytes fast
PM 8415	8	Kbytes dynamic
PM 8416	16	Kbytes dynamic
PM 8418	256	Kbytes dynamic
PM 8419	64	Kbytes fast

8.5 PM4300 support

For the MAB8400 the following support is available:

PM 4337 Personality module

and for the MAB 8048 family:

PM 4321 Personality module.

13. Application notes

CONTENTS – APPLICATION NOTES

APPENDIX 1

1.0 CLOCK GENERATION

1.1 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 Vss) over the area of the clock circuit.

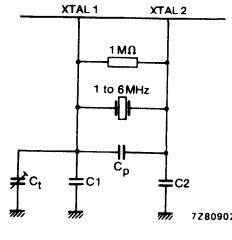
1.2 Crystal clock generator

Outline of MAB84XX 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 84XX 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 figure 1.



C_t is optional for trimming

Fig. 1 Circuit diagram MAB84XX

Approx. component values :

C1 = C2 = 27 pF

C_p = $\leqslant$ 6,75 pF (parasitic capacitance)

XTAL (AT cut) = The fundamental frequency crystal (governs the operating frequency of the device)

R_s = $\leqslant$ 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 microcomputer 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) Aging (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)

1.3 Clock generation using ceramic resonators

When implementing ceramic resonators, the circuit used is the same as that for crystal oscillation (see Fig 1.)

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)

1.4 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 figure 2.

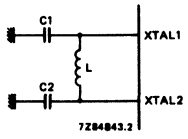


Fig. 2. circuit diagram for LC oscillator

For clock generation using LC networks, the values of L & 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

1.5 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 84XX device, the signal may be amplified and distributed to dependant devices.

1.5.1 Central oscillator

Figure 3 shows a suitable circuit for a central CMOS oscillator.

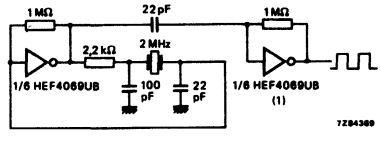


Fig. 3. A CMOS oscillator

The HEF4069UB (seen in Figure 3) must be used as an inverter as buffered devices do not readily oscillate.

1.5.2 One MAB84XX driving another seperate MAB84XX

Figure 4 shows the circuit for clocking two MAB84XX devices

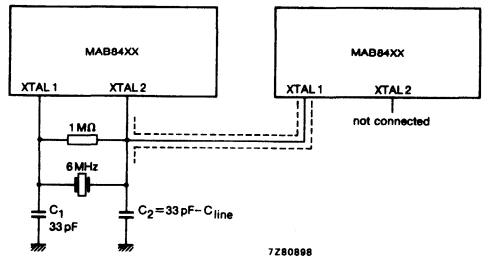


Fig. 4. Clocking two MAB84XX devices

Two MAB84XX devices may be clocked using a single crystal source from either of the devices. The second 84XX is fed with the clock pulse to it's XTAL1 input sourced by the originating XTAL2 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.

1.5.3 Buffered on-chip oscillation

The following diagrams illustrate some practical circuits:
1) Using TTL devices

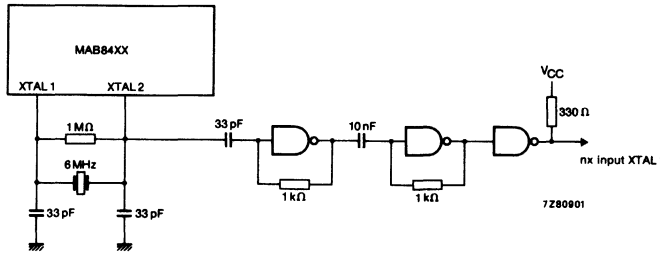


Fig 5. TTL linear amp & TTL buffer (SN74 series).

Maxm. load at 6 MHz = 12 MAB84XX devices
load capacitance $\leqslant$ 500 pF

2) Using CMOS devices

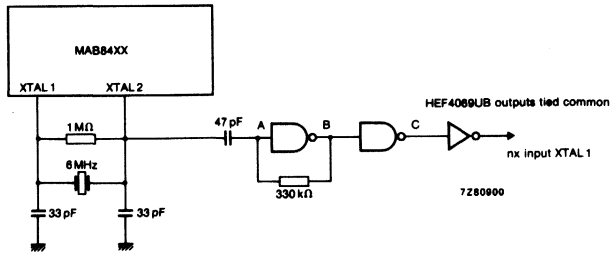


Fig 6. CMOS amp & CMOS buffer (HEF4069UB)

Maxm. load at 6 MHz = 3 MAB84XX devices
load capacitance = 140 pF
wiring capacitance generated at points A, B and C is less than 8 pF.

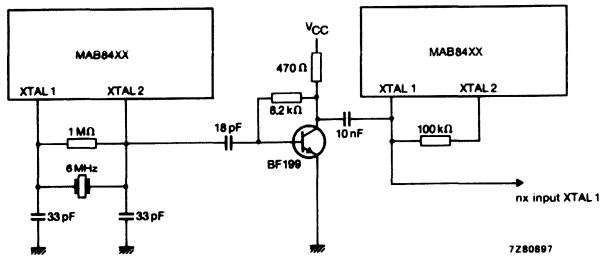


Fig. 7.

Bipolar amplifier stage offers good pulse reproduction and d.c. isolation for subsequent XTAL1 inputs. (Transistor BF199)

At 6 MHz : Maxm. series load with 100k ohm resistance = 1 MAB84XX
Maxm. parallel load across 100k ohm resistance = 3 MAB84XX
Total load capacitance $\leqslant$ 160 pF.

Figure 8 illustrates capacitive load of clock line.

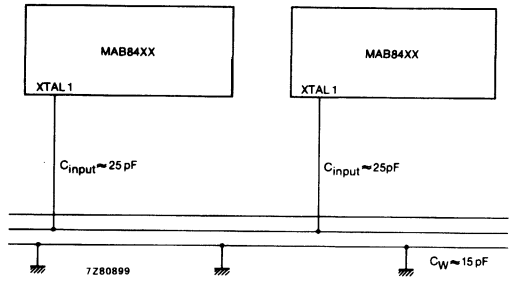


Fig. 8.

Capacitive load for each MAB84XX = 40 pF (approximate value)

C_{input} = input capacitance

C_W = wiring capacitance

APPENDIX 2

SOLDERING TECHNIQUES

In circuit construction, manufacturers today use a mixture of Surface mount and through-hole components. As surface mount technology gains in favour as forecast, and more and more IC's 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

1.0 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 it's temperature is below 300 C it must not be in contact for more than 10 seconds; if between 300 C and 400 C, for not more than 5 seconds.

2) By dip or wave

The maximum permissable temperature of the solder is 260 C; this temperature must not be in contact with the joint for more than 5 seconds. The total contact time of successive solder must again not exceed 5 seconds.

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.

2.0 SURFACE MOUNT PACKAGES

2.1 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 C to 230 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 seconds is sufficient to ensure good solder joints and evaporation of the binder fluid.

After soldering, the substrate must be cleansed of any remaining flux.

2.1.1 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 seconds up to 300 C, or 5 seconds up to a maximum of 400 C. When using the correct tools, all pins may be soldered in one operation within 2 to 5 seconds between a temperature of 270 C to 320 C.

2) By dip or wave

The maximum permissable temperature of the solder is 260 C. The total permissable time of immersing the whole package in the bath is 10 seconds, if it is allowed to cool down to less than 150 C within 6 seconds.

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 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 are presently manufactured in three distinct sizes;

- (PLCC) Plastic leaded chip carrier
- (SO) Small Outline package
- (VSO) Very Small Outline package.

2.1.2 SO packages

When placing surface mounted packages such as SO IC's, 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.

2.1.3 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 it's length and to a height equal to half the thickness of the lead.

2.1.4 PLCC packages

For IC's in plastic leaded chip carriers (PLCC's), part of the lead and it's 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 3

1.0 MICROCONTROLLER DECOUPLING

Decoupling and surge voltage protection

Surge voltages in power lines are caused by switching inductive loads, and direct or indirect lightning strikes. Other sources of spurious spikes 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).

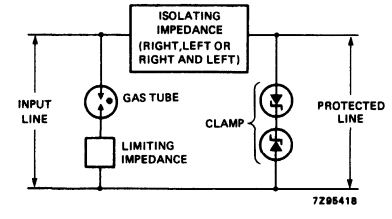


Fig. 1. A gas tube with series limiter, isolating impedance and clamp are the components necessary for surge protection. Gas discharge tubes with a wide range of breakdown voltages are available from manufacturers.

With the 84XX, 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 IC's 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 INTEL MDS 2 system. (format INTELLEC .HEX)

- 5,25 inch floppy disc;
  Double sided, double density, written on a PHILIPS PMDS (1 or 2) system in INTEL format.

Suitable EPROMS include; 2716, 2732 (A), 2764, 8748, 8749, 8751.

*Note: When submitting code to the manufacturers, use the official 'order entry' forms.

APPENDIX 4

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

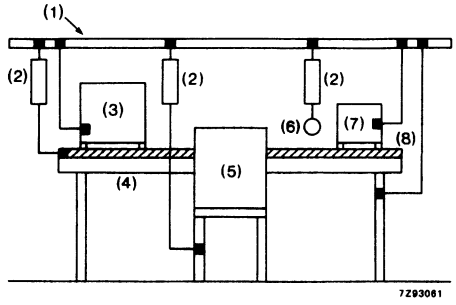
The work station (Fig. 1) is a working area constructed for the safe handling of electrostatic sensitive devices. It has a work bench with a conductive surface or with the surface covered by an antistatic sheet. A typical resistance value for the bench surface is 1 kΩ to 0.5 MΩ per cm^2. The floor covering 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. I^{2}C bus support chip set

List 1 represents the complete range of our I²C support chips. As various components are still in development, the availability status at the time of going to press, is shown in list 2.

LIST 1

Universal IC's

A. 8-Bit Microcontrollers

MAB 8401	: No ROM	128 byte RAM	(Piggy-back or bond-out in PLCC)
MAB 8411	: 1K byte ROM	64 byte RAM	
MAB 842/1/2	: 2K byte ROM	64 byte RAM	
MAB 844/1/2	: 4K byte ROM	128 byte RAM	
MAB 8460/1	: 6K byte ROM	128 byte RAM	
PCB 84C00	: CMOS version		

B. 16-Bit Microprocessor

SCC 68070 : 68000 Compatible

C. Peripheral IC's

SAA 5240	: CCT(Teletext decoder)
SAA 3028	: Infrared remote control transcoder
PCF 8200	: Speech syntheziser
PCF 8570	: 256 x 8-Bit static RAM
PCF 8571	: 128 x 8-Bit static RAM
PCF 8573	: Clock/Calender chip
PCF 8574	: 8-Bit I/O expander /LED driver
PCF 8576	: 1 to 4 MUX/LCD driver (160 seg.)
PCF 8577	: 1 to 2 MUX/LCD driver (64 seg.)
PCF 8578	: Dot-matrix LCD controller
PCF 8579	: Dot matrix LCD driver
PCB 8582	: 256 x 8-Bit E²PROM
PCF 8591	: Multi-channel ADC/DAC

Special Application IC's

A. Microcontrollers

	PCD 3315	: Low-voltage CMOS (telephone) Microcontroller
*	PCD 3343	: Low-voltage CMOS (telephone & multipurpose) Microcontroller

B. Peripheral IC's

SAB 3035	: 8-Bit DAC, CITAC
SAB 3036	: NO DAC, CITAC
SAB 3037	: 4-Bit DAC, CITAC
TDA 8420	: TV-tone processor
TDA 8440	: Analogue switch
TDA 8442	: 4-line DAC (6-Bit)
TDA 8460	: Colour decoder

TEA 6000 : FM/ZF Tuning interface
PCD 3311 : MFV Generator (telephone)
PCD 3312 : " " " "
PCX 0330/5 : Gate array family
PCX 0450/5 : " " "
PCX 0700/5 : " " "
PCX 1100/5 : " " "

* PCD 3343 is hardware identical to PCB84C00, and to all practical ends may be considered as the same device.

C-Bus IC's

C-Bus's 3-line structure supports I^2C software.

PCF 2100 : Duplex LCD driver, 40 seg.
PCF 2110 : Duplex LCD driver, 60 seg + 2 LED's.
PCF 2111 : Duplex LCD driver, 64 seg.
PCF 2112 : Static LCD driver, 32 seg
SAA 1060 : LED driver, 16-Bit static, 32-Bit dynamic
SAA 1061 : 16 line output driver
SAB 3013 : 6 line DAC
SAF 3019 : Clock/calander
TEA 1017 : 13 line display driver

LIST 2.

Type	Description	Data Sheet issue date	Status	In Production
MAB 8401	µC, No ROM 128 byte RAM	5/85	available	yes
MAB 8411	µC, 1K byte ROM 64 byte RAM	5/85	available	yes
MAB 842/1/2	µC, 2K byte ROM 64 byte RAM	5/85	available	yes
MAB 844/1/2	µC, 4K byte ROM 128 byte RAM	9/84	available	yes
MAB 8461	µC, 6K byte ROM 128 bytes RAM	5/85	available	yes
PCB 84C00	CMOS 8400 family	----	III. Q 1986	under devel.
PCB 84C20	" " " "	----	II.Q 1986	" " "
PCB 84C40	" " " "	----	I. Q 1986	" " "
PCB 68070	16-Bit 68000 compatible	----	II. Q 1986	" " "

Type	Description	Data sheet issue date	Status	In Production
PCF 8200	Speech syntheziser	3/85	IV. Q 1985	I. Q 1986
PCF 8570	256 x 8-Bit RAM	5/85	available	yes
PCF 8571	128 x 8-Bit RAM	5/85	available	yes
PCF 8573	Clock/calender	12/84	available	yes
PCF 8574	8-Bit I/O expander	12/84	available	yes
PCF 8576	1:(1-4) MUX/LCD driver, 160 seg.	1/85	available	yes
PCF 8577	1:(1-2) MUX/LCD driver, 64 seg.	12/83	available	yes
PCF 8578	Dot-matrix LCD controller	----	IV. Q 1986	under devel.
PCF 8579	Dot-matrix LCD driver	----	IV. Q 1986	" " "
PCB 8582	256 x 8-Bit E^2PROM	12/84	IV. Q 1985	I. Q 1986
PCF 8591	Multi-channel ADC/DAC	5/85	available	I. Q 1986
PCF 2100	Duplex LCD driver, 40 seg.	12/83	available	yes
PCF 2110	Duplex LCD + 2 LED driver, 60 seg.	12/83	available	yes
PCF 2111	Duplex LCD driver, 64 seg.	12/83	available	yes
PCF 2112	Static LCD driver, 32 seg.	12/83	available	yes
SAA 1057	Frequency Synth.	8/83	available	yes
SAA 1060	LED driver, 32-Bit dyn, 16-Bit static,	1/84	available	yes
SAA 1061	16 line output driver	1/84	available	yes
SAB 3013	6 line D/A conver.	1/82	available	yes
SAF 3019	Clock/calendar	6/83	available	yes

Type	Description	Data Sheet issue date	Status	In Production
TEA 1017	13 line series to parallel conver.	10/84	available	yes
PCD 3311	MFV/Modem generator	12/83	available	yes
PCD 3312	MFV/Modem generator	12/83	available	yes
PCD 3315C	Telephone control.	----	available	IV. Q 1985
PCD 3343	Tel & Multi applic.	8/84	available	yes
PCX 0330/5	Gate array family	4/83	available	yes
PCX 0450/5	" " " "	"	" " "	" "
PCX 0700/5 PCX 1100/5	Customised " applications "	" "	" " " " " "	yes
SAA 1300	5 line series to parallel conver.	6/82	available	yes
SAA 3028	Infra-red remote control transcoder	1/84	available	yes
SAA 5240	CCT (Teletext decoder)	8/84	available	yes
SAB 3035	8 line DAC, CITAC	6/83	available	yes
SAB 3036	CITAC without DAC	6/83	available	yes
SAB 3037	4 line DAC, CITAC	6/83	available	yes
TDA 8420	TV tone processor	6/85	III. Q 1985	IV. Q 1985
TDA 8440	SCART switch	11/84	available	III. Q 1985
TDA 8442	4 line DAC (6-bit)	11/84	available	yes
TDA 8460	Colour decoder	----	I. Q 1986	under devel.
TEA 6000	FM/ZF Tuning interface	9/83	available	yes

15. Data sheets

CONTENTS – DATA SHEETS

N.B. These data sheets are correct at time of going to press. Contact you local Sales Organization for latest data.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

DESCRIPTION

The MAB84XX family of microcontrollers is fabricated in NMOS. The family consists of 8 devices:

- MAB8401 – like 8400 but with 8-bit LED-driver (10 mA), emulation of MAB/F8422/42* possible
- MAB/F8411 – 1K ROM/ 64 RAM bytes plus 8-bit LED-driver
- MAB/F8421 – 2K ROM/ 64 RAM bytes plus 8-bit LED-driver
- MAB/F8441 – 4K ROM/128 RAM bytes plus 8-bit LED-driver
- MAB/F8461 – 6K ROM/128 RAM bytes plus 8-bit LED-driver

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. Two 20-pin versions, MAB/F8422 and MAB/F8442* are also available.

This microcontroller family is designed to be 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 abilities and facilities for both binary and BCD arithmetic.

For detailed information see the "Users manual Single-chip microcomputers" (supplied upon request).

* See data sheet on MAB/F8422/42.

Features

- 8-bit: CPU, ROM, RAM and I/O in a single 28-lead DIL package
- 1K, 2K, 4K or 6K ROM bytes plus a ROM-less version
- 64 or 128 RAM bytes
- 20 quasi-bidirectional I/O port lines
- Two testable inputs: one of which can be used to detect zero cross-over, the other is also the external interrupt input
- Single level vectored interrupts: external, timer/event counter, serial I/O
- Serial I/O that 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
- Internal oscillator, generated with inductor, crystal, ceramic resonator or external source
- Over 80 instructions (based on MAB8048) all of 1 or 2 cycles
- Single 5 V power supply (± 10%)
- Operating temperature ranges:

0 to + 70 °C	MAB84XX	family
−40 to + 85 °C	MAF84XX	family (extended temperature)
−40 to + 110 °C	MAF84AXX	family (automotive temperature)

PACKAGE OUTLINES

MAB8401B: 28-lead 'Piggy-back' package (with up to 28-pin EPROM on top).
MAB8401WP: 68-lead plastic leaded chip-carrier (PLCC) (SOT-188A).
MAB/F8411/21/41/61P: 28-lead DIL; plastic (SOT-117D).
MAF84A11/A21/A41/A61P: 28-lead DIL; plastic (SOT-117D).
MAB8411/21/41/61T: 28-lead mini-pack; plastic (SO-28; SOT-136A).

PINNING

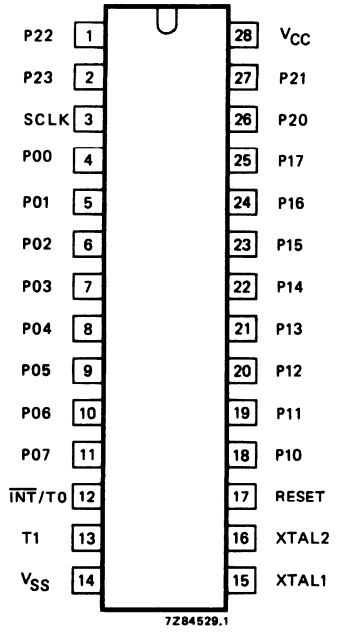
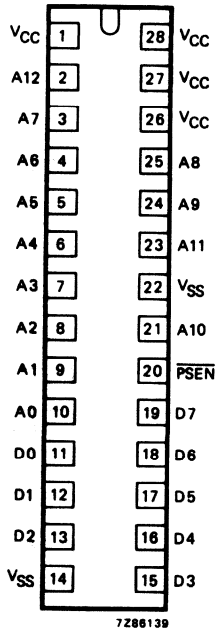


Fig. 1 Pinning diagram for mask-programmable devices MAB8411, MAB8421, MAB8441, MAB8461 and for MAB8400 and MAB8401 'Piggy-back' version bottom pinning (for top pinning see Fig. 2).

PINNING DESIGNATION

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 I/O in serial I/O mode
SCLK	3	Bidirectional clock for serial I/O
$\overline{INT}$/T0	12	External interrupt input (sensitive to a negative-going edge min LOW > 7 clock pulses, min HIGH > 4 clock pulses), testable using the JT0 or JNT0 instructions.
T1	13	Input pin, testable using the JT1 or JNT1 instructions. It can be designated as event counter input using the STRT CNT instruction. It can also be used to detect zero cross-over of slowly moving a.c. inputs.
RESET	17	Input to initialize the processor (active HIGH).
XTAL1	15	Connection to timing component (crystal) that determines the frequency of the internal oscillator. It is also the input for an external clock source.
XTAL2	16	Connection to other side of the timing component.

MAB8400B/01B (top pinning)



PIN DESIGNATION

designation	pin	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 inputs
$\overline{PSEN}$	20	Program store enable

Fig. 2(a) Pinning diagram for MAB8400/01B 'Piggy-back' version top pinning (for bottom pinning see Fig. 1); to access a 2732 or 2764 EPROM.

Note

Access times for ROMS/EPROMS to be below 1 μs.

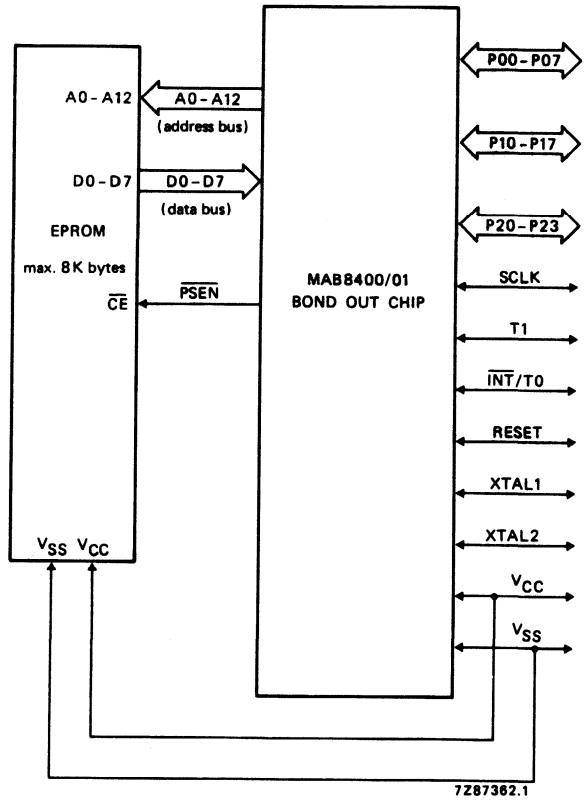


Fig. 2(b) Connection of EPROM to 'Piggy-back' package MAB8400/01B.

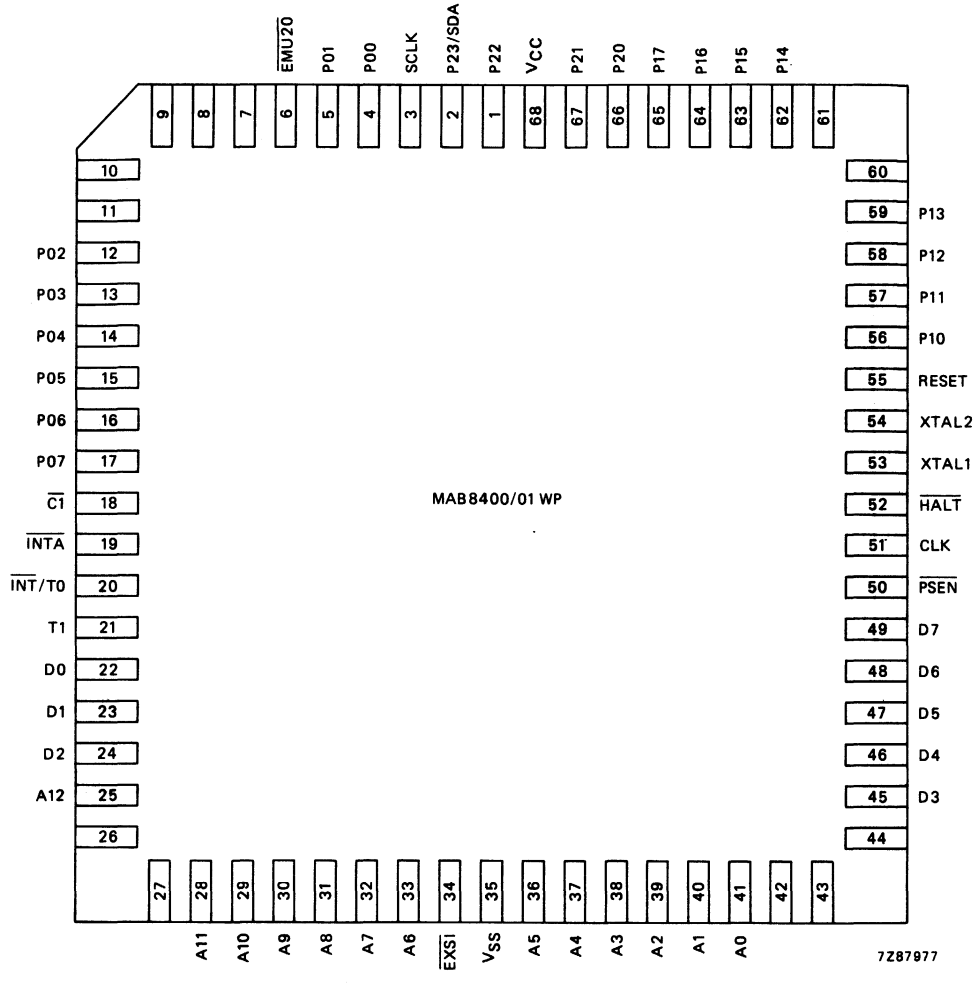


Fig. 3 Pad diagram for PLCC.

CHIP CARRIER DESIGNATION

designation	pad no.	function
V_{SS}	35	**Ground**
V_{CC}	68	**Power supply,** + 5 V
P00–P07	4–5, 12–17	**Port 0,** 8-bit quasi-bidirectional I/O port
P10–P17	56–59, 62–65	**Port 1,** 8-bit quasi-bidirectional I/O port
P20–P22	66, 67, 1	**Port 2,** 4-bit quasi-bidirectional I/O port; P23 is the serial
P23/SDA	2	data I/O in series I/O mode
SCLK	3	Bidirectional clock for serial I/O
$\overline{INT}$/T0	20	External interrupt input (sensitive to a negative-going edge), testable using the JT0 or JNT0 instructions

T1	21	Input pin, testable using the JT1 or JNT1 instructions. It can be designated as event counter input using the STRT CNT inctruction. 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 the timing component
$\overline{EXSI}$	34	External serial I/O interrupt (active-LOW) for emulation of MAB/F8422/42 (8400 back-bias)
A0–A12	41–36, 33–28 25	Program memory address outputs (active HIGH); A0 = LSB, A12 = MSB. Address output change after begin Phi3 of TS8.
D0–D7	22–24, 45–49	Data input lines (active HIGH) used for reading external program memory. D0 = LSB, D7 = MSB.
CLK	51	Clock output buffered from XTAL2. On the positive-going edge the (internal) Phi clock goes HIGH.
$\overline{PSEN}$	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 it 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 opcode fetch cycle (useful for external instruction decoding, real-time trace). Active from start of TS10 of the cycle preceding 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 (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 action and goes into the idle state. Interrupts are *not* sampled in the HALT mode, they are only sampled when the microcontroller is running. Interrupt routines can 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 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 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) (8400 not connected)

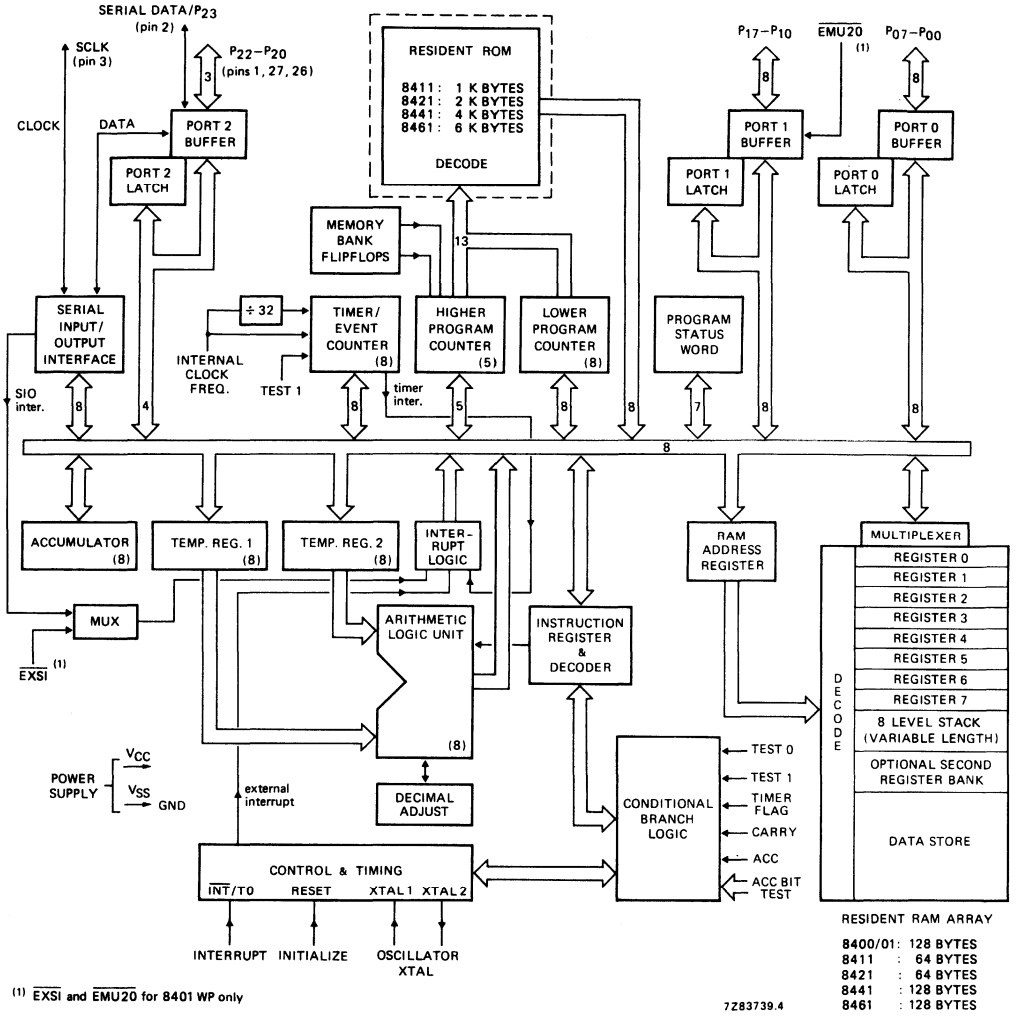


Fig. 4(a) Block diagram of the MAB84XX family.

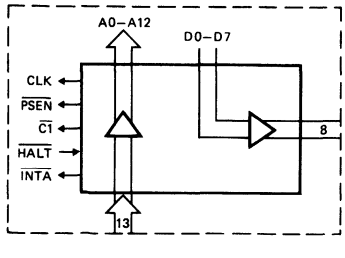


Fig. 4(b) Replacement for dotted part in Fig. 4(a) for the MAB8401WP bond-out version.

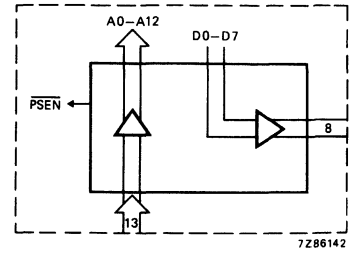


Fig. 4(c) Replacement of dotted part in Fig. 4(a) for the MAB8401B 'Piggy-back' version.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

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
- PCF84C20 – 2 K ROM/64 RAM bytes
- PCF84C40 – 4 K ROM/128 RAM bytes

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. The instruction set is based on that of the MAB8048 and is pin- and instruction set compatible with the MAB8400 family. The microcontrollers have extensive bit handling abilities and facilities for both binary and BCD arithmetic.

For detailed information see the "User manual Single-chip microcomputer".

Features

- 8-bit CPU, ROM, RAM, I/O in a single 28-lead DIL or SO package
- 2 K or 4 K ROM bytes plus a ROM-less version
- 64 or 128 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
- Serial I/O 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 (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 °C

PACKAGE OUTLINES

PCF84C20/40P : 28-lead DIL; plastic (SOT-117D).
PCF84C20/40D : 28-lead DIL; ceramic (CERDIP) (SOT-135A).
PCF84C20/40T : 28-lead mini-pack; plastic (SO-28; SOT-136A).
PCF84C00WP : 28-lead 'Piggy-back' package (with up to 28-pin EPROM on top).
PCF84C00T : 56-lead mini-pack; plastic (VSO-56; SOT-190).

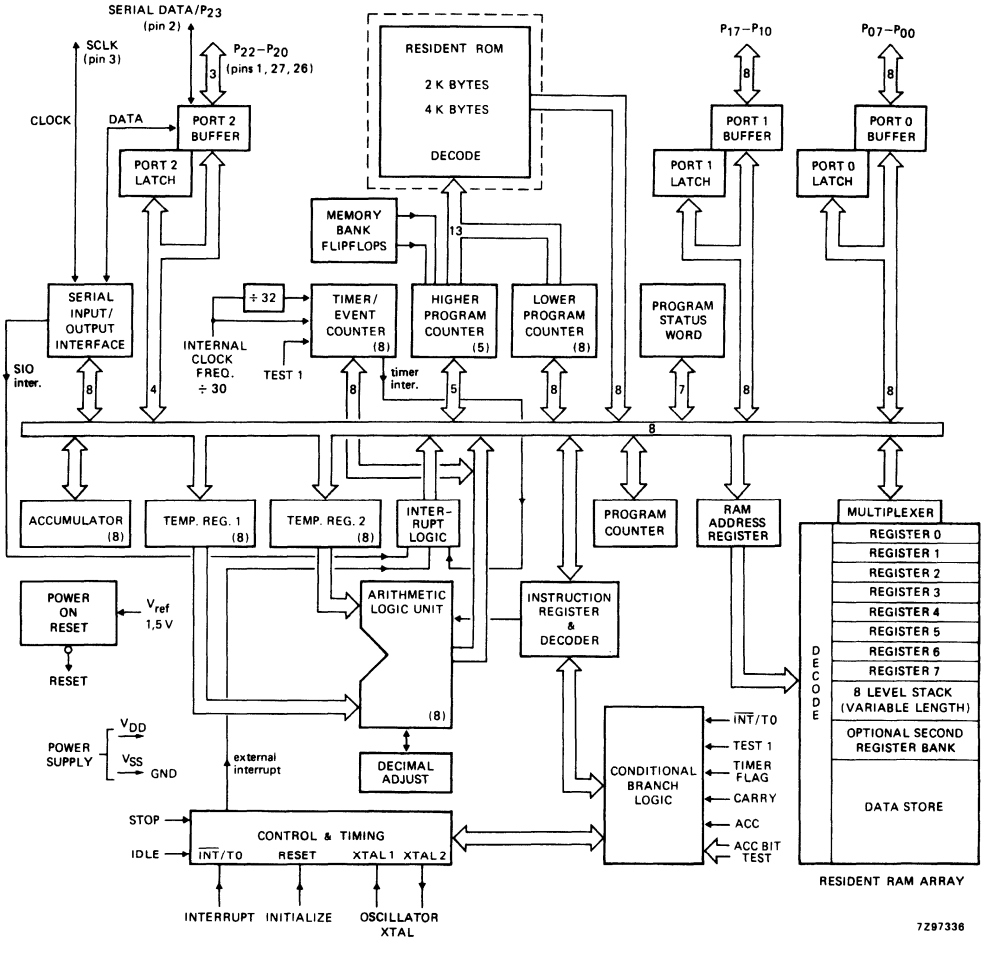
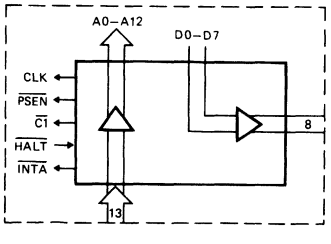
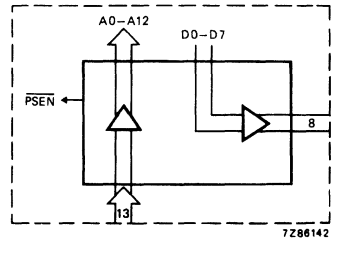


Fig. 1 Block diagram; PCF84CXX.



(a)

Fig. 1a Replacement of dotted part in Fig. 1, for the PCF8400WP bond-out version.



(b)

Fig. 1b Replacement of dotted part in Fig. 1, for the PCF84C00B 'Piggy-back' version.

DEVELOPMENT DATA

PINNING

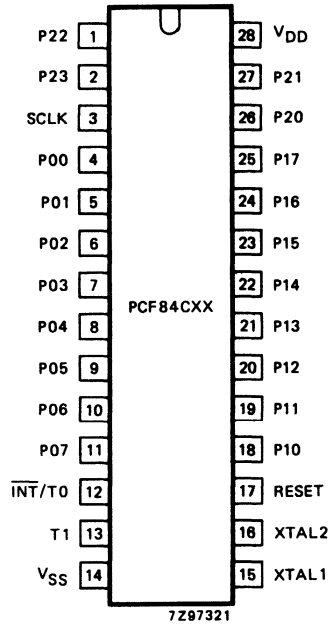


Fig. 2 Pinning diagram; PCF84CXX.

PIN DESIGNATION

3	SCLK	**Clock:** bidirectional clock for serial I/O.
4-11	P00-P07	**Port 0:** 8-bit quasi-bidirectional I/O port.
12	$\overline{INT}$/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.
13	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.
14	V_{SS}	**Ground:** circuit earth potential.
15	XTAL 1	**Oscillator input:** crystal which determines the internal oscillator frequency or the external clock generator.
16	XTAL 2	**Oscillator output**
17	RESET	**Reset input:** used to initialize the processor (active HIGH), or output of power-on-reset circuit.
18-25	P10-P17	**Port 1:** 8-bit quasi-bidirectional I/O port.
26, 27, 1, 2	P20-P23	**Port 2:** 4-bit quasi-bidirectional I/O port. P23 is the serial data input/output in serial I/O mode.
28	V_{DD}	**Power supply:** 2,5 V to 5,5 V

PINNING (continued)

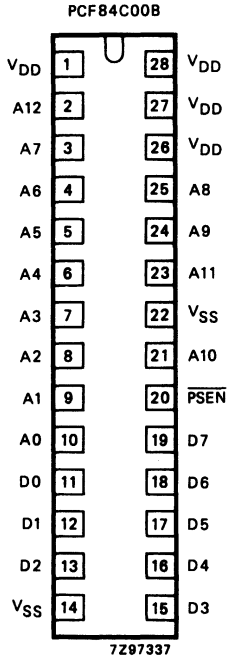


Fig. 3 Pinning diagram: PCF84C00B.
'Piggy-back' version top pinning;
to access a 2732 or 2764 EPROM.

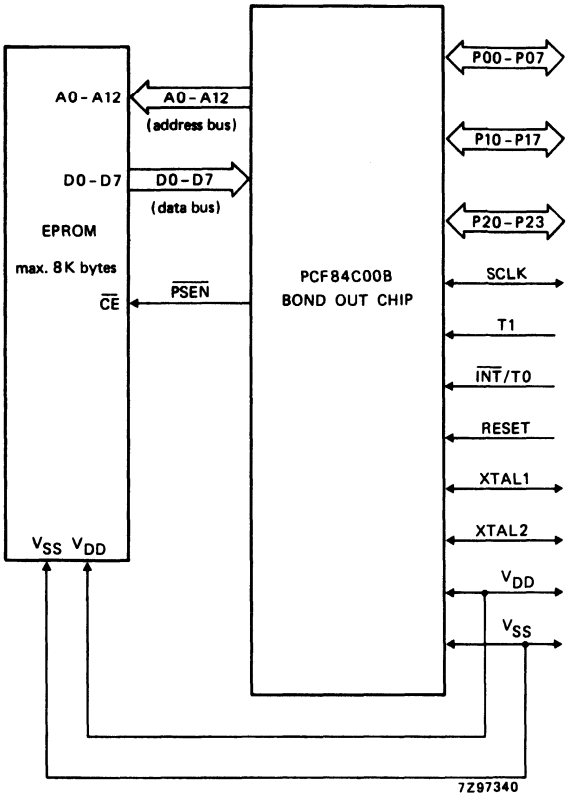


Fig. 3a Connection of EPROM to
'Piggy-back' package PCF84C00B.

PIN DESIGNATION

14, 22	V_{SS}	**Ground**
1, 26-28	V_{DD}	**Power supply**
10-3, 25, 24, 21, 23, 2	A0-A12	**Address outputs**
11-13, 15-19	D0-D7	**Data**
20	$\overline{PSEN}$	**Program store enable**

Notes

1. RAM capacity of PCF84C00B is 256 bytes.
2. Access time for ROMS/EPROMS to be below 7 x $1/f_{XTAL}$.

DEVELOPMENT DATA

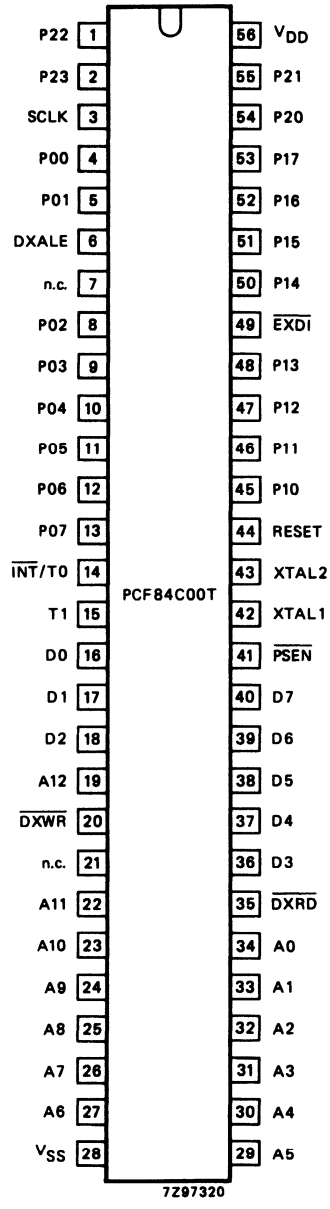


Fig. 4 Pinning diagram; PCF84C00T.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

The MAB8422/8442 is a high-performance microcontroller incorporating dedicated hardware, memory capacity and I/O lines. This dedication means a microcontroller can be economically installed in high-volume products where its main function is control.

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 - 2K ROM/64 RAM bytes plus 8-bit LED-driver
- MAB8442 - 4K ROM/128 RAM bytes plus 8-bit LED-driver

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 ($\overline{INT}$/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. Standard software for I^2C-bus control is available on request.

Features

- 8-bit: CPU, ROM, RAM and I/O
- 20 pin package
- MAB8422: 2K ROM/64 RAM bytes
- MAB8442: 4K ROM/128 RAM bytes
- 13 quasi-bidirectional I/O port lines
- Two testable inputs $\overline{INT}$/T0 and T1
- High current output on P0 (I_{OL} = 10 mA at V_{OL} = 1 V)
- One interrupt line combined with the testable input line $\overline{INT}$/T0
- Single-level interrupts: external, timer/event counter, 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
- Internal oscillator, generated with inductor, 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 5 V power supply
- 0 to 70 °C operating temperature range, also versions for −40 to 85 °C (extended temperature range) and −40 to 110 °C (automotive temperature range)

PACKAGE OUTLINES

MAB/F8422/42P: 20-lead DIL; plastic (SOT-146).
MAF84A22/A42P: 20-lead DIL; plastic (SOT-146).

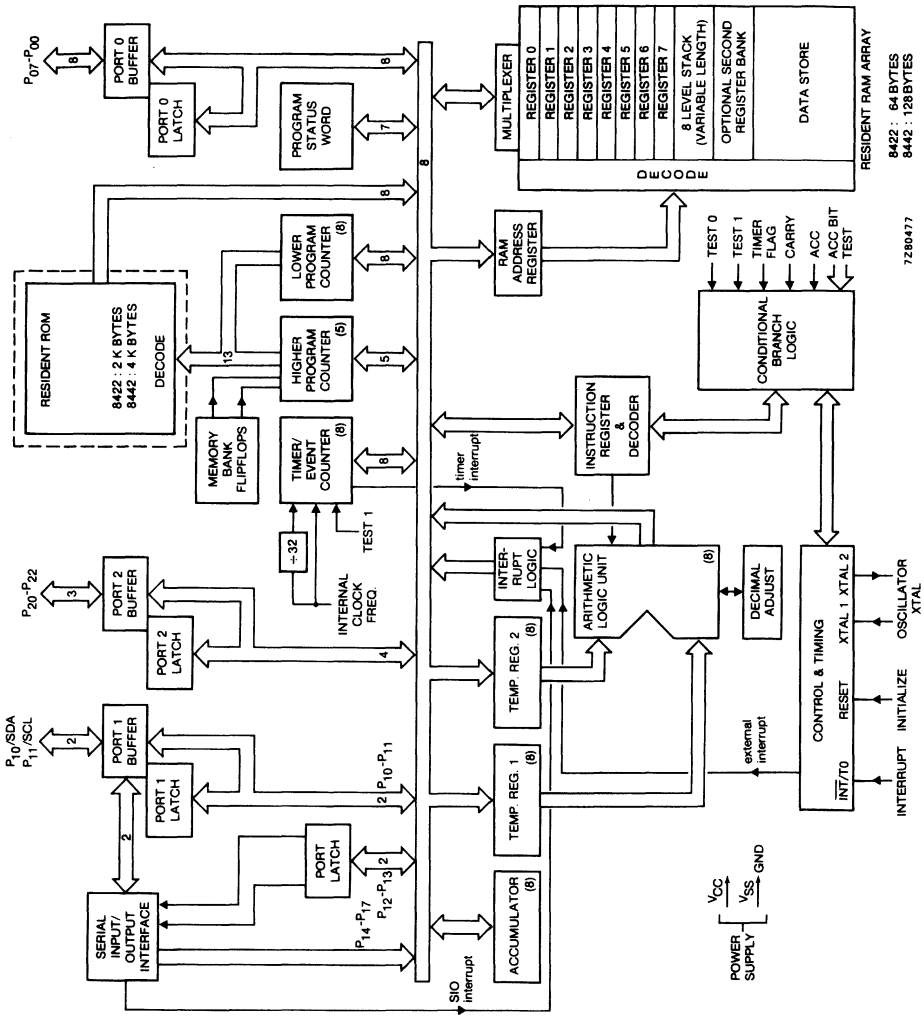


Fig. 1 Block diagram of the MAB8422/8442.

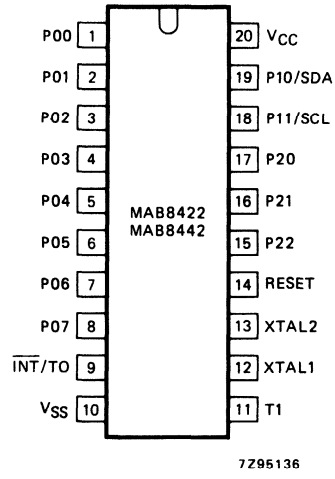


Fig. 2 MAB8422/8442 Pinning diagram.

PINNING

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 or JNT0 instructions.
V_{SS}	10	Ground.
T1	11	Input pin, testable using the JT1 or JNT1 instructions. It can be designated as event counter input using the STRT CNT-instruction. It can also be used to detect zero cross-over 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.

SINGLE-CHIP 8-BIT MICROCONTROLLER

DESCRIPTION

The MAB8052AH is a member of the MAB8051AH family with a higher performance. This single-chip 8-bit microcontroller is manufactured in an advanced 2 μ NMOS process. For this version the following members exist:

- MAB8032AH: ROM-less version of the MAB8052AH
- MAB8052AH: 8 K bytes mask programmable ROM, 256 bytes RAM

Both types are available in 12 MHz versions. In the following, the generic term "MAB8052AH" is used to refer to both family members.

The device provides hardware features, architectural enhancements and new instructions to function as a controller for applications requiring up to 64 K bytes of program memory and/or up to 64 K bytes of data storage.

The MAB8052AH 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; 32 I/O lines; three 16-bit timer/event counters; a six-source, two-priority-level, nested interrupt structure; a serial I/O port for either multi-processor communications, I/O expansion, or full duplex UART; and on-chip oscillator and timing circuits. For systems that require extra capability, the MAB8052AH 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 255 instructions; 44% one-byte, 41% two-byte and 15% 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.
For further detailed information see users manual 'single-chip microcomputer', chapter 8051.

Features

- 8 K x 8 ROM (8052AH only), 256 x 8 RAM
- Four 8-bit ports, 32 I/O lines
- Three 16-bit timer/event counters
- Full-duplex serial port
- External memory expandable to 128 K
- Boolean processing
- 218 bit-addressable locations
- On-chip oscillator
- Six-source interrupt structure with two priority levels
- 58% of instructions executed in 1 μs; multiply and divide in 4 μs
- Enhanced architecture with:
  - non-page-oriented instructions
  - direct addressing
  - four 8-bit register banks
  - stack depth up to 128-bytes
  - multiply, divide, substract and compare
- Upward compatible with 8031AH/8051AH
- Extended temperature range (−40 to +100 °C) in preparation

PACKAGE OUTLINES

MAB8032/52AHP: 40-lead DIL, plastic (SOT-129).
MAB8032/52AHWP: 44-lead plastic leaded chip-carrier (PLCC); SOT-187A.

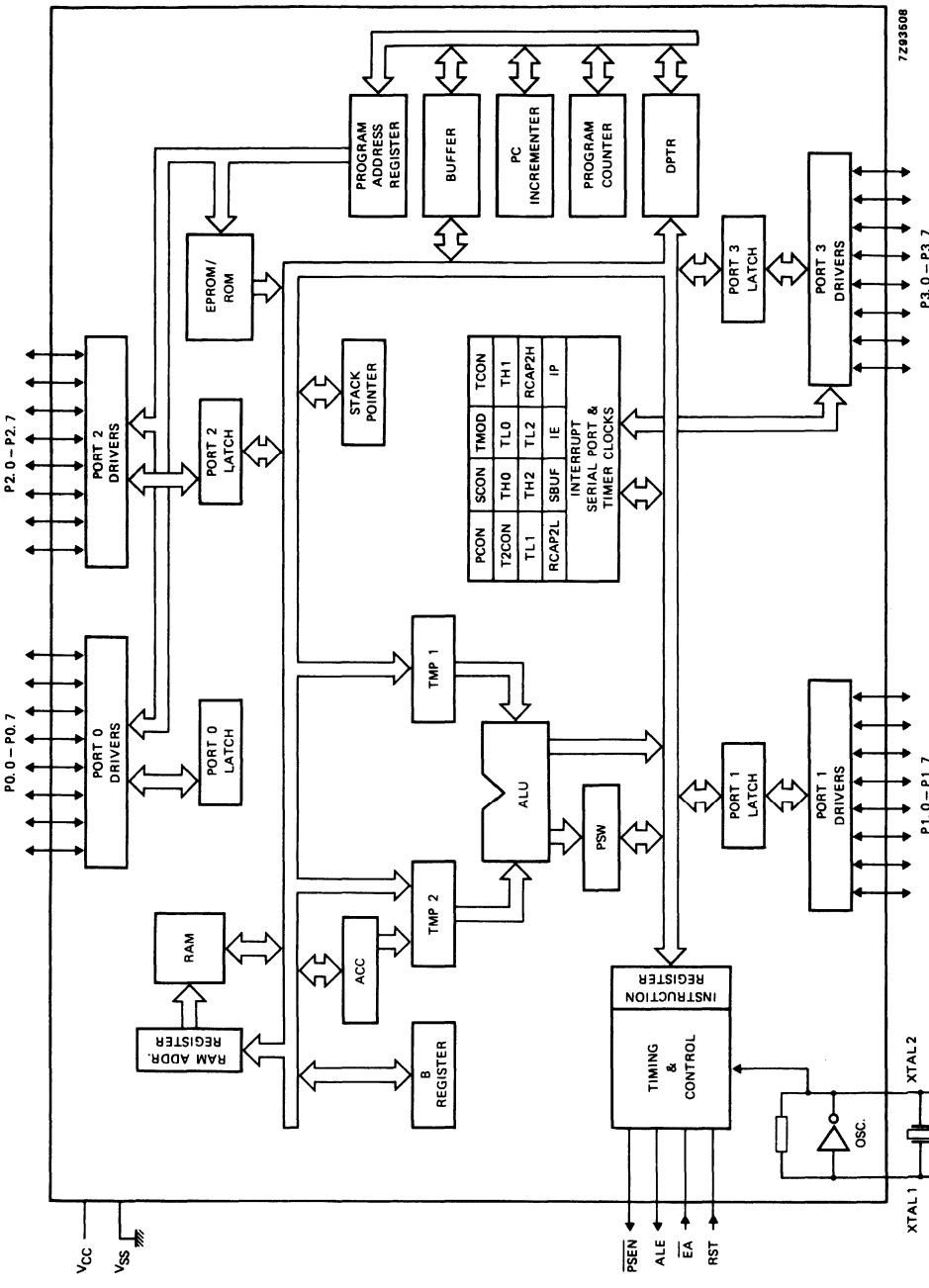


Fig. 1 Block diagram.

DEVELOPMENT DATA

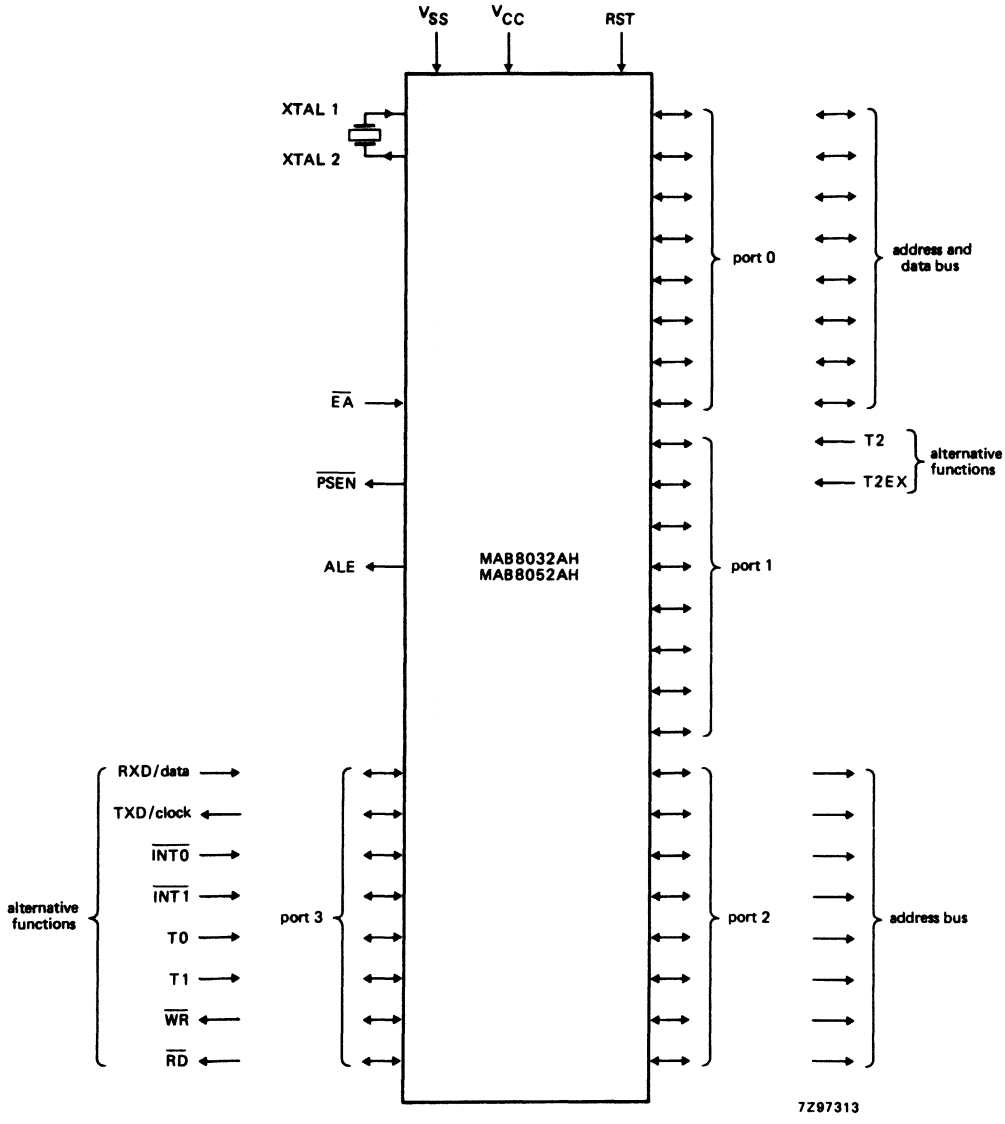


Fig. 2 Functional diagram.

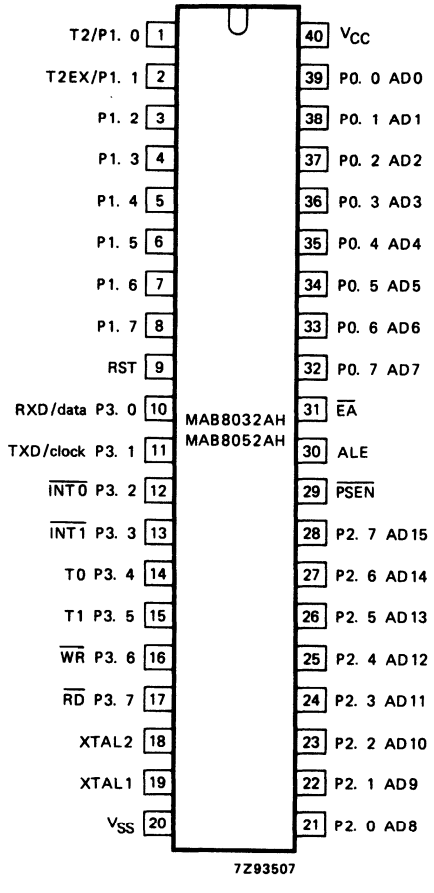


Fig. 3a Pinning diagram: MAB8032/52AHP in 40-lead DIL package; SOT-129.

DEVELOPMENT DATA

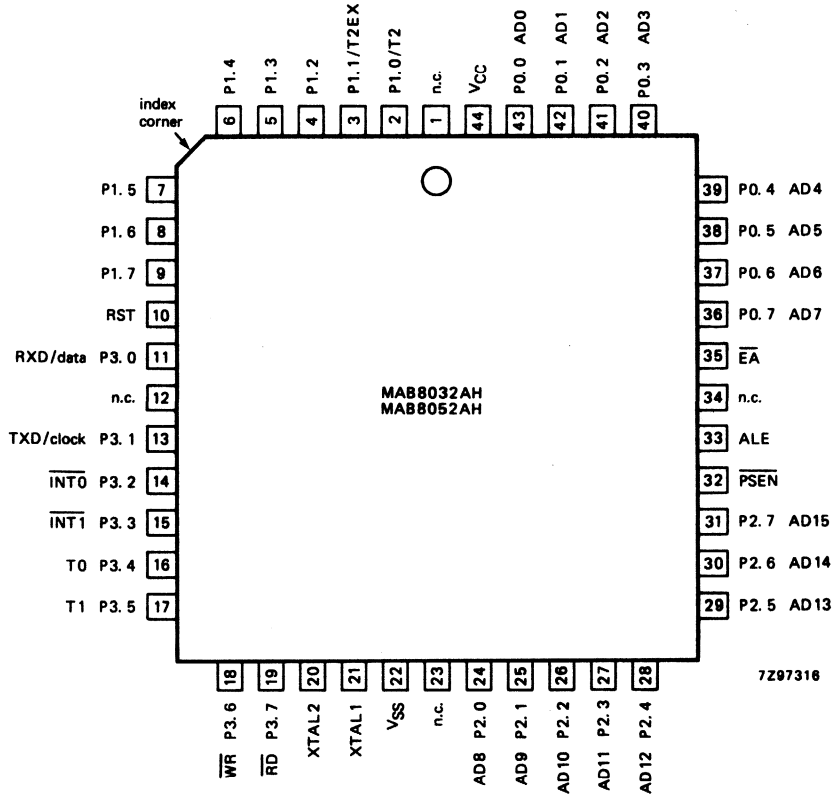


Fig. 3b Pinning diagram: MAB8032/52AHWP in 44-lead PLCC package; SOT-187A.

PINNING (DIL-package)

1-8	P1.0-P1.7	**Port 1:** 8-bit quasi-bidirectional I/O port. It receives the low-order address byte during program verification. Port 1 can sink/source four LS TTL (= 1TTL) inputs. It can drive MOS inputs without external pull-ups. Pins 1 and 2 also supply 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.
9		**RST:** a high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown permits Power-On reset using only a capacitor connected to V_{CC} (see Fig. 18).
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	**$\overline{\text{INT0}}$**: external interrupt 0 or gate control input for timer/event counter 0
P3.3	**$\overline{\text{INT1}}$**: 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	**$\overline{\text{WR}}$**: external data memory write strobe
P3.7	**$\overline{\text{RD}}$** : external data memory read strobe

Operation of an alternative function is determined by the relevant output latch programmed to logic 1. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pull-ups.

18	XTAL 2	**Crystal input 2**: 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 (see figures 15 and 16).
19	XTAL 1	**Crystal input 1**: input to the inverting amplifier that forms the oscillator. Connected to V_{SS} when an external oscillator is used.
20	V_{SS}	**Ground**: circuit earth potential.
21-28	P2.0-P2.7	**Port 2**: 8-bit quais-bidirectional I/O port with internal pull-ups. It emits the high-order address byte when accessing external memory. It also receives the high-order address bits and control signals during program verification. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pull-ups.
29	$\overline{\text{PSEN}}$	**Program Store Enable output**: read strobe to the external Program Memory. It is activated twice each machine cycle during fetches from external Program Memory. When executing out of external Program Memory two activations of $\overline{\text{PSEN}}$ are skipped during each access to external Data Memory. $\overline{\text{PSEN}}$ is not activated (remains HIGH) during fetches from internal Program Memory.

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 addess.
31	$\overline{EA}$	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. $\overline{EA}$ does not float.
32-39	P0.7-P0.0	**Port 0**: 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during these accesses it activates internall pull-ups). It also outputs instruction bytes during program verification. (External pull-ups are required during program verification). Port 0 can sink (and in bus operations can source) eight LS TTL inputs.
40	V_{CC}	**Power Supply**: + 5 V power supply pin during normal operation.

DEVELOPMENT DATA

As the MAB8052AH is based on the MAB8051 the following pages are a description of the additional features.

MEMORIES (see Fig. 4)

ROM: 8 K bytes mask programmable. An external ROM is accessed up to 64 K to accommodate an address higher than 8191.

RAM: 256 bytes (on-chip) plus special function registers detailed in Table 1.

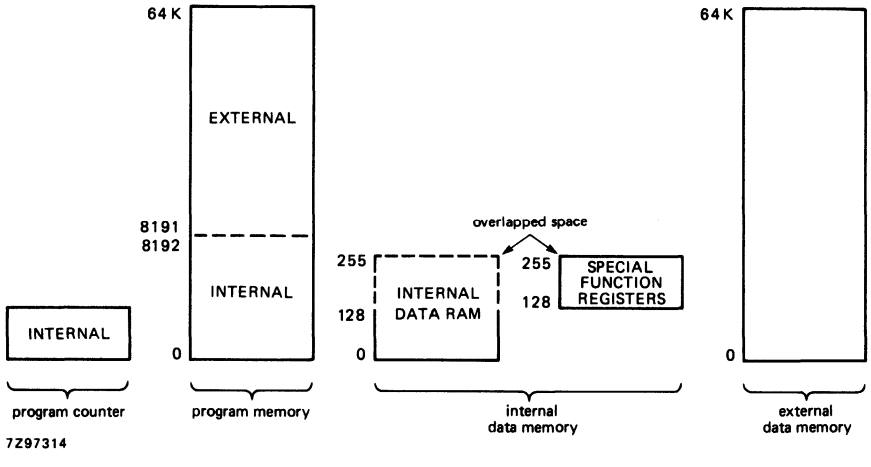


Fig. 4 Memory map.

DEVELOPMENT DATA

SPECIAL FUNCTION REGISTERS

Table 1 Special Function Registers (SFR)

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.

SPECIAL FUNCTION REGISTERS (continued)

T2CON

Special Function Register T2CON is the timer/counter 2 control register. T2CON contains the control and status bits shown in Fig. 5 and described in Table 2.

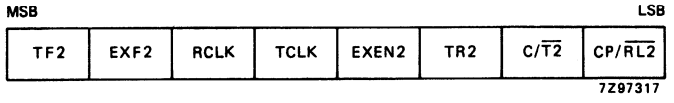


Fig. 5 Timer/counter 2 control register (T2CON).

Table 2 T2CON: Control and status bits

symbol	position	name	operation
TF2	T2CON.7	Timer 2 overflow flag	Set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag	Set when either a capture or reload is caused by a negative transition on T2EX and 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	T2CON.5	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	T2CON.4	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	T2CON.3	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	T2CON.2	Start/stop control for timer 2	A logic 1 starts the timer.
C/$\overline{T2}$	T2CON.1	Time or counter select (timer 2)	0 = internal timer (OSC/12) 1 = external event counter (falling edge triggered)
CP/RL2	T2CON.0	Capture/reload flag	When set capture will occur on negative transitions at T2EX if EXEN2 = 1. When cleared automatic 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 automatic-reload on Timer 2 overflow.

TH2; TL2

Timer/counter 2 (high byte); Timer/counter 2 (low byte) are the register pair for the 16-bit counting of timer/counter 2.

RCAP2H; RCAP2L

Timer/counter 2 capture (high byte); Timer/counter 2 capture (low byte) are the register pair for the "capture mode" of Timer 2. In this mode, a signal at T2EX initiates a copy of TH2 and TL2 into RCAP2H and RCAP2L. Timer 2 also has an automatic-reload mode. RCAP2H and RCAP2L hold the reload value for this mode.

T2CON

Timer 2 is 16-bit and operates the same as Timer 0 and Timer 1. The operating mode is selected by the T2CON register. Table 3 shows the operating modes of Timer 2.

Table 3 Timer 2 operating modes.

RCLK + TCLK	CP/$\overline{RL2}$	TR2	mode
0	0	1	16-bit automatic-reload
0	1	1	16-bit capture
1	X	1	baud rate generator
X	X	0	off

Where

X = don't care.

Capture mode (see Fig. 6)

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which on overflow will set bit TF2 (Timer 2 overflow bit). TF2 can be used to generate an interrupt. If EXEN2 = 1, Timer 2 operates as for EXEN2 = 0 with the added feature that a 1-to-0 transition at the external input T2EX causes the current value in the Timer 2 registers (TL2 and TH2) to be captured into registers RCAP2L and RCAP2H respectively. The 1-to-0 transition of T2EX also causes bit EXF2 in T2CON to be set. EXF2 can be used to generate an interrupt.

Automatic-reload mode (see Fig. 7)

In the automatic-reload mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, when Timer 2 overflows it sets TF2 and 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, Timer 2 operates as for EXEN2 = 0 with the added feature that a 1-to-0 transition at the external input T2EX triggers the 16-bit reload and sets EXF2.

Baud rate generator mode (see Fig. 8)

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1 in T2CON. Thus baud rates for transmit and receive can be simultaneously different. This mode is described in conjunction with the serial port.

DEVELOPMENT DATA

SPECIAL FUNCTION REGISTERS (continued)

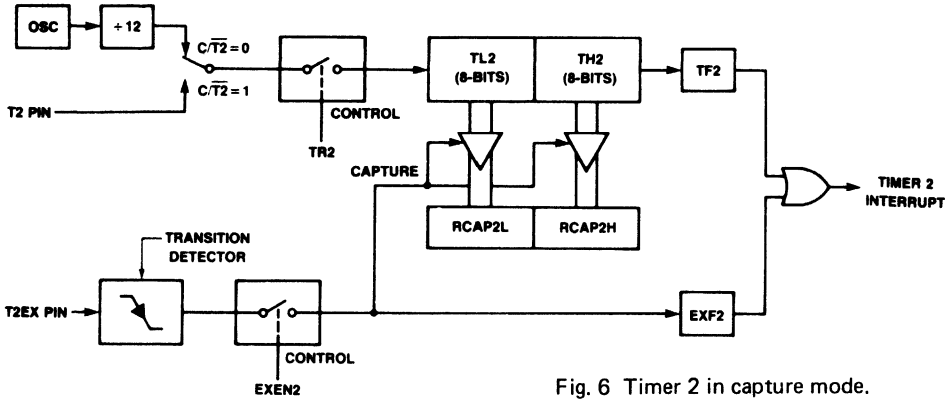


Fig. 6 Timer 2 in capture mode.

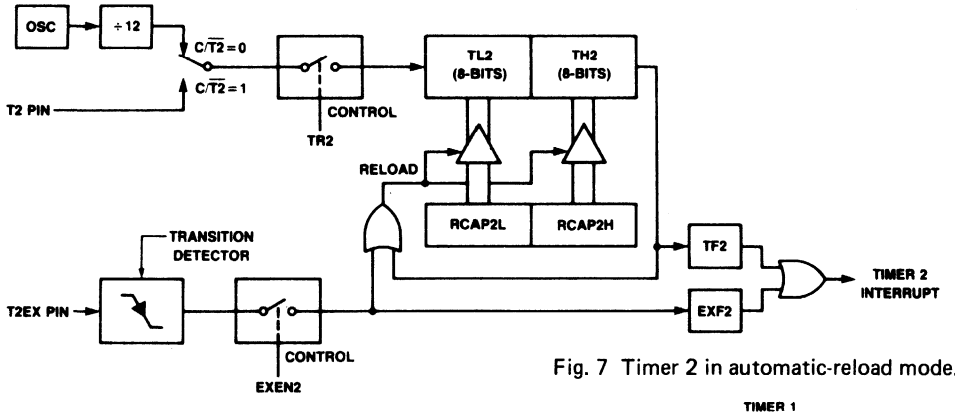


Fig. 7 Timer 2 in automatic-reload mode.

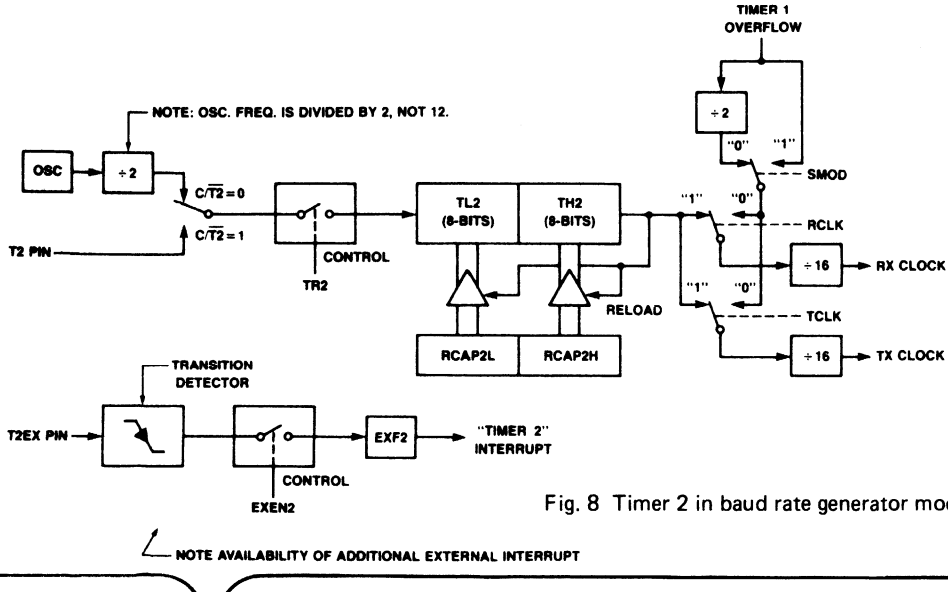


Fig. 8 Timer 2 in baud rate generator mode.

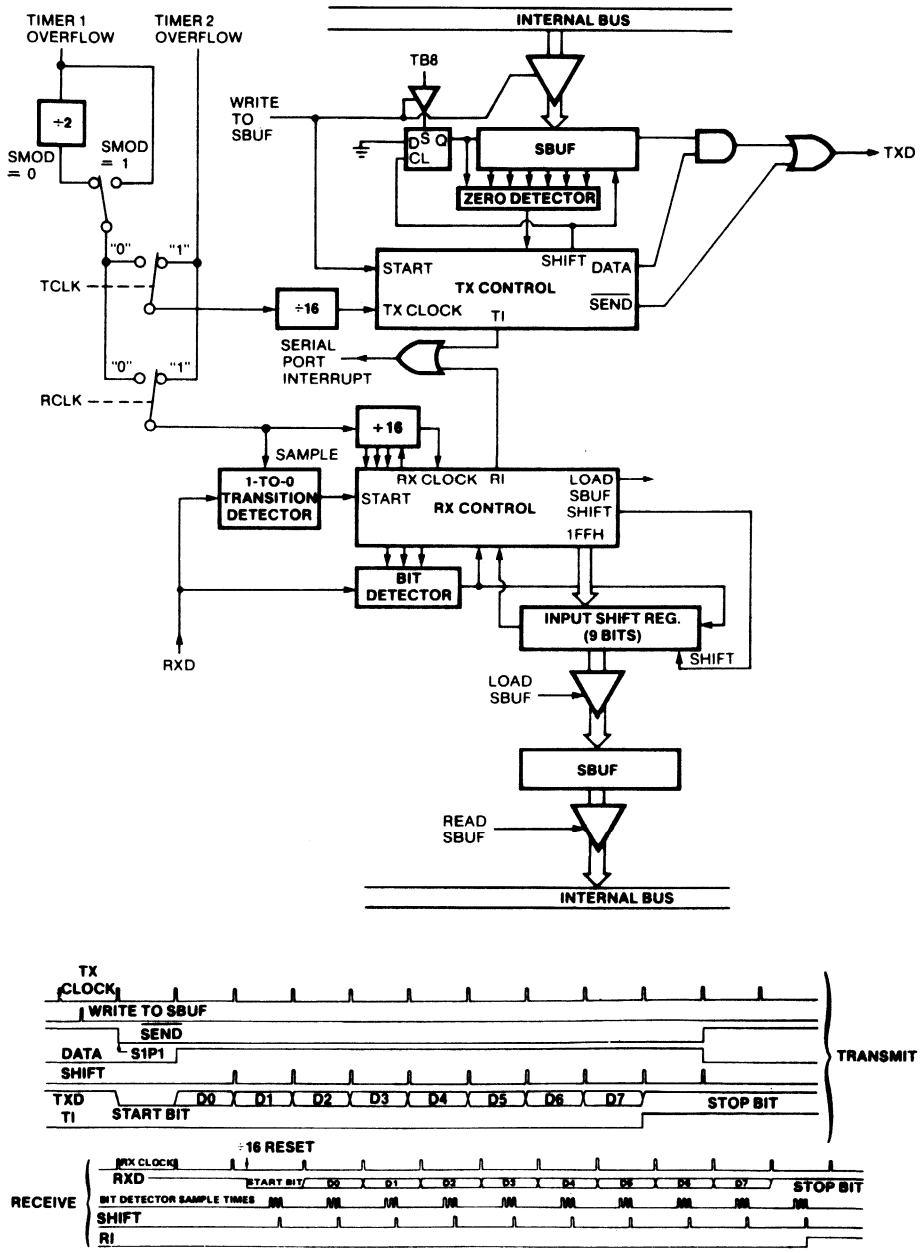


Fig. 9 The serial port in mode 1.

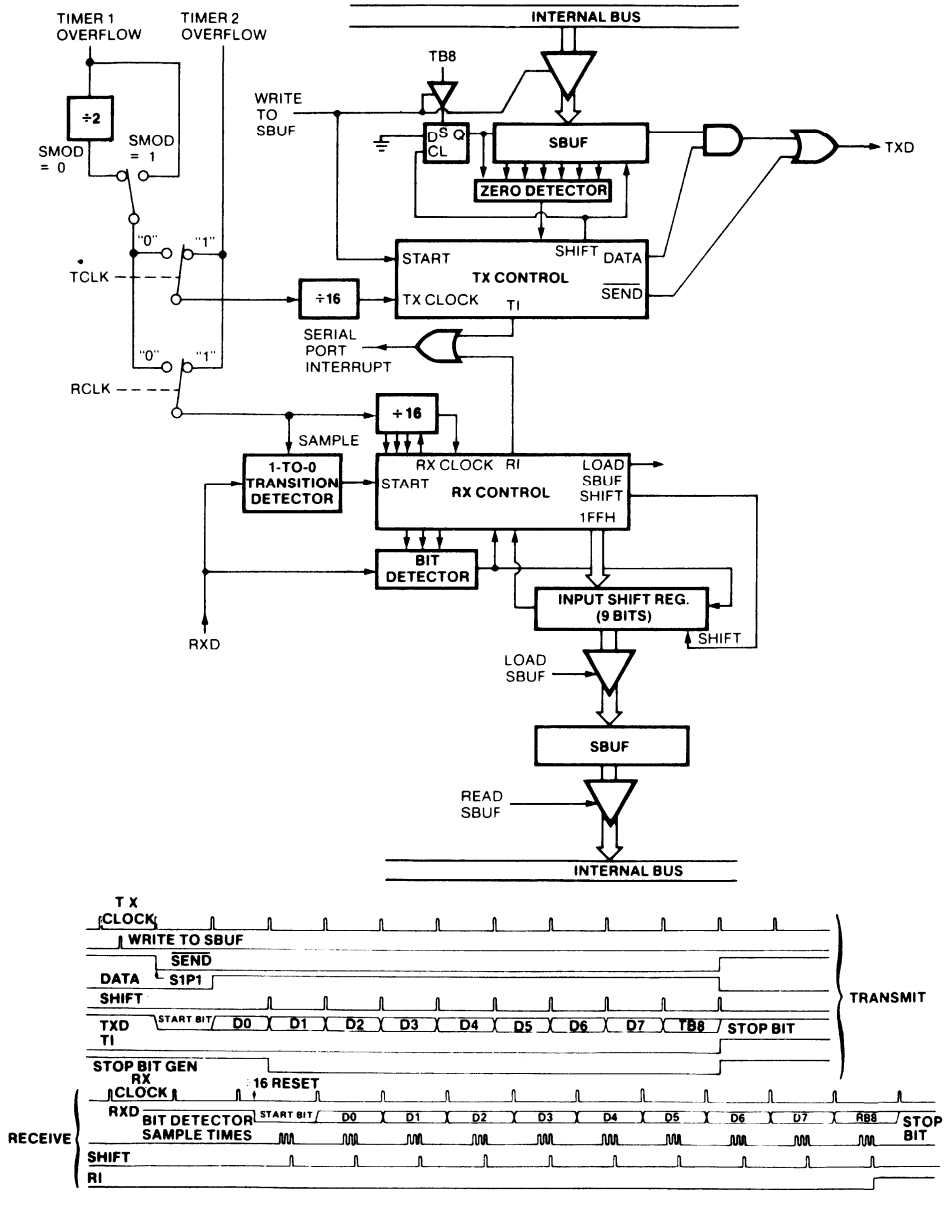


Fig. 10 The serial port in mode 3.

INTERRUPTS

The MAB8052AH has 6 interrupts as shown in Fig. 11.

$\overline{INT0}$; $\overline{INT1}$

The external interrupts $\overline{INT0}$ and $\overline{INT1}$ can be level-activated or transition-activated dependent on bits IT0 and IT1 in register TCON. The flags that generate these interrupts are bits IE0 and IE1.

When a transition-activated external interrupt is generated, the interrupt flag is cleared by the on-chip hardware when the CPU transfers control to the service routine. If the interrupt is level-activated, the request flag is controlled by the external requesting source.

Timer 0; Timer 1

The Timer 0 and Timer 1 interrupts are generated by TF0 and TF1, which are set by an overflow in their respective timer/counter registers. This is not applicable for Timer 0 in mode 3. When a timer interrupt is generated, the interrupt flag is cleared by the on-chip hardware when the CPU transfers control to the service routine.

Timer 2

The interrupt of Timer 2 is generated by the logical OR of TF2 and EXF2. The service routine determines whether it was TF2 or EXF2 that generated the interrupt, and the bits are cleared by software.

Serial port interrupt

The serial port interrupt is generated by the logical OR of R1 and T1 flags. These flags are not cleared by the on-chip hardware when the CPU transfers control to the service routine. The service routine determines whether it was R1 or T1 that generated the interrupt, and the bit is cleared by software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as being set or cleared by hardware. Thus interrupts can be generated, or pending interrupts cancelled, by software.

Each of the interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (see Fig. 12 and Table 4).

Register IE contains a global disable bit (EA) which disables all interrupts simultaneously.

DEVELOPMENT DATA

INTERRUPTS (continued)

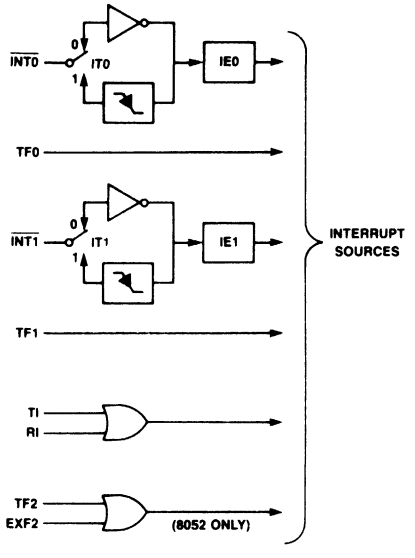


Fig. 11 Interrupt sources.

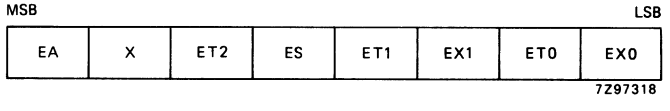


Fig. 12 Interrupt enable register (IE).

Table 4 IE: Control and status bits

symbol	position	function
EA	IE.7	Disables all interrupts. If EA = 0 no interrupt will be acknowledged. If EA = 1 each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	Reserved.
ET2	IE.5	Enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0 the Timer 2 interrupt is disabled.
ES	IE.4	Enables or disables the Serial Port interrupt. If ES = 0 the Serial Port interrupt is disabled.
ET1	IE.3	Enables or disables the Timer 1 overflow interrupt. If ET1 = 0 the Timer 1 interrupt is disabled.
EX1	IE.2	Enables or disables external interrupt 1. If EX1 = 0 the external interrupt is disabled.
ET0	IE.1	Enables or disables the Timer 0 overflow interrupt. If ET0 = 0 the Timer 0 interrupt is disabled.
EX0	IE.0	Enables or disables external interrupt 0. If EX0 = 0 the external interrupt is disabled.

Priority level structure (see Fig. 13 and Table 5)

Each interrupt source can be programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP. A low-priority interrupt can be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt cannot be interrupted by any other interrupt source.

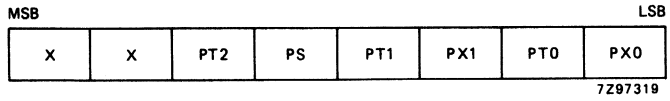


Fig. 13 Interrupt priority register (IP).

Table 5 IP: Control and status bits

symbol	position	function
–	IP.7	Reserved.
–	IP.6	Reserved.
PT2	IP.5	Defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	Defines the Serial Port interrupt priority level. PS1 = 1 programs it to the higher priority level.
PT1	IP.3	Defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PX1	IP.2	Defines the external interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	Defines the external interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

DEVELOPMENT DATA

The interrupt response timing is shown in Fig. 14. The fastest possible response occurs when C2 is the final cycle of an instruction other than "return from interrupt" instruction (RET1), or an access to registers IE or IP.

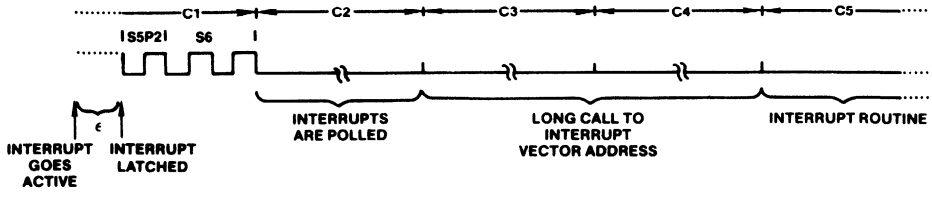


Fig. 14 Interrupt response timing diagram.

INTERRUPTS (continued)

Priority level structure (continued)

If two requests of different priority levels are received simultaneously, the higher priority request is serviced. If requests of the *same* priority level are received simultaneously, an internal polling sequence determines which request is serviced. This second priority structure determined by the polling sequence is detailed in Table 6.

Table 6 Vector addresses by interrupt depending on interrupt source and priority

number	source	priority within level	vector address
1	IE0	(highest)	0003H
2	TF0		000BH
3	IE1		0013H
4	TF1		001BH
5	R1 + T1		0023H
6	TF2 + EXF2	(lowest)	002BH

OSCILLATOR CIRCUITRY

The oscillator circuitry of the MAB8052AH is a single-stage inverting amplifier in a Pierce oscillator configuration. The circuitry has a combination of depletion and enhancement mode MOS FETs to produce the inverting characteristics, and passive components. Either a crystal or ceramic resonator can be used as the feedback element to complete the oscillator circuitry. XTAL1, pin 19, is the high gain amplifier input, and XTAL2, pin 18, is the output (see Fig. 15).

To drive the MAB8052AH externally, XTAL1 should be connected to ground and XTAL2 driven from an external source (see Fig. 16).

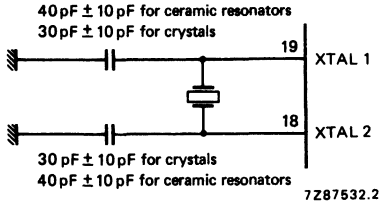


Fig. 15 MAB8052AH oscillator circuit.

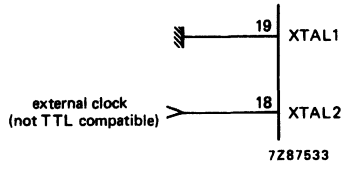


Fig. 16 Driving the MAB8052AH from an external source.

RESET CIRCUITRY

The reset circuitry for the MAB8052AH is connected to the reset pin, RST, as shown in Fig. 17. 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.

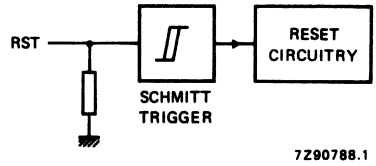


Fig. 17 Reset configuration at RST.

A reset is accomplished 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. 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 shown in Table 1.

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

Power-on reset (see Fig. 18)

When V_{CC} 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_{CC} via a 10 μF capacitor. When the power is switched on, the current drawn by RST is the difference between V_{CC} and the capacitor voltage, and decreases from V_{CC} 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.

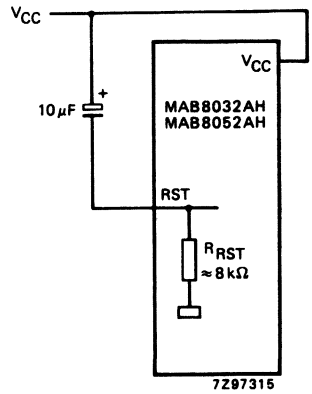


Fig. 18 Power-on reset.

INSTRUCTION SET

The MAB8052AH uses a powerful instruction set to allow expansion of on-chip CPU peripherals and to optimize byte efficiency and execution speed. Reassigned opcodes add new high-power operations and permit new addressing modes to make old operations more orthogonal. The instruction set consists of 49 single-byte, 45 two-byte and 17 three-byte instructions. When using a 12 MHz oscillator, 64 instructions execute in 1 μs and 45 instructions execute in 2 μs. Multiply and divide instructions execute in 4 μs.

Table 7 Instruction set description

mnemonic		description	bytes/ cycles	opcode (hex.)
Arithmetic operation				
ADD	A,Rr	Add register to A	1 1	2*
ADD	A,direct	Add direct byte to A	2 1	25
ADD	A,@Ri	Add indirect RAM to A	1 1	26, 27
ADD	A,#data	Add immediate data to A	2 1	24
ADDC	A,Rr	Add register to A with carry flag	1 1	3*
ADDC	A,direct	Add direct byte to A with carry flag	2 1	35
ADDC	A,@Ri	Add indirect RAM to A with carry flag	1 1	36, 37
ADDC	A,#data	Add immediate data to A with carry flag	2 1	34
SUBB	A,Rr	Subtract register from A with borrow	1 1	9*
SUBB	A,direct	Subtract direct byte from A with borrow	2 1	95
SUBB	A,@Ri	Subtract indirect RAM from A with borrow	1 1	96, 97
SUBB	A,#data	Subtract immediate data from A with borrow	2 1	94
INC	A	Increment A	1 1	04
INC	Rr	Increment register	1 1	0*
INC	direct	Increment direct byte	2 1	05
INC	@Ri	Increment indirect RAM	1 1	06, 07
DEC	A	Decrement A	1 1	14
DEC	Rr	Decrement register	1 1	1*
DEC	direct	Decrement direct byte	2 1	15
DEC	@Ri	Decrement indirect RAM	1 1	16, 17
INC	DPTR	Increment data pointer	1 2	A3
MUL	AB	Multiply A & B	1 4	A4
DIV	AB	Divide A by B	1 4	84
DA	A	Decimal adjust A	1 1	D4

mnemonic		description	bytes/ cycles	opcode (hex.)
Logic operations				
ANL	A,Rr	AND register to A	1 1	5*
ANL	A,direct	AND direct byte to A	2 1	55
ANL	A,@Ri	AND indirect RAM to A	1 1	56, 57
ANL	A,#data	AND immediate data to A	2 1	54
ANL	direct,A	AND A to direct byte	2 1	52
ANL	direct,#data	AND immediate data to direct byte	3 2	53
ORL	A,Rr	OR register to A	1 1	4*
ORL	A,direct	OR direct byte to A	2 1	45
ORL	A,@Ri	OR indirect RAM to A	1 1	46, 47
ORL	A,#data	OR immediate data to A	2 1	44
ORL	direct,A	OR A to direct byte	2 1	42
ORL	direct,#data	OR immediate data to direct byte	3 2	43
XRL	A,Rr	Exclusive-OR register to A	1 1	6*
XRL	A,direct	Exclusive-OR direct byte to A	2 1	65
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1 1	66, 67
XRL	A,#data	Exclusive-OR immediate data to A	2 1	64
XRL	direct, A	Exclusive-OR to direct byte	2 1	62
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3 2	63
CLR	A	Clear A	1 1	E4
CPL	A	Complement A	1 1	F4
RL	A	Rotate A left	1 1	23
RLC	A	Rotate A left through the carry flag	1 1	33
RR	A	Rotate A right	1 1	03
RRC	A	Rotate A right through the carry flag	1 1	13
SWAP	A	Swap nibbles within A	1 1	C4

INSTRUCTION SET (continued)

mnemonic		description	bytes/ cycles	opcode (hex.)
Data transfer				
MOV	A,Rr	Move register to A	1 1	E*
MOV	A,direct	Move direct byte to A	2 1	E5
MOV	A,@Ri	Move indirect RAM to A	1 1	E6, E7
MOV	A,#data	Move immediate data to A	2 1	74
MOV	Rr,A	Move A to register	1 1	F*
MOV	Rr,direct	Move direct byte to register	2 2	A*
MOV	Rr,#data	Move immediate data to register	2 1	7*
MOV	direct,A	Move A to direct byte	2 1	F5
MOV	direct,Rr	Move register to direct byte	2 2	8*
MOV	direct,direct	Move direct byte to direct	3 2	85
MOV	direct,@Ri	Move indirect RAM to direct byte	2 2	86, 87
MOV	direct,#data	Move immediate data to direct byte	3 2	75
MOV	@Ri,A	Move A to indirect RAM	1 1	F6, F7
MOV	@Ri,direct	Move direct byte to indirect RAM	2 2	A6, A7
MOV	@Ri,#data	Move immediate data to indirect RAM	2 1	76, 77
MOV	DPTR,#data16	Load data pointer with a 16-bit constant	3 2	90
MOVC	A,@A+DPTR	Move code byte relative to DPTR to A	1 2	93
MOVC	A,@A+PC	Move code byte relative to PC to A	1 2	83
MOVX	A,@Ri	Move external RAM (8-bit address) to A	1 2	E2, E3
MOVX	A,@DPTR	Move external RAM (16-bit address) to A	1 2	E0
MOVX	@Ri,A	Move A to external RAM (8-bit address)	1 2	F2, F3
MOVX	@DPTR,A	Move A to external RAM (16-bit address)	1 2	F0
PUSH	direct	Push direct byte onto stack	2 2	C0
POP	direct	Pop direct byte from stack	2 2	D0
XCH	A,Rr	Exchange register with A	1 1	C*
XCH	A,direct	Exchange direct byte with A	2 1	C5
XCH	A,@Ri	Exchange indirect RAM with A	1 1	C6, C7
XCHD	A,@Ri	Exchange LOW-order digit indirect RAM with A	1 1	D6, D7

mnemonic		description	bytes/ cycles	opcode (hex.)
Boolean variable manipulation				
CLR	C	Clear carry flag	1 1	C3
CLR	bit	Clear direct bit	2 1	C2
SETB	C	Set carry flag	1 1	D3
SETB	bit	Set direct bit	2 1	D2
CPL	C	Complement carry flag	1 1	B3
CPL	bit	Complement direct bit	2 1	B2
ANL	C,bit	AND direct bit to carry flag	2 2	82
ANL	C,/bit	AND complement of direct bit to carry flag	2 2	B0
ORL	C,bit	OR direct bit to carry flag	2 2	72
ORL	C,/bit	OR complement of direct bit to carry flag	2 2	A0
MOV	C,bit	Move direct bit to carry flag	2 1	A2
MOV	bit,C	Move carry flag to direct bit	2 2	92
Program and machine control				
ACALL	addr11	Absolute subroutine call	2 2	●1addr
LCALL	addr16	Long subroutine call	3 2	12
RET		Return from subroutine	1 2	22
RET1		Return from interrupt	1 2	32
AJMP	addr11	Absolute jump	2 2	▲1addr
LJMP	addr16	Long jump	3 2	02
SJMP	rel	Short jump (relative address)	2 2	80
JMP	@A+DPTR	Jump indirect relative to the DPTR	1 2	73
JZ	rel	Jump if A is zero	2 2	60
JNZ	rel	Jump if A is not zero	2 2	70
JC	rel	Jump if carry flag is set	2 2	40
JNC	rel	Jump if no carry flag	2 2	50
JB	bit,rel	Jump if direct bit is set	3 2	20
JNB	bit,rel	Jump if direct bit is not set	3 2	30
JBC	bit,rel	Jump if direct bit is set and clear bit	3 2	10
CJNE	A,direct,rel	Compare direct to A and jump if not equal	3 2	B5
CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3 2	B4
CJNE	Rr,#data,rel	Compare immed. to reg. and jump if not equal	3 2	B*
CJNE	@Ri,#data,rel	Compare immed. to ind. and jump if not equal	3 2	B6, B7
DJNZ	Rr,rel	Decrement register and jump if not zero	2 2	D*
DJNZ	direct,rel	Decrement direct and jump if not zero	3 2	D5
NOP		No operation	1 1	00

DEVELOPMENT DATA

Notes to Table 7

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 instruction.
#data16	16-bit constant included as bytes 2 and 3 of 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 64 K-byte program memory address space.
addr11	11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2 K-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.

Hexadecimal opcode cross-reference to Table 8

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

●: 11, 31, 51, 71, 91, B1, D1, F1.

▲: 01, 21, 41, 61, 81, A1, C1, E1.

Table 8 Instruction map

first hexadecimal chacter 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	AJMP page 0	LJMP addr16	RR A	INC A	INC dir	INC @Ri 0	1	INC Rr 0	1	2	3	4	5	6	7
1	JBC bit, addr8	ACALL page 0	LCALL addr16	RRC A	DEC A	DEC dir	DEC @Ri 0	1	DEC Rr 0	1	2	3	4	5	6	7
2	JB bit, addr8	AJMP page 1	RET	RL A	ADD A,#data	ADD A,dir	ADD A,@Ri 0	1	ADD A,Rr 0	1	2	3	4	5	6	7
3	JNB bit, addr8	ACALL page 1	RET1	RLC A	ADDC A,#data	ADDC A,dir	ADDC A,@Ri 0	1	ADDC A,Rr 0	1	2	3	4	5	6	7
4	JC addr8	AJMP page 2	ORL dir,A	ORL dir,#data	ORL A,#data	ORL A,dir	ORL A,@Ri 0	1	ORL A,Rr 0	1	2	3	4	5	6	7
5	JNC addr8	ACALL page 2	ANL dir,A	ANL dir,#data	ANL A,#data	ANL A,dir	ANL A,@Ri 0	1	ANL A,Rr 0	1	2	3	4	5	6	7
6	JZ addr8	AJMP page 3	XRL dir,A	XRL dir,#data	XRL A,#data	XRL A,dir	XRL A,@Ri 0	1	XRL A,Rr 0	1	2	3	4	5	6	7
7	JNZ addr8	ACALL page 3	ORL C,bit	JMP @A+DPTR	MOV A,#data	MOV dir,#data	MOV @Ri,#data 0	1	MOV Rr,#data 0	1	2	3	4	5	6	7
8	SJMP addr8	AJMP page 4	ANL C,bit	MOVC A,@A+PC	DIV AB	MOV dir,dir	MOV dir,@Ri 0	1	MOV dir,Rr 0	1	2	3	4	5	6	7
9	MOV DPTR, #data 16	ACALL page 4	MOV bit,C	MOVC A,@A+DPTR	SUBB A,#data	SUBB A,dir	SUBB A,@Ri 0	1	SUBB A,Rr 0	1	2	3	4	5	6	7
A	ORL C,/bit	AJMP page 5	MOV C,bit	INC DPTR	MUL AB		MOV @Ri,dir 0	1	MOV Rr,dir 0	1	2	3	4	5	6	7
B	ANL C,/bit	ACALL page 5	CPL bit	CPL C	CJNE A, #data, addr8	CJNE A,dir,addr8	CJNE @Ri,#data, addr8 0	1	CJNE Rr,#data, addr8 0	1	2	3	4	5	6	7
C	PUSH dir	AJMP page 6	CLR bit	CLR C	SWAP A	XCH A,dir	XCH A,@Ri 0	1	XCH A,Rr 0	1	2	3	4	5	6	7
D	POP dir	ACALL page 6	SETB bit	SETB C	DA A	DJNZ dir, addr8	XCHD A,@Ri 0	1	DJNZ Rr, addr8 0	1	2	3	4	5	6	7
E	MOVX A,@DPTR	AJMP page 7	MOVX A,@Ri 0	1	CLR A	MOV A,dir	MOV A,@Ri 0	1	MOV A,Rr 0	1	2	3	4	5	6	7
F	MOVX @DPTR,A	ACALL page 7	MOVX @Ri,A 0	1	CPL A	MOV dir,A	MOV @Ri,A 0	1	MOV Rr,A 0	1	2	3	4	5	6	7

RATINGS

Limiting values in accordance with the Absolute Maximum System (IEC 134)

Input voltage on any pin with respect to ground (V_{SS})	V_I	−0,5 to + 7	V
Total power dissipation	P_{tot}	max. 2	W
Input, output current	$\pm I_I$, I_O	max. 10	mA
Storage temperature range	T_{stg}	−65 to + 150	°C
Operating ambient temperature range	T_{amb}	0 to + 70	°C

D.C. CHARACTERISTICS

V_{CC} = 5 V (± 10%); V_{SS} = 0 V; T_{amb} = 0 to + 70 °C; all voltages with respect to V_{SS} unless otherwise specified

parameter	symbol	min.	max.	unit	conditions
Supply current	I_{CC}	–	175	mA	all outputs disconnected; $\overline{EA} = V_{CC}$
Inputs					
Input voltage LOW	V_{IL}	−0,5	0,8	V	
Input voltage HIGH all inputs except RST and XTAL 2	V_{IH}	2	V_{CC} + 0,5	V	
Input voltage HIGH to RST and XTAL 2	V_{IH1}	2,5	V_{CC} + 0,5 V	V	XTAL 1 to V_{SS}
Outputs					
Output voltage LOW (Ports 1, 2, 3) (note 1)	V_{OL}	–	0,45	V	I_{OL} = 1,6 mA
Output voltage LOW (Port 0, ALE, $\overline{PSEN}$) (note 1)	V_{OL1}	–	0,45	V	I_{OL1} = 3,2 mA
Output voltage HIGH (Ports 1, 2, 3)	V_{OH}	2,4	–	V	I_{OH} = −80 μA
Output voltage HIGH (Port 0, ALE, $\overline{PSEN}$)	V_{OH1}	2,4	–	V	I_{OH1} = −400 μA
Input leakage current (Port 0, $\overline{EA}$)	$\pm I_{LI}$	–	10	μA	0,45 V < V_I < V_{CC}
Input current HIGH (RST)	I_{IH1}	–	500	μA	$V_I = V_{CC}$−1,5 V
current logic 0 (Ports 1, 2, 3)	I_{IL}	–	−800	μA	V_{IL} = 0,45 V;
Input current logic 0 (XTAL 2)	I_{IL2}	–	−3,2	mA	V_{IL} = 0,45 V; XTAL 1 to V_{SS}
Capacitance of I/O buffer	$C_{I/O}$	–	10	pF	f_c = 1 MHz; T_{amb} = 25 °C

Note 1

V_{OL} is degraded when the MAB8052AH rapidly discharges external capacitance.
This a.c. noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the MAB8052AH as possible.

datum	emitting ports	time slot interval	degraded I/O lines	V_{OL} (max.)
address	P2, P0	TS3, TS9	P1, P3	0,8 V
write data	P0	TS6	P1, P3, ALE	0,8 V

A.C. CHARACTERISTICS

V_{CC} = 5 V ± 10%; V_{SS} = 0 V; T_{amb} = 0 to + 70 °C; C_L = 100 pF (Port 0, ALE and $\overline{PSEN}$); C_L = 80 pF all other outputs unless otherwise specified (see waveforms Figs 21, 22 and 23).

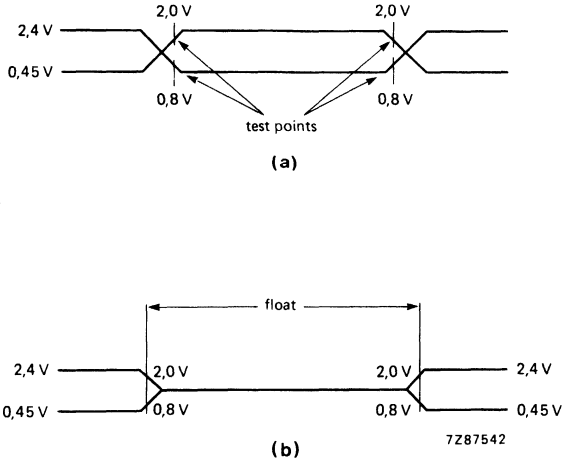
parameter	symbol	12 MHz		variable clock (note 1)		unit
		min.	max.	min.	max.	
Program memory						
ALE pulse duration	t_{LL}	127	–	$2t_{CK}-40$	–	ns
Address set-up time to ALE	t_{AL}	43	–	$t_{CK}-40$	–	ns
Address hold time after ALE	t_{LA}	48	–	$t_{CK}-35$	–	ns
Time from ALE to valid instruction input	t_{LIV}	–	233	–	$4t_{CK}-100$	ns
Time from ALE to control pulse $\overline{PSEN}$	t_{LC}	58	–	$t_{CK}-25$	–	ns
Control pulse duration $\overline{PSEN}$	t_{CC}	215	–	$3t_{CK}-35$	–	ns
Time from $\overline{PSEN}$ to valid instruction input	t_{CIV}	–	125	–	$3t_{CK}-125$	ns
Input instruction hold time after $\overline{PSEN}$	t_{CI}	0	–	0	–	ns
Input instruction float delay after $\overline{PSEN}$ (note 2)	t_{CIF}	–	63	–	$t_{CK}-20$	ns
Address valid after $\overline{PSEN}$ (note 2)	t_{AC}	75	–	$t_{CK}-8$	–	ns
Address to valid instruction input	t_{AIV}	–	302	–	$5t_{CK}-115$	ns
Address float time to $\overline{PSEN}$	t_{AFC}	−12	–	−12	–	ns

parameter	symbol	12 MHz		variable clock (note 1)		unit
		min.	max.	min.	max.	
External data memory						
$\overline{RD}$ pulse duration	t_{RR}	400	–	$6t_{CK}-100$	–	ns
$\overline{WR}$ pulse duration	t_{WW}	400	–	$6t_{CK}-100$	–	ns
Address hold time after ALE	t_{LA}	48	–	$t_{CK}-35$	–	ns
RD to valid data input	t_{RD}	–	250	–	$5t_{CK}-165$	ns
Data hold time after $\overline{RD}$	t_{DR}	0	–	0	–	ns
Data float delay after $\overline{RD}$	t_{DFR}	–	97	–	$2t_{CK}-70$	ns
Time from ALE to valid data input	t_{LD}	–	517	–	$8t_{CK}-150$	ns
Address to valid data input	t_{AD}	–	585	–	$9t_{CK}-165$	ns
Time from ALE to $\overline{RD}$ or $\overline{WR}$	t_{LW}	200	300	$3t_{CK}-50$	$3t_{CK}+50$	ns
Time from address to $\overline{RD}$ or $\overline{WR}$	t_{AW}	203	–	$4t_{CK}-130$	–	ns
Time from $\overline{RD}$ or $\overline{WR}$ HIGH to ALE HIGH	t_{WHLH}	43	123	$t_{CK}-40$	$t_{CK}+40$	ns
Data valid to $\overline{WR}$ transition	t_{DWX}	23	–	$t_{CK}-60$	–	ns
Data set-up time before $\overline{WR}$	t_{DW}	433	–	$7t_{CK}-150$	–	ns
Data hold time after $\overline{WR}$	t_{WD}	33	–	$t_{CK}-50$	0	ns
Address float delay after $\overline{RD}$	t_{AFR}	–	+ 12	–	+ 12	ns

Notes to the a.c. characteristics

1. $1/t_{CK}$ = 3,5 to 12 MHz (see Fig. 20 and Table 9).
2. Interfacing the MAB8052AH to devices with float times up to 75 ns is permitted. This limited bus contention will cause damage to port 0 drivers.

DEVELOPMENT DATA



A.C. testing inputs are driven at 2,4 V for a logic 1 and 0,45 V for a logic 0. Timing measurements are taken at 2,0 V for a logic 1 and 0,8 V for logic 0. The float state is defined as the point at which a Port 0 pin sinks 3,2 mA or sources 400 μA at the voltage test levels.

Fig. 19 A.C. testing input, output waveform (a) and float waveform (b).

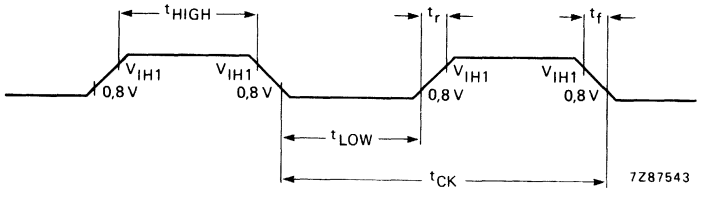


Fig. 20 External clock drive XTAL 2 (see Table 9).

Table 9 External clock drive XTAL 2 (see Fig. 20)

parameter	symbol	variable clock (f = 3,5 to 12 MHz)		unit
		min.	max.	
oscillator clock period	t_{CK}	83,3	286	ns
HIGH time	t_{HIGH}	20	$t_{CK}-t_{LOW}$	ns
LOW time	t_{LOW}	20	$t_{CK}-t_{HIGH}$	ns
rise time	t_r	–	20	ns
fall time	t_f	–	20	ns

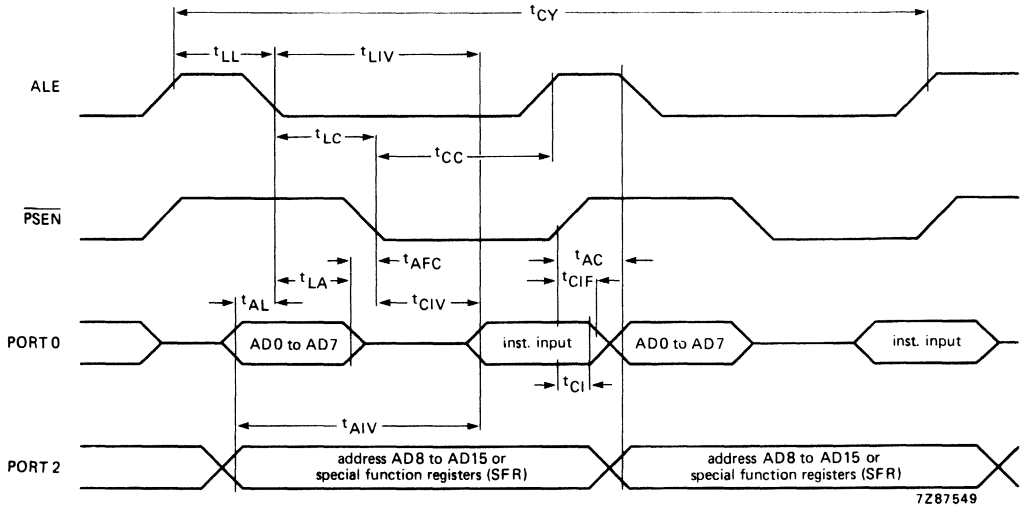


Fig. 21 Read from program memory.

DEVELOPMENT DATA

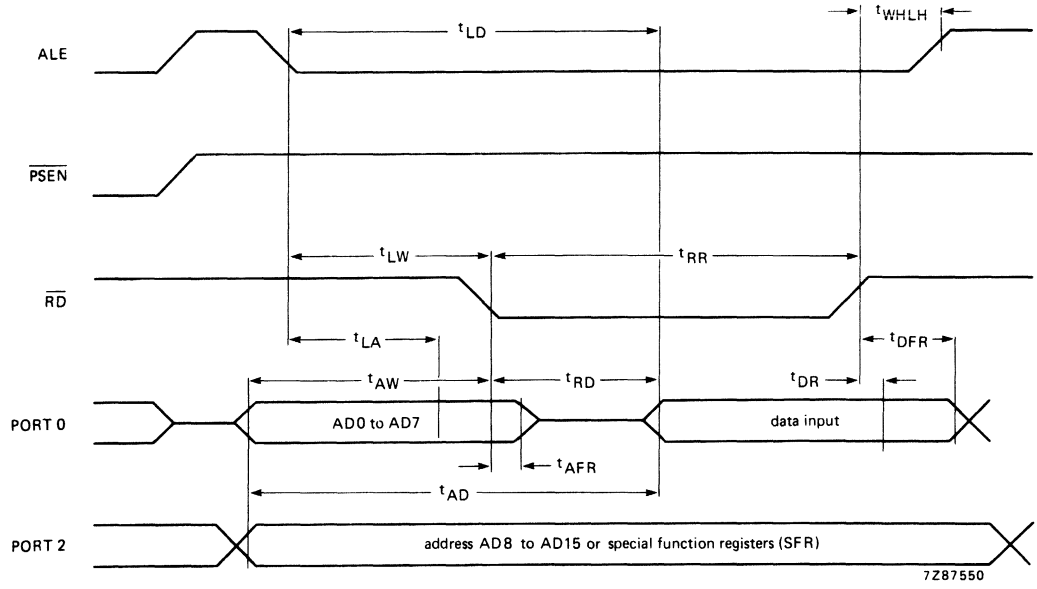


Fig. 22 Read from data memory.

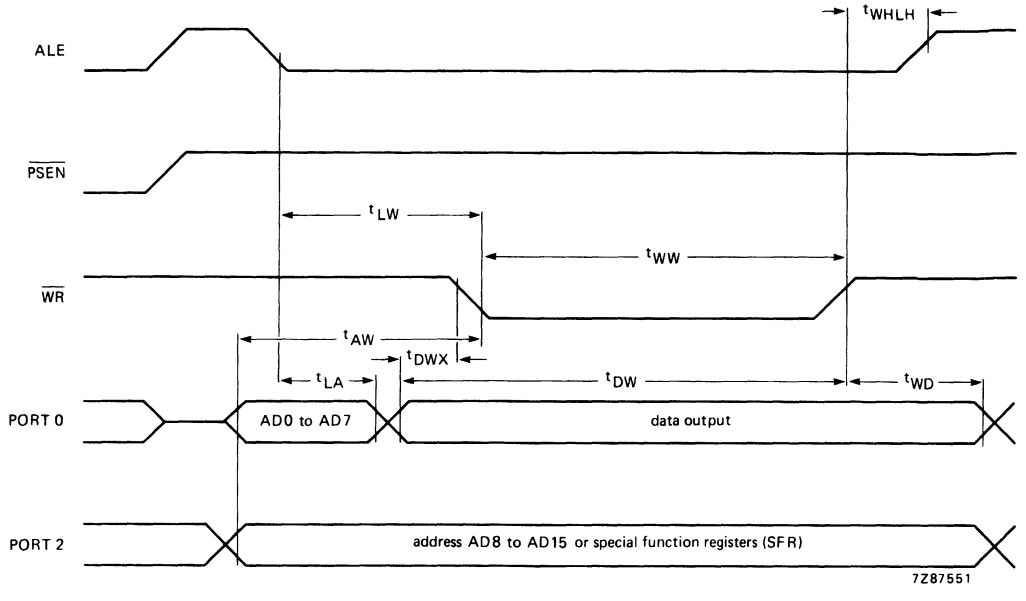


Fig. 23 Write to data memory.

DEVELOPMENT DATA

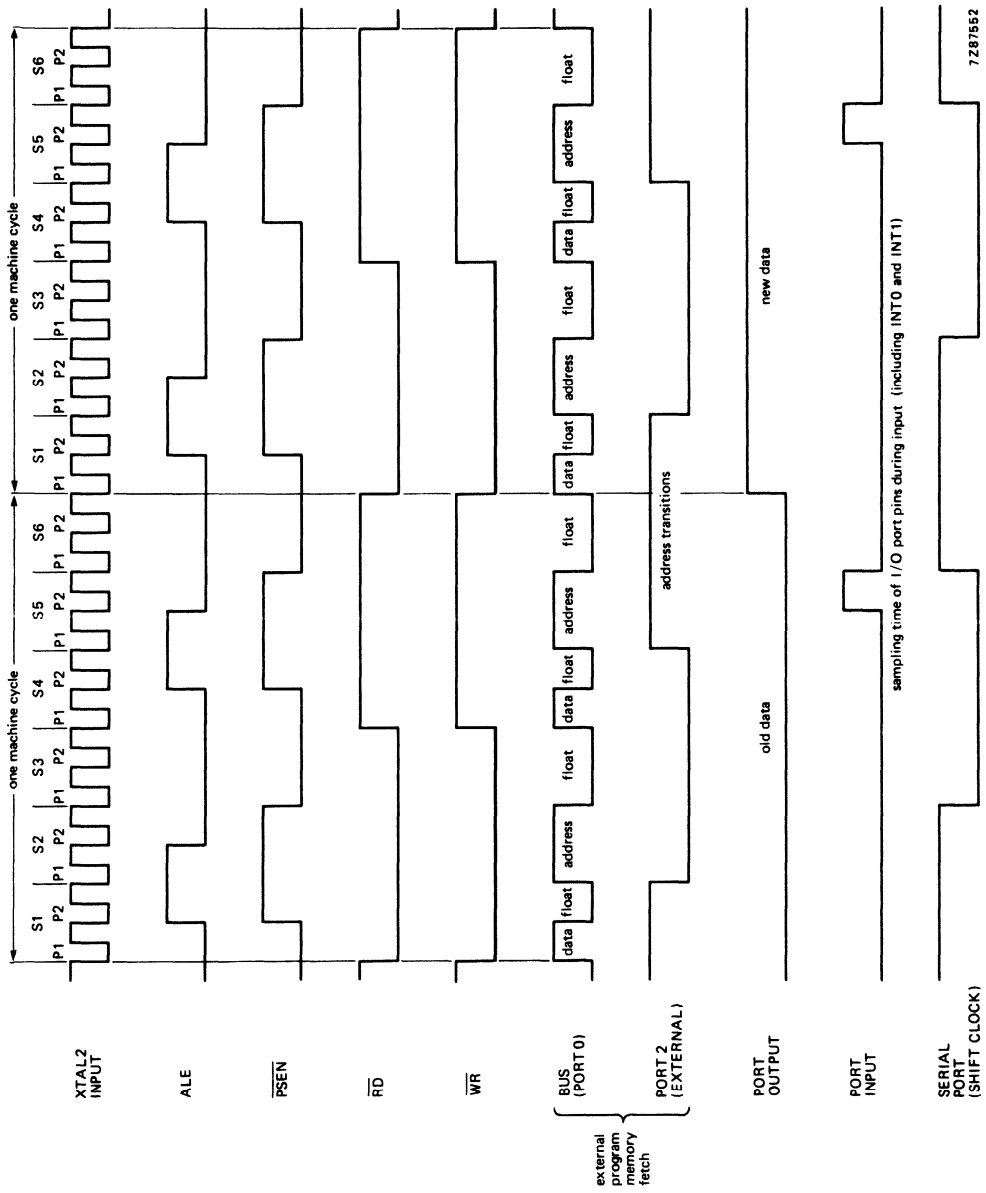


Fig. 24 Instruction cycle timing.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

DESCRIPTION

The MAB8051AH family of single-chip 8-bit microcontrollers is manufactured in an advanced 2 μ NMOS process. The family consists of the following members:

- MAB8031AH: ROM-less version of the MAB8051AH
- MAB8051AH: 4 K bytes mask-programmable ROM, 128 bytes RAM

Both types are available in 8, 10 and 12 MHz versions and 15 MHz for the MAB8031AH. In the following, the generic term "MAB8051AH" is used to refer to both family members.

The device provides hardware features, architectural enhancements and new instructions to function as a controller for applications requiring up to 64 K bytes of program memory and/or up to 64 K bytes of data storage.

The MAB8051AH contains a non-volatile 4 K x 8 read-only program memory (not ROM-less version); a volatile 128 x 8 read/write data memory; 32 I/O lines; two 16-bit timer/event counters; a five-source, two-priority-level, nested interrupt structure; a serial I/O power for either multi-processor communications, I/O expansion, or full duplex UART; and on-chip oscillator and timing circuits. For systems that require extra capability, the MAB8051AH 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 255 instructions; 44% one-byte, 41% two-byte and 15% 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. Multiply, divide, subtract and compare are among the many instructions added to the standard MAB8048H instruction set.
For further detailed information see Users Manual 'Single-chip microcomputer'.

Features

- 4 K x 8 ROM (8051AH only), 128 x 8 RAM
- Four 8-bit ports, 31 I/O lines
- Two 16-bit timer/event counters
- Full duplex serial port
- External memory expandable to 128 K
- Boolean processing
- 218 bit-addressable locations
- On-chip oscillator
- Five-source interrupt structure with two priority levels
- 58% of instructions executed in 1 μs; multiply and divide in 4 μs (at 12 MHz clock)
- Enhanced architecture with:
  - non-page-oreinted instructions
  - direct addressing
  - four 8-bit register banks
  - stack depth up to 128-bytes
  - multiply, divide, subtract and compare
- Available with extended temperature range: −40 to + 85 °C (MAF8031/51AH)
- Available with automotive temperature range: −40 to + 100 °C (MAF80A31/51AH)

PACKAGE OUTLINES

MAB8031/51AHP; MAF8031/51AHP; MAF80A31/51AHP: 40-lead DIL; plastic (SOT-129).
MAB8031/51AHWP; MAF8031/51AHWP; MAF80A31/51AHWP: 44-lead, plastic leaded-chip-carrier (PLCC); SOT-187A.

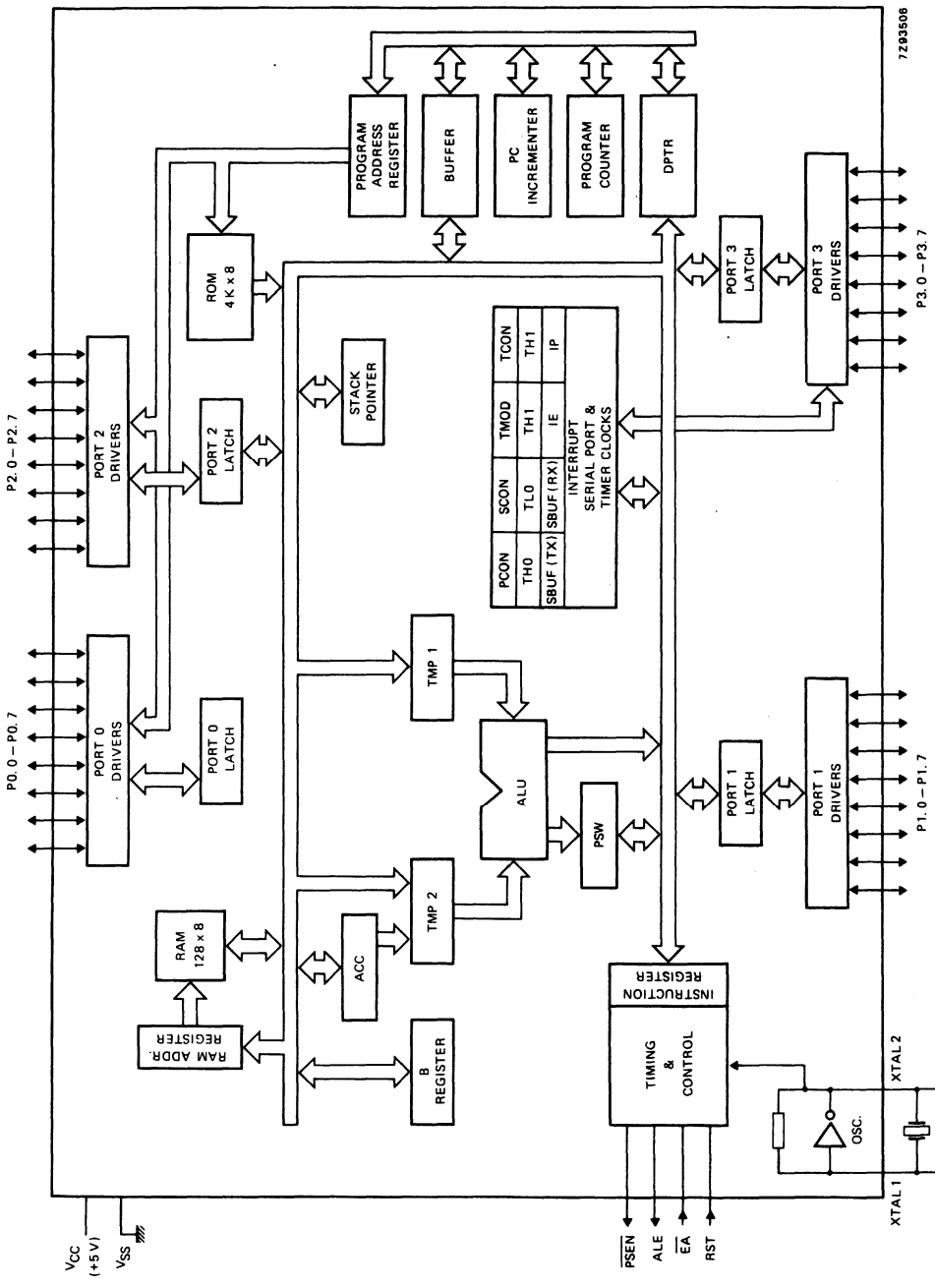


Fig. 1 Block diagram.

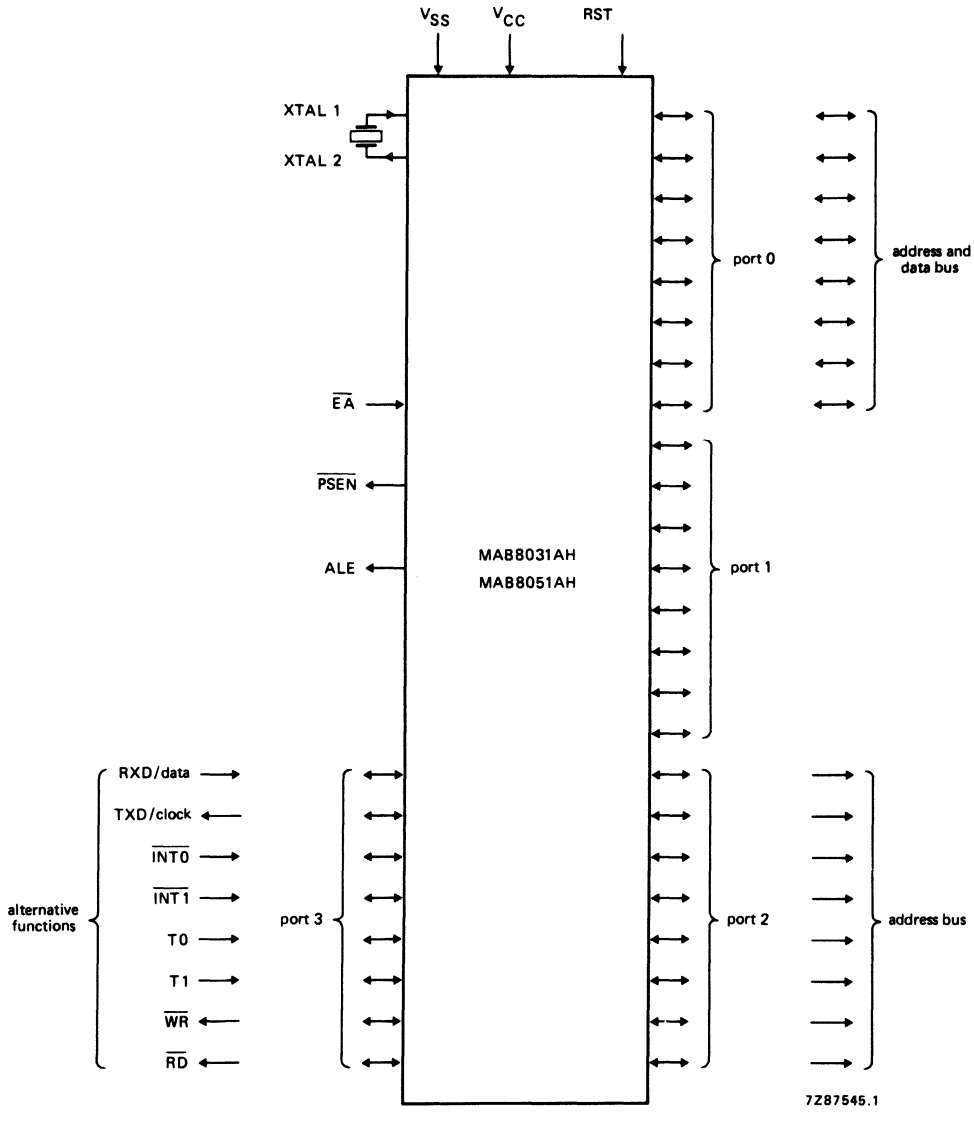


Fig. 2 Functional diagram.

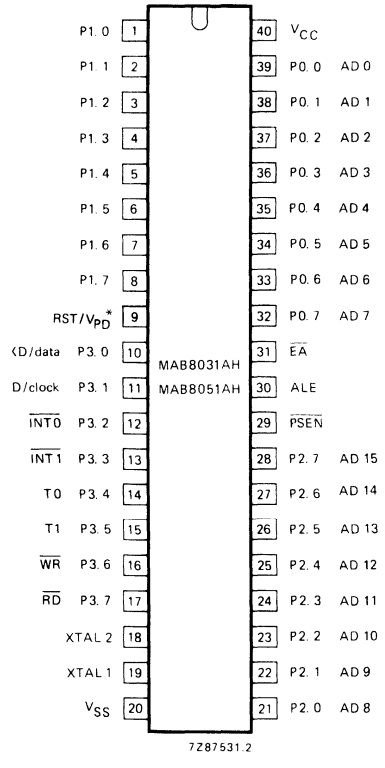


Fig. 3a Pinning diagram for MAB8031/51AHP; MAF8031/51AHP; MAF80A31/51AHP.

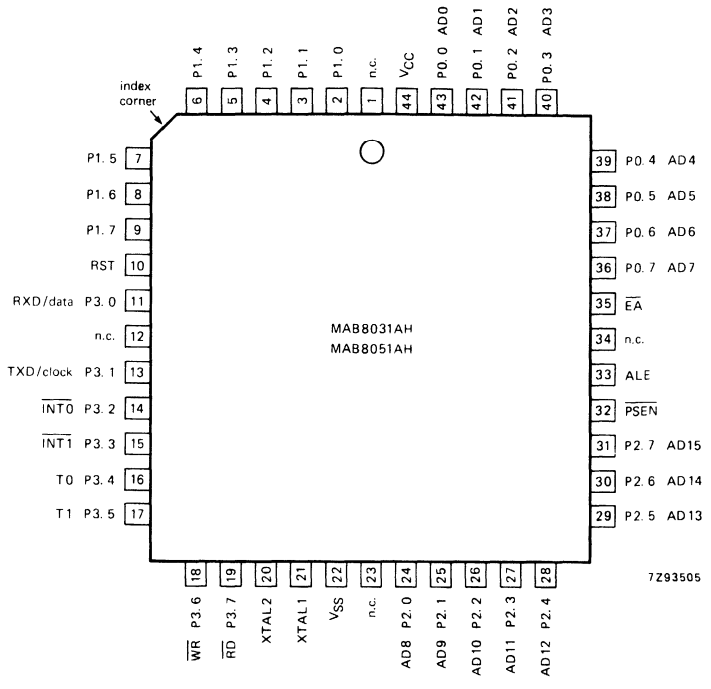


Fig. 3b Pinning diagram for MAB8031/51AHWP; MAF8031/51AHWP; MAF80A31/51AHWP.

* V_{PD} option available on request.

PINNING (DIL package)

1-8	P1.0-P1.7	**Port 1:** 8-bit quasi-bidirectional I/O port. It receives the low-order address byte during program verification. Port 1 can sink/source one TTL (= 4 LS TTL) input. It can drive MOS inputs without external pull-ups.
9		**RST/V_{PD}**: a high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown permits Power-On reset using only a capacitor connected to V_{CC}. As an available option, this pin also supplies standby power to the RAM: V_{PD} should be held within its specified limit while V_{CC} drops below its specified limit. When V_{PD} is LOW the RAM current is drawn from V_{CC}.
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	**$\overline{\text{INT0}}$**: external interrupt 0 or gate control input for timer/event counter 0
P3.3	**$\overline{\text{INT1}}$** : 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	**$\overline{\text{WR}}$**: external data memory write strobe
P3.7	**$\overline{\text{RD}}$** : external data memory read strobe.

Operation of an alternative function is determined by the relevant output latch programmed to logic 1. Port 3 can sink/source one TTL input. It can drive MOS inputs without external pull-ups.

18	XTAL 2	**Crystal input 2**: 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 (see figures 6 and 7).
19	XTAL 1	**Crystal input 1**: input to the inverting amplifier that forms the oscillator. Connected to V_{SS} when an external oscillator is used (see figures 6 and 7).
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. It emits the high-order address byte when accessing external memory. It also receives the high-order address bits and control signals during program verification. Port 2 can sink/source one TTL input. It can drive MOS inputs without external pull-ups.
29	$\overline{\text{PSEN}}$	**Program Store Enable output**: read strobe to the external Program Memory. It is activated twice each machine cycle during fetches from external Program Memory. When executing out of external Program Memory two activations of $\overline{\text{PSEN}}$ are skipped during each access to external Data Memory. $\overline{\text{PSEN}}$ is not activated (remains HIGH) during fetches from internal Program Memory.
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.

PINNING (continued)

31	$\overline{EA}$	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 4096. When $\overline{EA}$ is held at a TTL low level the CPU executes out of external Program Memory. $\overline{EA}$ does not float.
32-39	P0.7-P0.0	**Port 0**: 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during these accesses it activates internal pull-ups). It also outputs instruction bytes during program verification. (External pull-ups are required during program verification). Port 0 can sink/source two TTL inputs.
40	V_{CC}	**Power Supply**: + 5 V power supply pin during normal operation.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

GENERAL DESCRIPTION

The MAB80XXH family of single-chip 8-bit microcontrollers are fabricated in NMOS.
Three interchangeable (pin compatible) versions are available:

- MAB8048H with resident mask-programmed 1 K x 8 ROM, 64 x 8 RAM
- MAB8035HL without resident program memory for use with external EPROM/ROM
- MAB8049H with resident mask-programmed 2 K x 8 ROM, 128 x 8 RAM
- MAB8039HL without resident program memory for use with external EPROM/ROM
- MAB8050H with resident mask-programmed 4 K x 8 ROM, 256 x 8 RAM
- MAB8040HL without resident program memory for use with external EPROM/ROM

The MAB80XXH family are designed to be efficient control processors as well as arithmetic processors. Their instruction set allows the user to directly set and reset individual I/O lines as well as test individual bits within the accumulator. A large variety of branch and table look-up instructions enable efficient 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, which can count either machine cycles (÷ 32) or external events. The counter can be programmed to cause an interrupt to the processor.
Program and data memories plus input/output capabilities can be expanded using standard devices.
For details see users manual 'single-chip microcomputer'.

Features

- 8-bit CPU, ROM, RAM and I/O
- Internal counter/timer
- Internal oscillator, clock driver
- 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
- Easily expandable memory and 27 I/O lines
- TTL compatible inputs and outputs
- Single 5 V supply
- Standard and extended temperature range

Applications

- Peripheral interfaces and controllers
- Test and measuring instruments
- Sequencers
- Modems and data enciphering
- Environmental control systems
- Audio/video systems

PACKAGE OUTLINES

All versions ; with type no. suffix P: 40-lead DIL; plastic (SOT-129).
MAB80XXH/HLWP : 44-lead plastic leaded chip-carrier (PLCC); SOT-187A.

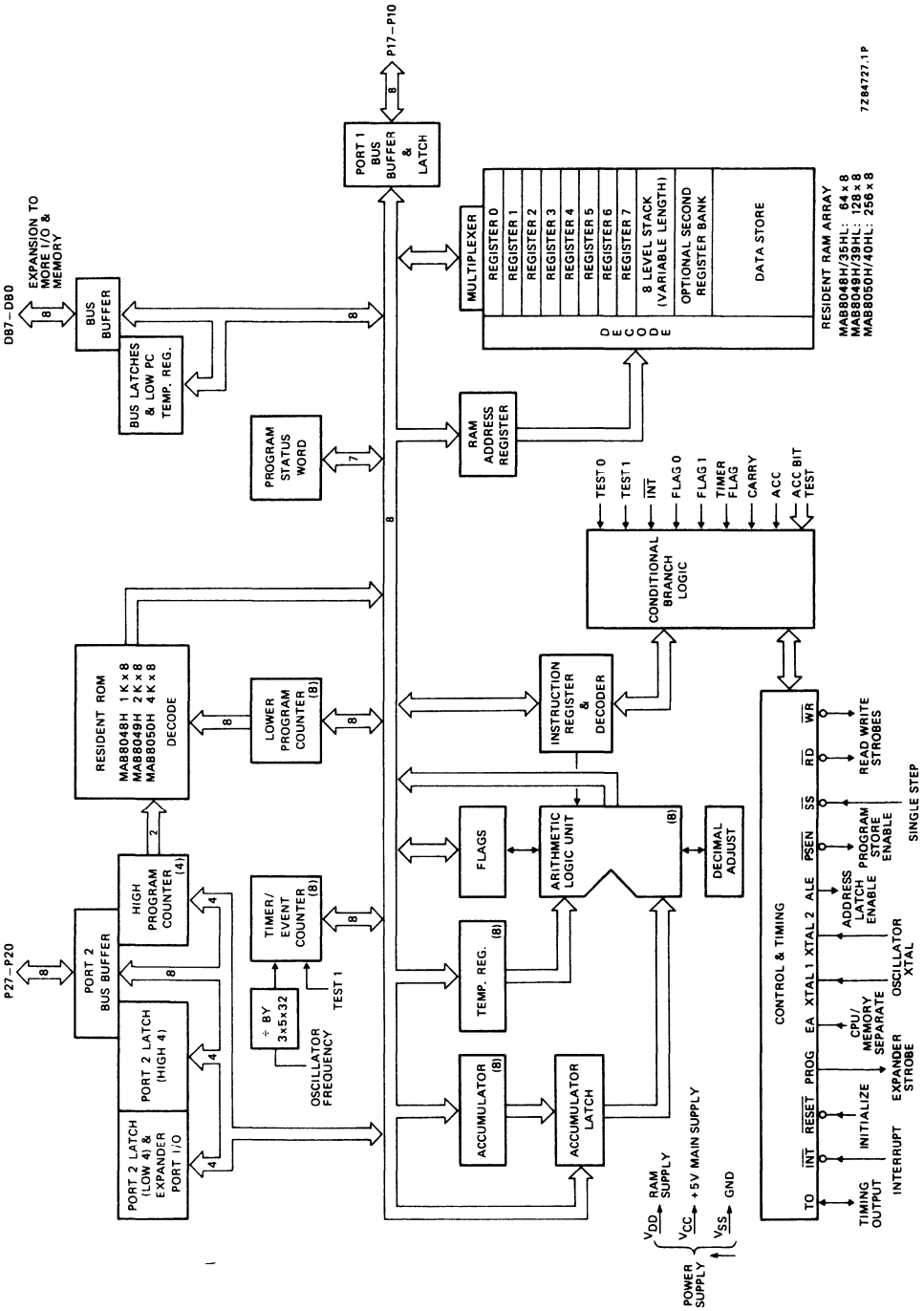


Fig. 1 Block diagram.

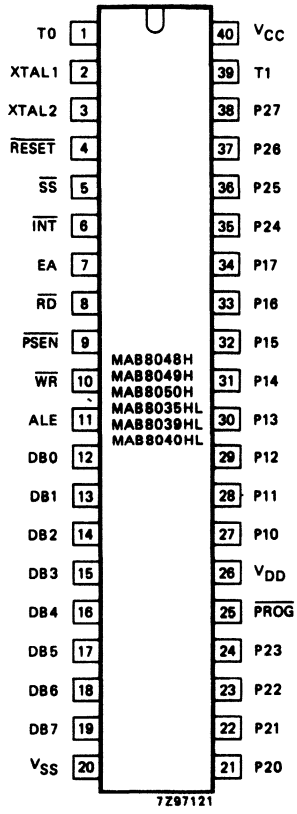
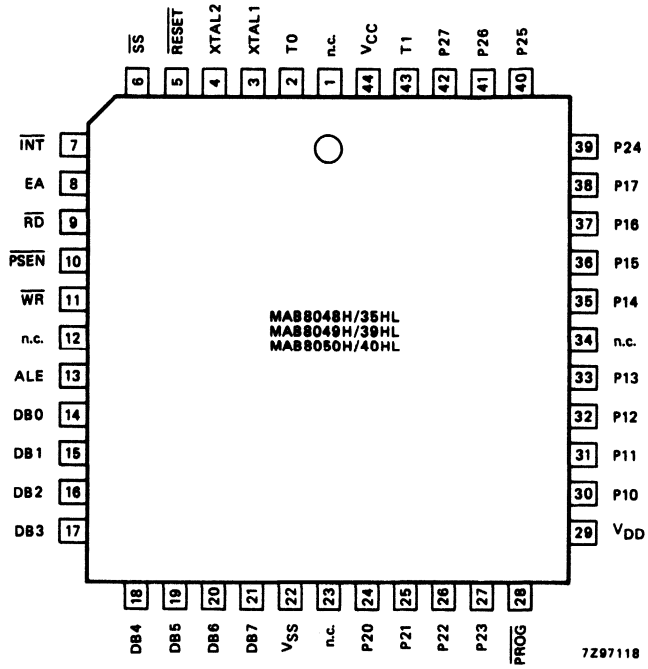


Fig. 2a Pinning diagram; for pin designation see next page.



Where: n.c. = not connected.

Fig. 2b Pinning diagram for MAB80XXH/HLWP; for pin designation see next page.

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. $\overline{\text{RESET}}$ sets all lines to logic 1.
2. Non-standard TTL V_{IH}.

PINNING

12–19	DB0–DB7	**Data Bus**: bidirectional I/O port which can 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	P10–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 instructions.
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 microcontroller. 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 3	XTAL 1 XTAL 2	**Crystal inputs**: inputs for a crystal, LC-network or an external timing signal to determine the internal oscillator frequency (note 2).
20	V_{SS}	**Ground**: circuit earth potential.
40	V_{CC}	**Power supply**: + 5 V main supply pin.
26	V_{DD}	**Power supply**: + 5 V RAM standby power supply; low power standby pin.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT CMOS MICROCONTROLLER

DESCRIPTION

The PC80CXX family of single-chip 8-bit CMOS microcontrollers consists of:

- The PCB80C49 with resident mask programmed ROM.
- The PCB80C39 without resident program memory for use with external EPROM/ROM.

All versions are pin and function compatible to their NMOS counter parts but with additional features and high performance.

The PC80CXX family are designed to be efficient control processors as well as arithmetic processors. Their instruction set allows the user to directly set and reset individual I/O lines as well as test individual bits within the accumulator. A large variety of branch and table look-up instructions enable efficient 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, which can count 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. Input/output capabilities can be expanded using standard devices.
The family has low power consumption and in addition a power down mode is provided.

FEATURES

- 8-bit CPU, ROM, RAM, I/O in a single 40-pin package
- PCB80C49: 2K x 8 ROM, 128 x 8 RAM
- Internal counter/timer
- Internal oscillator, clock driver
- 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
- Easily expandable memory and I/O
- TTL compatible inputs and outputs
- Single 5 V supply
- Wide frequency operating range
- Low current consumption
- Also available with extended temperature range; PCF 80CXX = −40 °C to + 85 °C

APPLICATIONS

- Peripheral interfaces and controllers
- Test and measurement instruments
- Sequencers
- Audio/video systems
- Environmental control systems
- Modems and data enciphering

PACKAGE OUTLINES

PCB/F80C39/C49P: 40-lead DIL; plastic (SOT-129).
PCB80C39/C49WP: 44-lead plastic leaded chip-carrier (PLCC); SOT-187A.

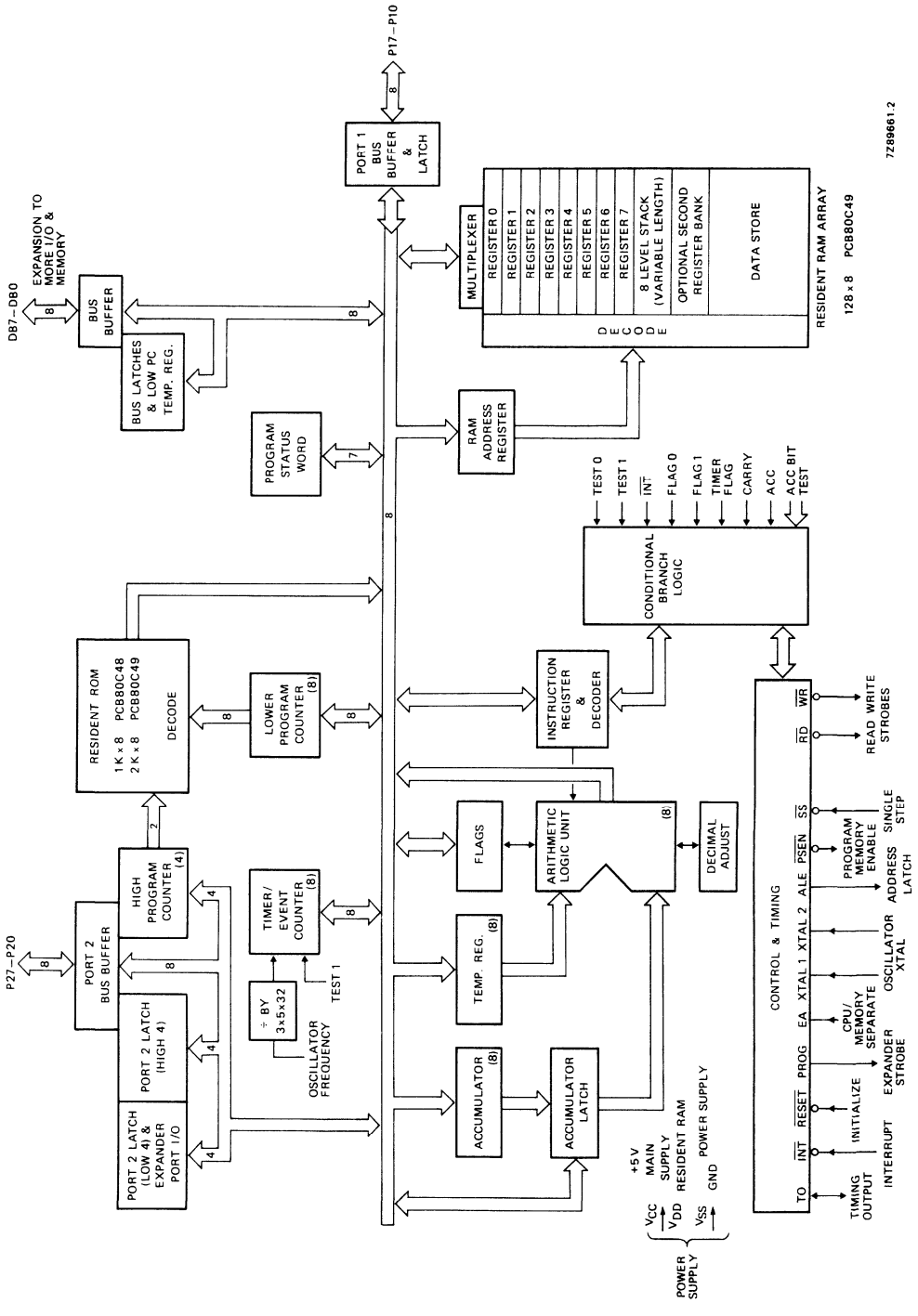


Fig. 1 Block diagram.

DEVELOPMENT DATA

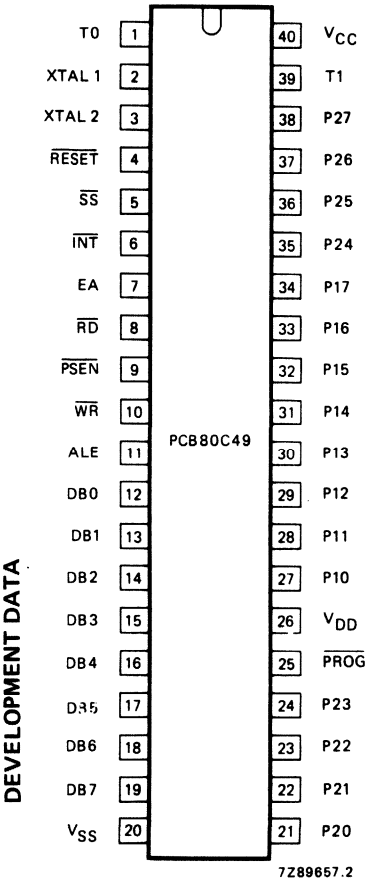


Fig. 2a Pinning diagram; for pin designation see next page.

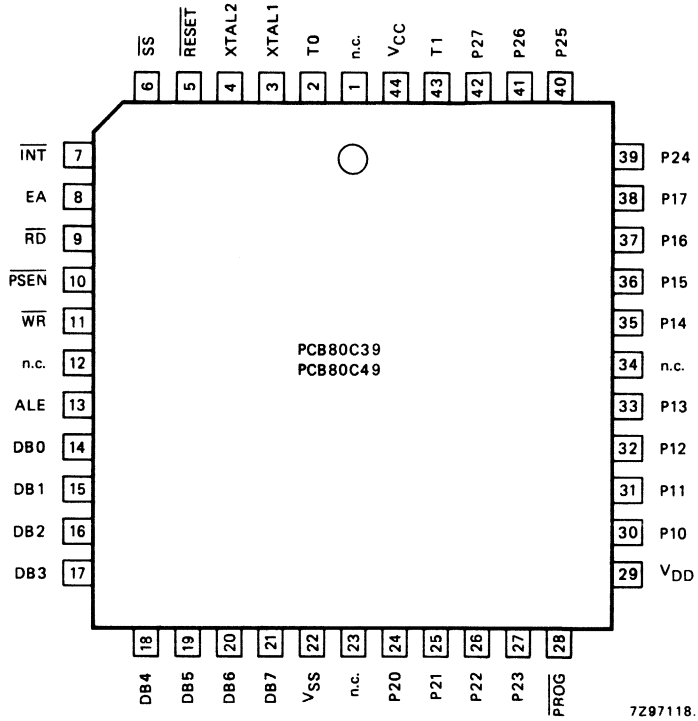


Fig. 2b Pinning diagram for PCB80CXXWP; for pin designation see next page.

Where: n.c. = not connected.

PIN DESIGNATION

designation	pin no.	function
DB0–DB7	12–19	**BUS.** Bidirectional I/O port that can be read or written using the $\overline{RD}$ and $\overline{WR}$ strobes. This port can also be statically latched. 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.
P10–P17	27–34	**Port 1.** 8-bit quasi-bidirectional I/O port (note 1).
P20–P27	21–24, 35–38	**Port 2.** 8-bit quasi-bidirectional I/O port (note 1). P20–P23 contain the 4 higher order address bits during an access of external program memory.
$\overline{PROG}$	25	**Output strobe** (active LOW) for I/O expander.
T0	1	**Input pin** sensed using the JT0 and JNT0 instructions. **Clock output pin** when designated as such by the ENT0 CLK instruction.
T1	39	**Input pin** sensed using the JT1 and JNT1 instructions. Can be designated as the timer/counter input by the STRT CNT instruction.
$\overline{INT}$	6	**Interrupt input pin.** When LOW causes an interrupt in the current program if external interrupt is enabled. Can also be used as an input, testable using the JNI instruction. Interrupt is disabled during and after a RESET.
$\overline{RESET}$	4	**Reset input pin** used to initialize the microcontroller. Active LOW. 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).
ALE	11	**Address latch enable.** Occurs once each machine cycle and is useful 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.
$\overline{RD}$	8	**Read BUS.** Active LOW strobe used to gate data onto BUS lines when reading from an external source.
$\overline{WR}$	10	**Write BUS.** Active LOW strobe used to write data from BUS lines to an external designation.
EA	7	**External access input.** When HIGH, forces instruction fetch from external memory.
$\overline{PSEN}$	9	**Program store enable.** Active LOW strobe that occurs only during a fetch from external program memory.
$\overline{SS}$	5	**Single step input.** Active LOW which is used with ALE to cause the microcontroller to execute a single instruction.
V_{DD}	26	**RAM power supply,** + 5 V during normal operation and power-down mode.

designation	pin no.	function
XTAL1	2	**One side of crystal** (or inductor) input for internal oscillator. Can also be used as an input for an external timing source (note 2).
XTAL2	3	**Other side of crystal.**
V_{SS}	20	**Ground.**
V_{CC}	40	**Main power supply**, + 5 V during normal operation.

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. $\overline{\text{RESET}}$ sets all lines to logic '1'.
2. Non-standard TTL V_{IH}.

FOR DETAILED INFORMATION SEE RELEVANT DATA BOOK OR DATA SHEET

SINGLE-CHIP 8-BIT MICROCONTROLLER

DESCRIPTION

The PCB80C51 family of single-chip 8-bit microcontrollers is manufactured in an advanced CMOS process. The family consists of the following members:

- PCB80C31: ROM-less version of the PCB80C51
- PCB80C51: 4 K bytes mask-programmable ROM, 128 bytes RAM

In the following, the generic term "PCB80C51" is used to refer to both family members.

The device provides hardware features, architectural enhancements and new instructions to function as a controller for applications requiring up to 64 K bytes of program memory and/or up to 64 K bytes of data storage.

The PCB80C51 contains a non-volatile 4 K x 8 read-only program memory (not ROM-less version); a volatile 128 x 8 read/write data memory; 32 I/O lines; two 16-bit timer/event counters; a five-source, two-priority-level, nested interrupt structure; a serial I/O port for either multi-processor communications, I/O expansion, or full duplex UART; and on-chip oscillator and timing circuits. For systems that require extra capability, the PCB80C51 can be expanded using standard TTL compatible memories and logic.

The PCB80C31/80C51 has two software selectable modes of reduced activity for further power reduction – Idle and Power Down.
The Idle modes freezes the CPU while allowing the RAM, timers, serial port 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.

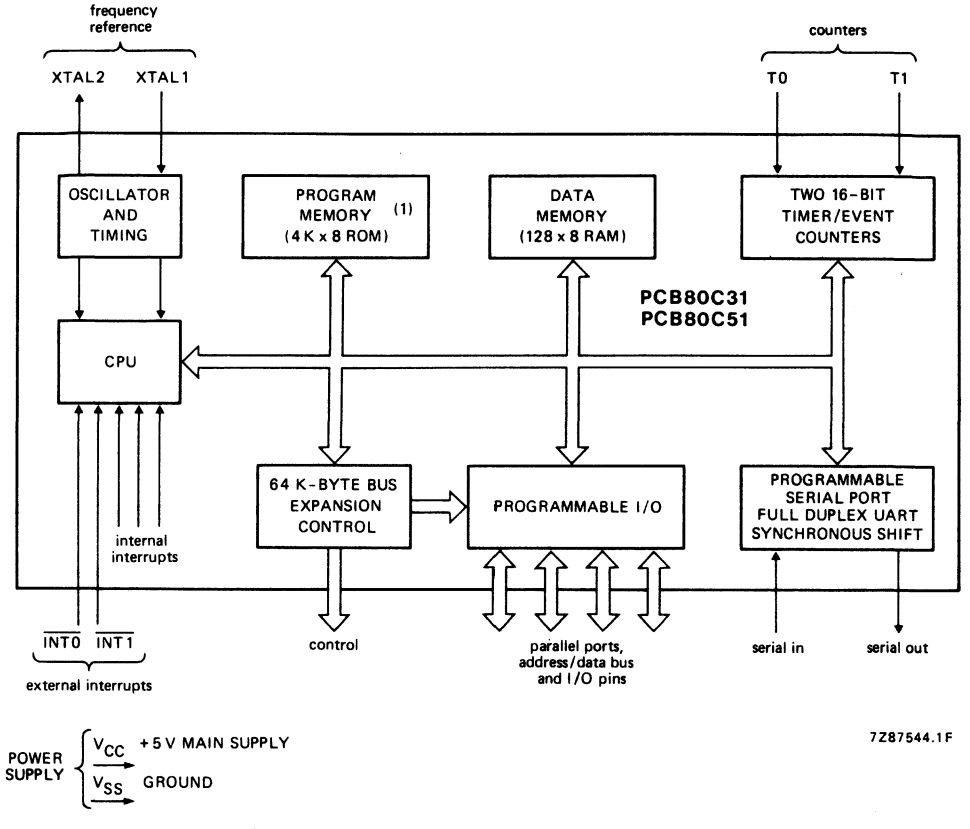
The device also functions as an arithmetic processor having facilities for both binary and BCD arithmetic plus bit-handling capabilities. The instruction set consists of 255 instructions; 44% one-byte, 41% two-byte and 15% three-byte. With a 12 MHz crystal, 58% of the instructions are executed in 1 μs and 40% in 2 μs. Multiply and divide instructions require 4 μs. Multiply, divide, subtract and compare are among the many instructions added to the standard PCB80C48 instruction set.
Software development to be announced: PCB85C51 in piggy-back and 84 pin PLCC.

Features

- 4 K x 8 ROM (80C51 only), 128 x 8 RAM
- Four 8-bit ports, 32 I/O lines
- Two 16-bit timer/event counters
- Full-duplex serial port
- External memory expandable to 128 K, external ROM up to 64 K and/or external RAM up to 64 K
- Boolean processing
- 218 bit-addressable locations
- On-chip oscillator
- Five-source interrupt structure with two priority levels
- 58% of instructions executed in 1 μs; multiply and divide in 4 μs; all others executed in 2 μs (at 12 MHz clock)
- Enhanced architecture with:
  non-page-oriented-instructions
  direct addressing
  four 8-byte + 1 byte register banks
  stack depth up to 128-bytes
  multiply, divide, subtract and compare instructions.

PACKAGE OUTLINES

PCB80C31/51P: 40-lead DIL; plastic (SOT-129).
PCB80C31/51WP: 44-lead PLCC; plastic, leaded-chip-carrier (SOT-187A).



(1) PCB80C51 only.

Fig. 1 Block diagram.

DEVELOPMENT DATA

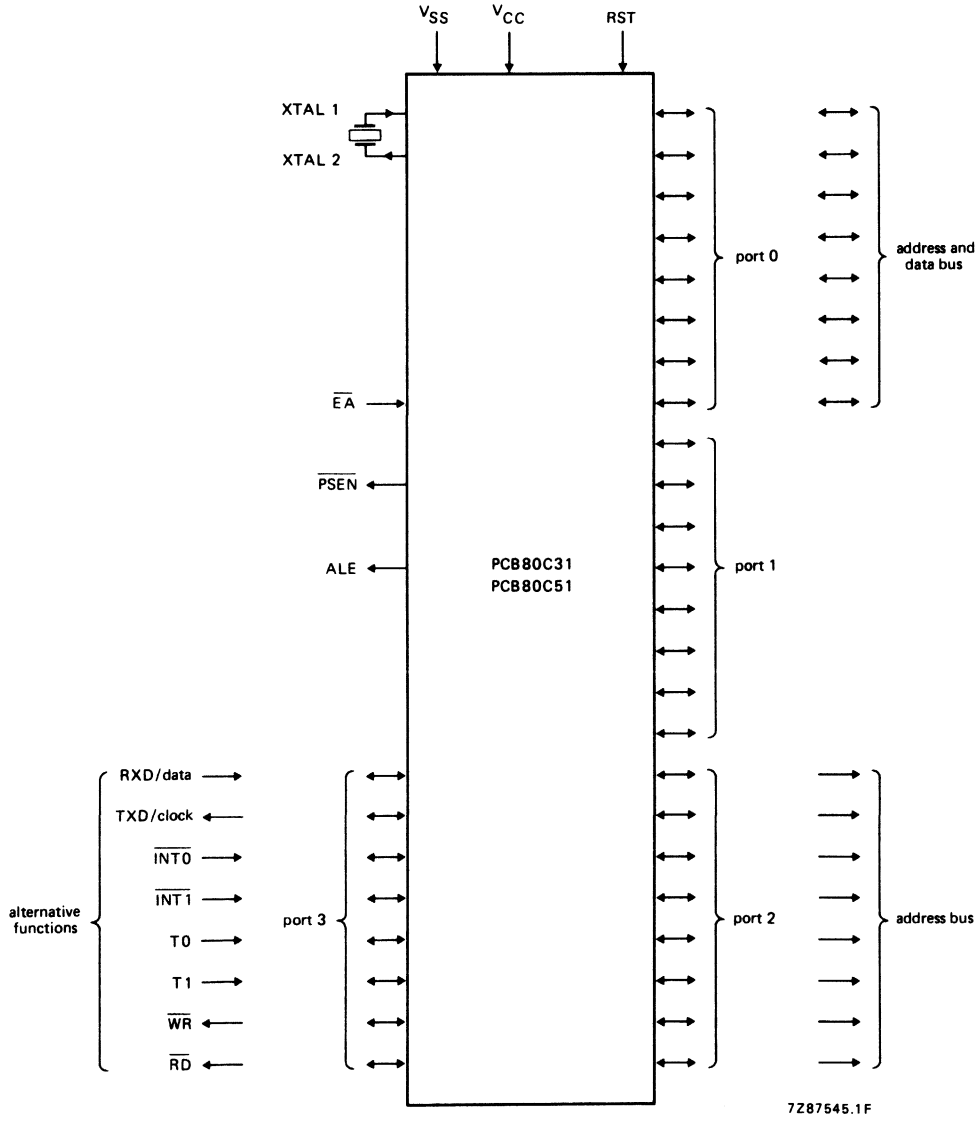


Fig. 2 Functional diagram.

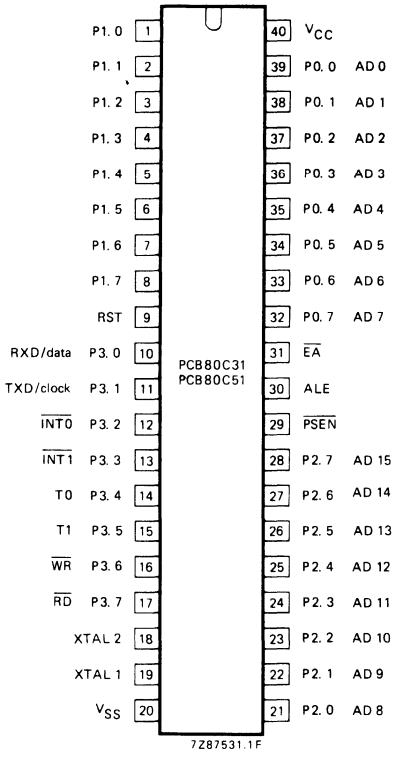


Fig. 3a Pinning diagram for PCB80C31/51P.

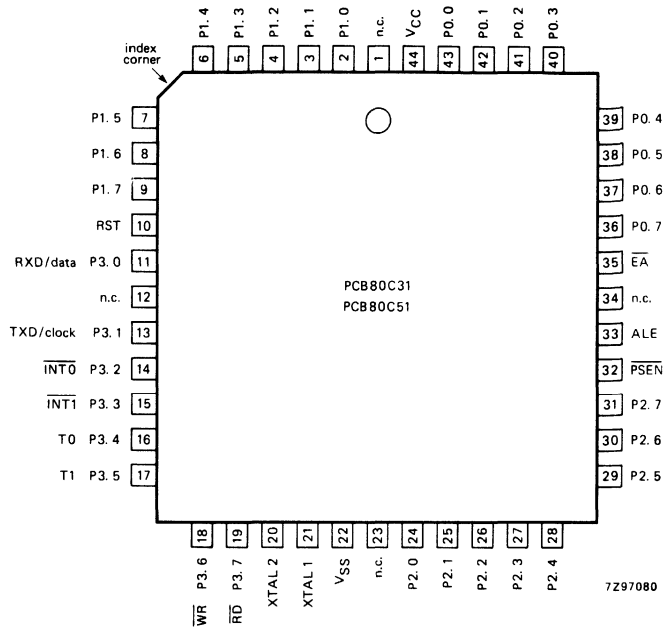


Fig. 3b Pinning diagram for PCB80C31/51WP.

PINNING (PCB80C31/51P)

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. It can drive CMOS inputs without external pull-ups.
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_{CC}.
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:

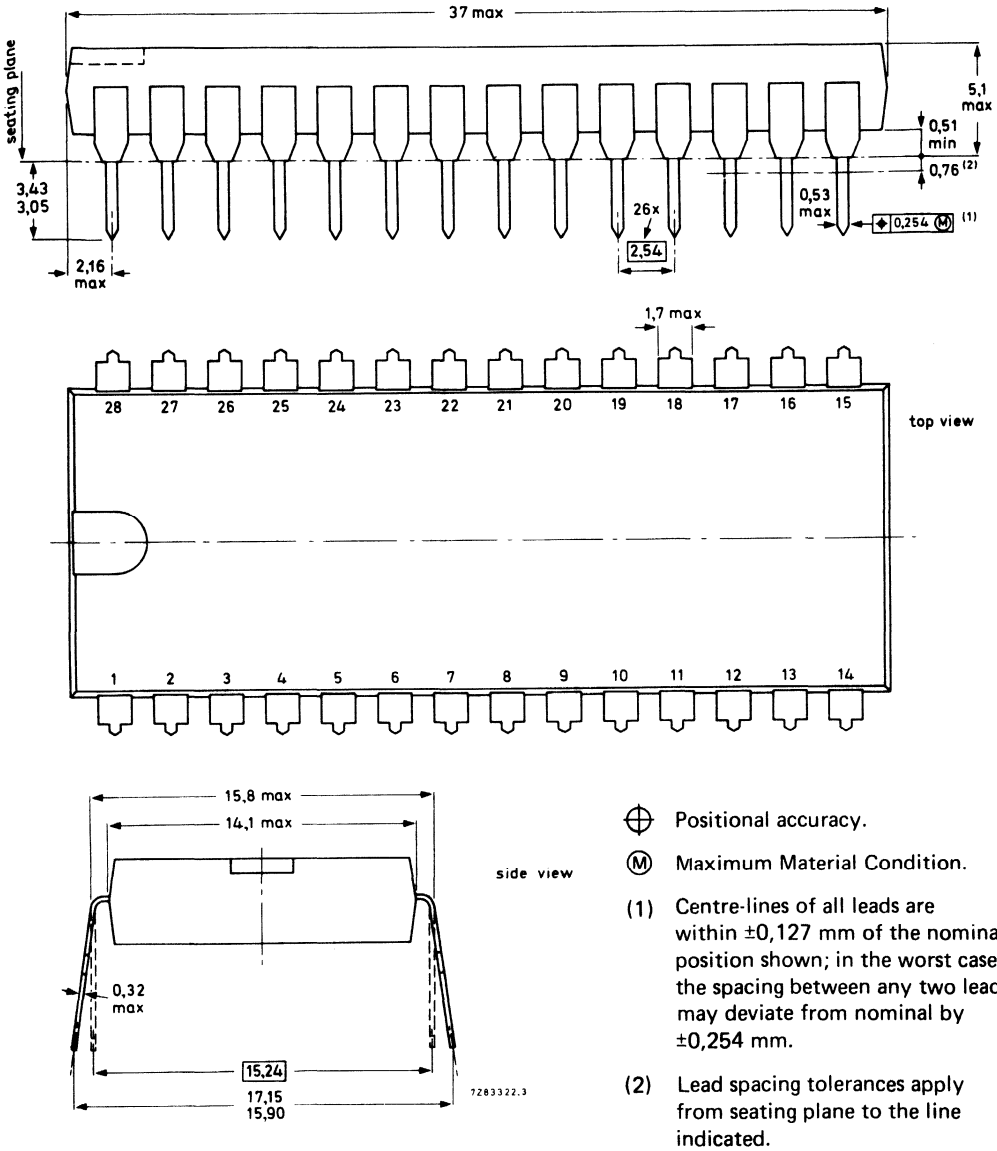
Port pin	*Alternative function*
P3.0	**RXD/data**: serial port receiver data input (asynchronous) or data input/output (synchronous)
P3.1	**TXD/clock**: serial port transmitter data output (asynchronous) or clock output (synchronous)
P3.2	**$\overline{\text{INT0}}$**: external interrupt 0 or gate control input for timer/event counter 0
P3.3	**$\overline{\text{INT1}}$**: 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
		Operation of an alternative function is determined by the relevant output latch programmed to logic 1. Port 3 can sink/source one TTL 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 is used (see figures 8 and 9).
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 is used (see figures 8 and 9).
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. Port 2 can sink/source one TTL input. It can drive CMOS inputs without external pull-ups. During the access to external memories (RAM/ROM) that uses 16-bit addresses (MOVX @DPTR) Port 2 emits the high-order address byte. When used for an external RAM an 8-bit address (MOVX @Ri) Port 2 emits the contents of the P2 Special Function Register.
29	$\overline{PSEN}$	**Program Store Enable output**: read strobe to the external Program Memory. 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 fetches from internal 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 4096. When $\overline{EA}$ is held at a TTL low level, the CPU executes out of external Program Memory. $\overline{EA}$ is not allowed to float.
32–39	P0.7–P0.0	**Port 0**: 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during these accesses it activates internal pull-ups). Port 0 can sink/source eight TTL inputs.
40	V_{CC}	**Power supply**: + 5 V power supply pin during normal operation, Idle mode and Power Down mode.

DEVELOPMENT DATA

16. Package outlines

28-LEAD DUAL IN-LINE; PLASTIC (SOT-117D)



⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

(1) Centre-lines of all leads are within ±0,127 mm of the nominal position shown; in the worst case, the spacing between any two leads may deviate from nominal by ±0,254 mm.

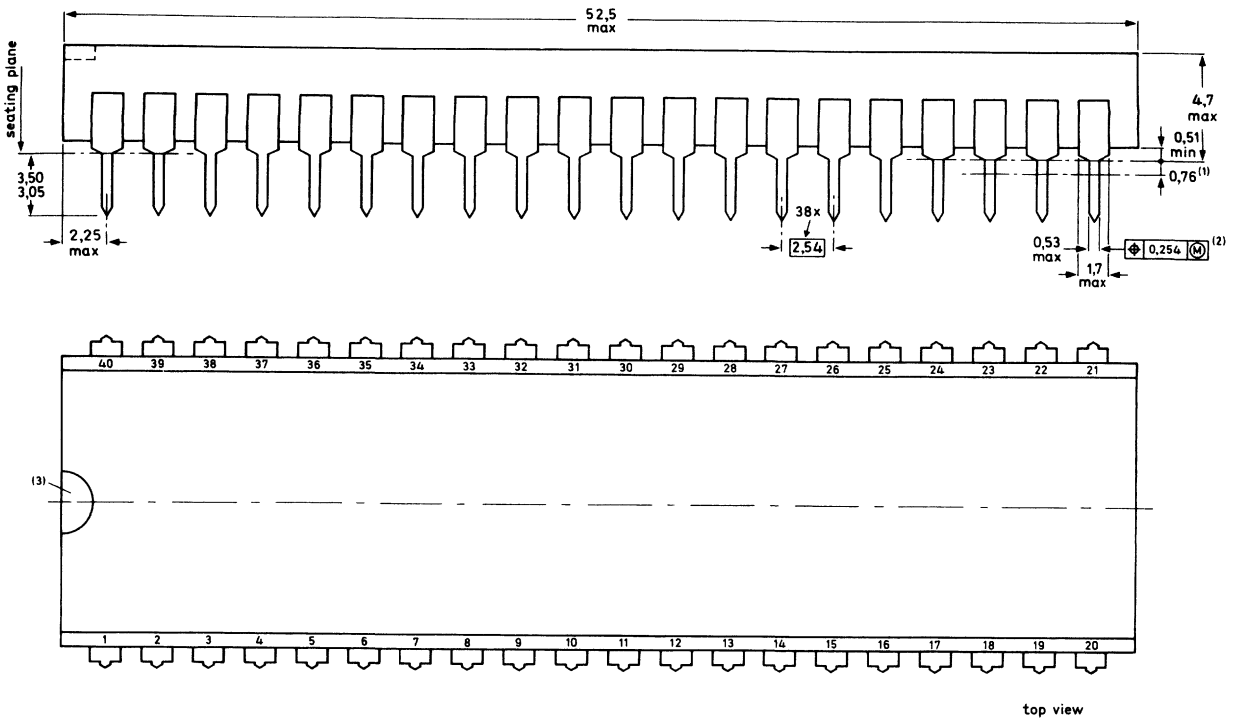
(2) Lead spacing tolerances apply from seating plane to the line indicated.

Dimensions in mm

SOLDERING

See SOT-135A.

40-LEAD DUAL IN-LINE; PLASTIC (SOT-129)



⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

(1) Centre-lines of all leads are within ±0,127 mm of the nominal position shown; in the worst case, the spacing between any two leads may deviate from nominal by ±0,254 mm.

(2) Lead spacing tolerances apply from seating plane to the line indicated.

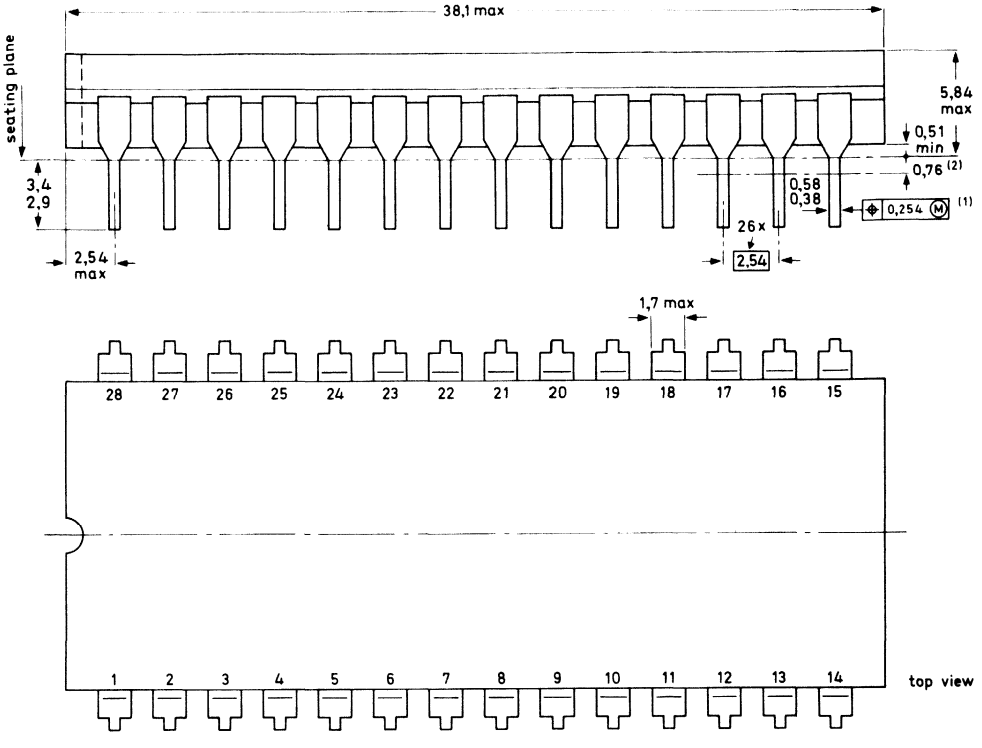
(3) Index may be horizontal as shown, or vertical.

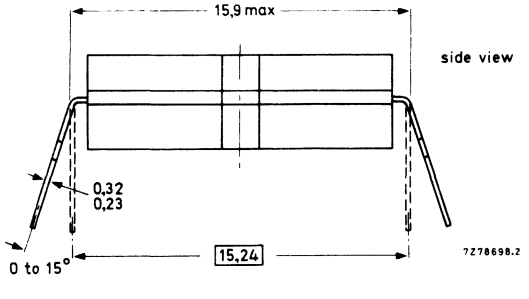
Dimensions in mm

SOLDERING

See next page.

28-LEAD DUAL IN-LINE; CERAMIC (CERDIP) (SOT-135A)





⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

(1) Centre-lines of all leads are within ±0,127 mm of the nominal position shown; in the worst case, the spacing between any two leads may deviate from nominal by ±0,254 mm.

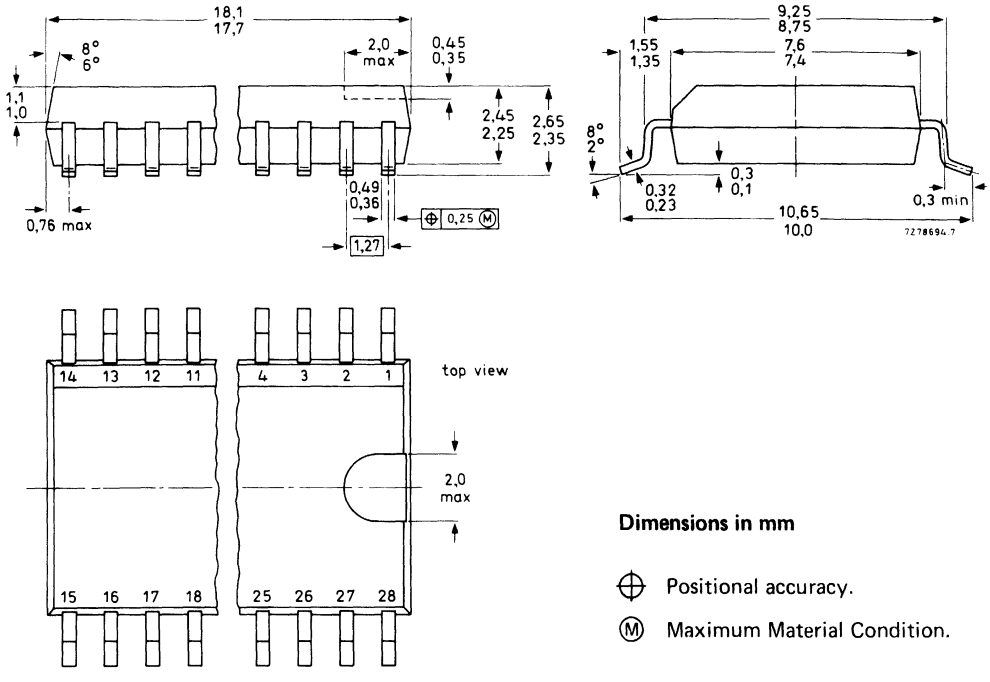
(2) Lead spacing tolerances apply from seating plane to the line indicated.

Dimensions in mm

SOLDERING

See next page.

28-LEAD MINI-PACK; PLASTIC (SO-28; SOT-136A)



Dimensions in mm

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

SOLDERING

The reflow solder technique

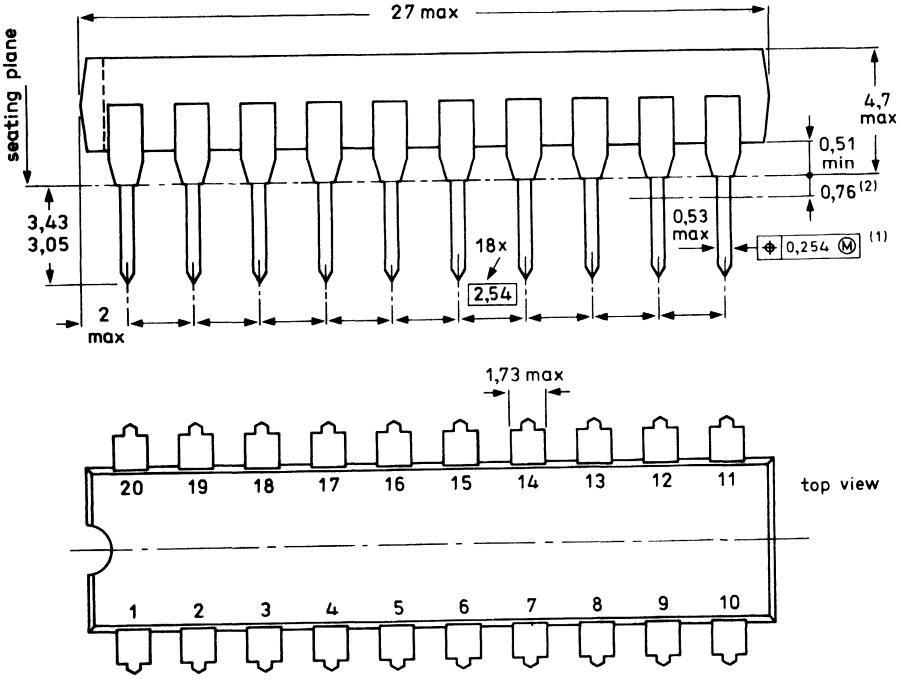
The preferred technique for mounting miniature components on hybrid thick or thin-film circuits is reflow soldering. Solder is applied to the required areas on the substrate by dipping in a solder bath or, more usually, by screen printing a solder paste. Components are put in place and the solder is reflowed by heating.

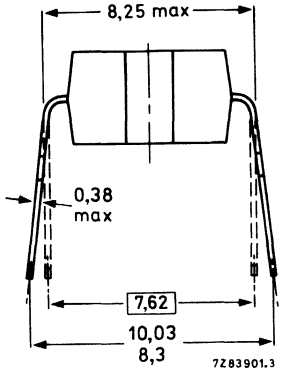
Solder pastes consist of very finely powdered solder and flux suspended in an organic liquid binder. They are available in various forms depending on the specification of the solder and the type of binder used. For hybrid circuit use, a tin-lead solder with 2 to 4% silver is recommended. The working temperature of this paste is about 220 to 230 °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 μm is used for which the emulsion thickness should be about 50 μm. 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 keep them in place. The substrate is heated to the solder working temperature preferably by means of a controlled hot plate. The soldering process should be kept as short as possible: 10 to 15 seconds is sufficient to ensure good solder joints and evaporation of the binder fluid.
After soldering, the substrate must be cleaned of any remaining flux.

20-LEAD DUAL IN-LINE; PLASTIC (SOT-146; 146C1)





side view

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

(1) Centre-lines of all leads are within ±0,127 mm of the nominal position shown; in the worst case, the spacing between any two leads may deviate from nominal by ±0,254 mm.

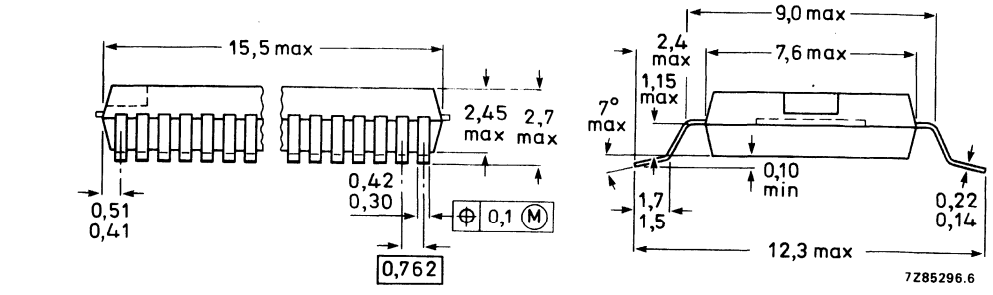
(2) Lead spacing tolerances apply from seating plane to the line indicated.

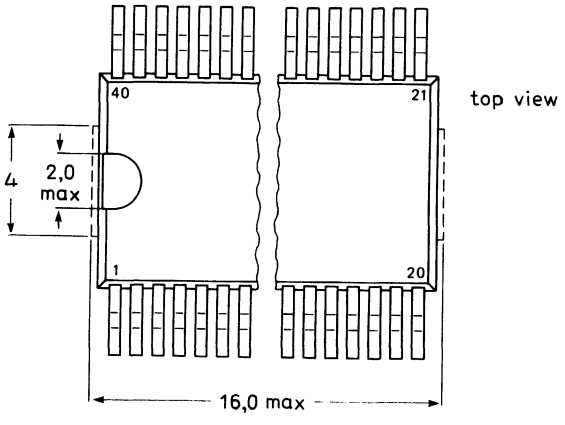
Dimensions in mm

SOLDERING

See next page.

40-LEAD MINI-PACK; PLASTIC (VSO-40; SOT-158A)





Dimensions in mm

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

SOLDERING

The reflow solder technique

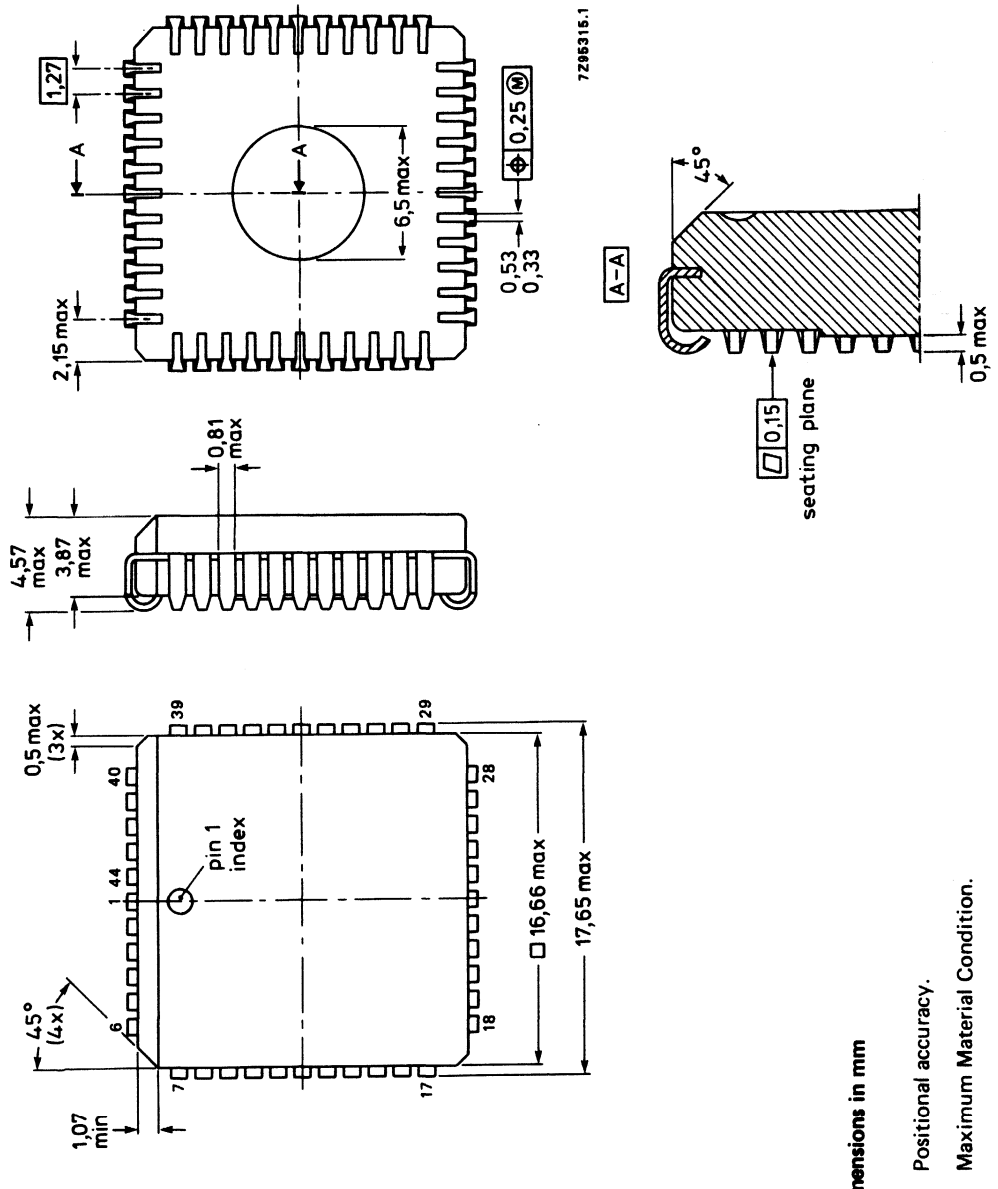
The preferred technique for mounting miniature components on hybrid thick or thin-film circuits is reflow soldering. Solder is applied to the required areas on the substrate by dipping in a solder bath or, more usually, by screen printing a solder paste. Components are put in place and the solder is reflowed by heating.

Solder pastes consist 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 to 230 °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 μm is used for which the emulsion thickness should be about 50 μm. 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 keep them in place. The substrate is heated to the solder working temperature preferably by means of a controlled hot plate. The soldering process should be kept as short as possible: 10 to 15 seconds is sufficient to ensure good solder joints and evaporation of the binder fluid.
After soldering, the substrate must be cleaned of any remaining flux.

44-LEAD PLASTIC LEADED CHIP-CARRIER (PLCC); SOT-187A

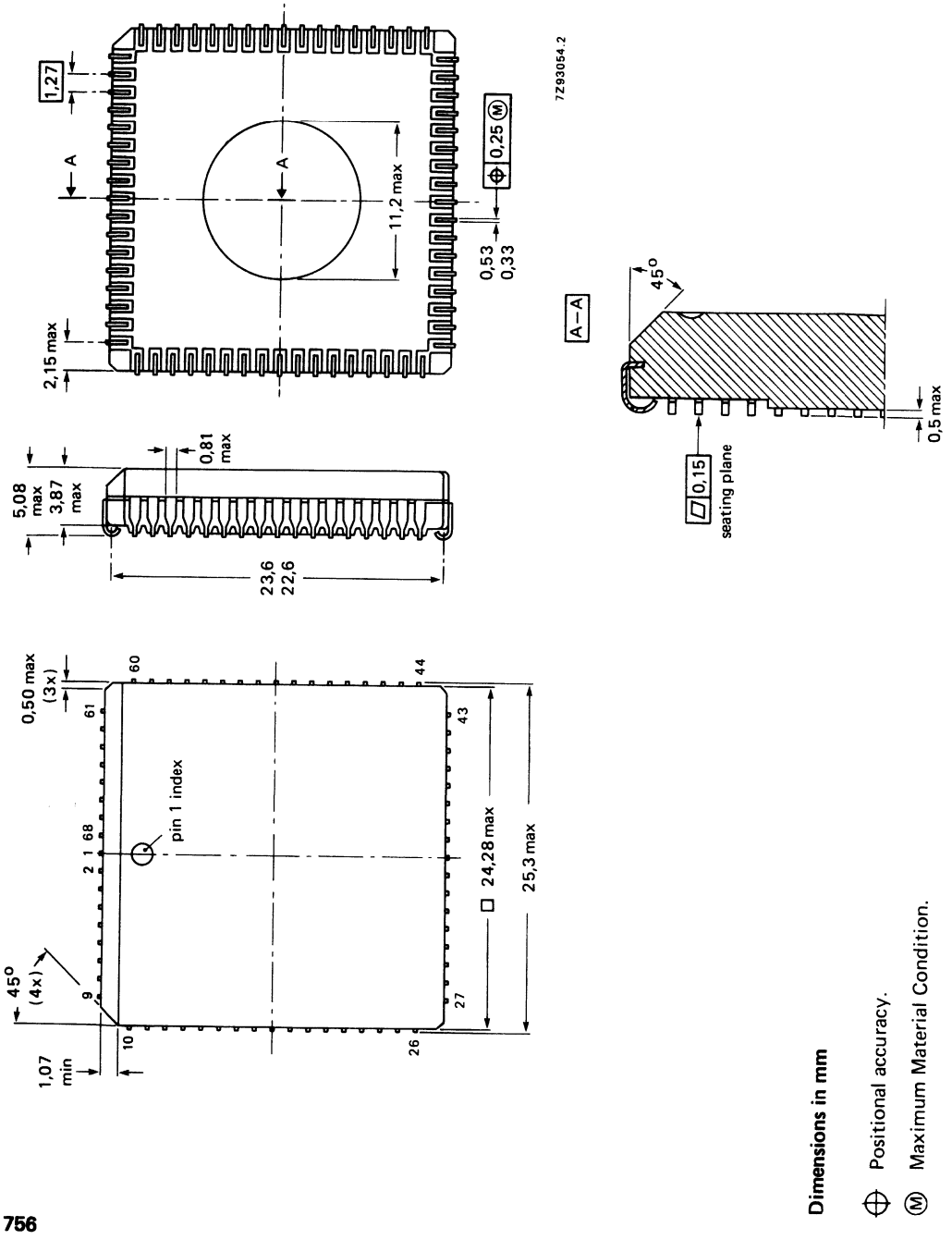


Dimensions in mm

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

68-LEAD PLASTIC LEADED CHIP-CARRIER (PLCC); SOT-188A

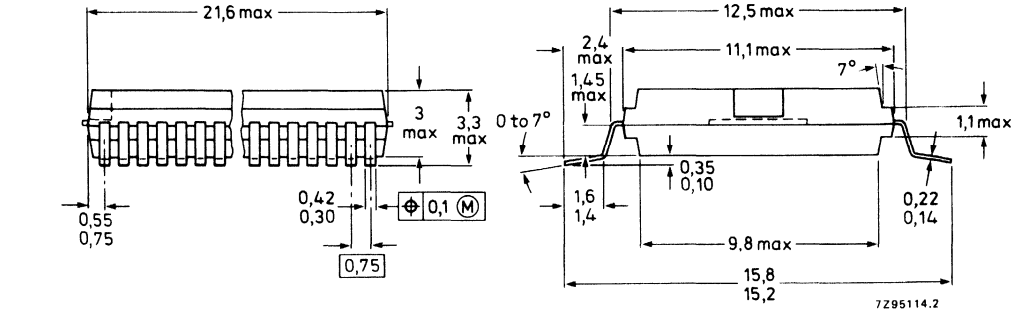


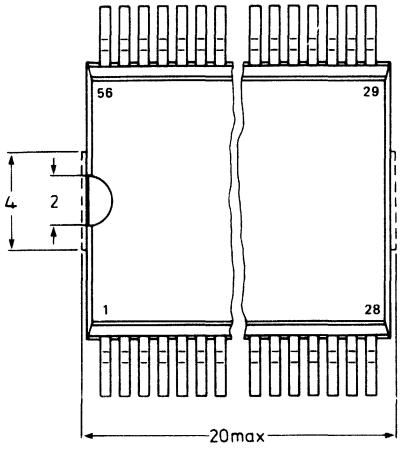
Dimensions in mm

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

56-LEAD MINI-PACK; PLASTIC (VSO-56; SOT-190)





top view

Dimensions in mm

⊕ Positional accuracy.

Ⓜ Maximum Material Condition.

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 maximum 10 seconds up to 300 °C, or 5 seconds up to maximum 400 °C.
When using the proper tools, all pins can be soldered in one operation within 2 to 5 seconds and 270 to 320 °C.

2. By dip or wave

The maximum permissible temperature of the solder is 260 °C. The permissible total time of immersing the whole package in the bath is 10 seconds, if it is allowed to cool down to less than 150 °C within 6 seconds.

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 °C.

17. Order entry forms

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 2QTY WK 1. QTY 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
☐	MAB 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 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 2095/01

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 20?? WK 1. QTY WK ?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 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 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 2094/01

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			

PORTS:
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 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 2093/01

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 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 2092/01

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 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 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 + HARDCOPY (LISTING)

C INTELLEC FLOPPY DISK (SINGLE OR DOUBLE DENSITY)

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

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

8021 - 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)	SUPPLY VOLTAGE (V)	FREQUENCY (MHZ) MIN.	MAX.
☐	MAB 8021	0 ... +70	70	4,5 ... 6,5	0,6	3,6
☐	MAF 8021	-40 ... +85	82	4,75 .. 6,25	0,6	3,2

WORK IN PROGRESS LIABILITY OPTION	WE ACCEPT VOLUME DELIVERIES WITHOUT APPROVAL OF SAMPLE DELIVERY, AND ARE LIABLE FOR WORK IN PROCESS - MIN. 3,5 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 2098/01