



Electronic
components
and materials

PHILIPS

2650 MICROPROCESSOR
USER'S MANUAL

(UPDATE 3)

signetics

2650 MICROPROCESSOR USER'S MANUAL (update 3)

Dear Sir,

The accompanying Application Notes supplement your Signetics 2650 Microprocessor User's Manual and are punched to fit the same binder.

Together with this update we would like to inform you of the current status of our Microprocessor Support Hardware and Course Schedule.

Since the last update many new enthusiastic 2650 users have joined with many new applications for our MOS Microprocessor. Moreover after several months of evaluating alternate sources, National Semiconductor Corporation of Santa Clara decided to second source the 2650, and the contract with Signetics has been signed.

In particular the TWIN Prototype Development System has gained acceptance as the Industry Standard mainly due to its unique dual-CPU architecture.

Registration

To ensure that you continue receiving Application Notes free of charge, it is essential that we have your correct address. To check on that, please inspect the address label with which this came to you. If any of the particulars are incorrect, enter the correct particulars on the upper part of the registration form you will find in the front of your Manual and return the form to us.

We shall then transfer the correction to the lower part of the form which we have kept on file.

Yours faithfully,

Microprocessor Marketing Group.

2650 APPLICATION NOTES

With this update (marked* in the list below) your manual should now contain the following Application Notes:

No.	Title	Summary
AS50	Serial Input/Output	Describes how the Sense/Flag capability of the 2650 can be used for serial I/O interfaces.
AS51	Bit & Byte Testing Procedures	Describes several methods of testing the contents of the internal registers in the 2650.
AS52	General Delay Routines	Describes several ways of writing software time delay routines for the 2650, including formulas for calculating the delay time.
AS53	Binary Arithmetic Routines	Provides examples for processing binary arithmetic addition, subtraction, multiplication, and division with the 2650.
AS54	Conversion Routines	Describes routines for converting: <ul style="list-style-type: none">– Eight-bit unsigned binary to BCD– Sixteen-bit signed binary to BCD– Signed BCD to binary– Signed BCD to ASCII– ASCII to BCD– Hexadecimal to ASCII– ASCII to Hexadecimal
*AS55	Fixed Point Decimal Arithmetic Routines	Describes methods of performing addition, subtraction, multiplication and division of binary-coded-decimal (BCD) numbers with the 2650.
SP50	2650 Evaluation Printed Circuit Board (PC1001)	Provides a detailed description of the PC1001, an evaluation and design tool for the 2650.
SP51	2650 Demo System	Provides a detailed description of the Demo System, a hardware base for use with the 2650 CPU prototyping board (PC1001 or PC1500).
SP52	Support Software for use with the NCSS Timesharing System	Provides step-by-step procedures for generating, editing, assembling, punching, and simulating Signetics 2650 programs using the NCSS timesharing service.
SP53	Simulator, Version 1.2	Summarizes the features and characteristics inherent in version 1.2 of the 2650 simulator.
SP54	Support Software for use with the General Electric Mark III Timesharing System	Provides step-by-step procedures for generating, editing, assembling, simulating, and punching Signetics 2650 programs using General Electric's Mark III timesharing system.
*SP55	The ABC 1500 Adaptable Board Computer	Describes the various components and applications of the ABC (Adaptable Board Computer) 1500 system development card.
SS50	PIPBUG	Provides a detailed description of PIPBUG, a monitor program designed for use with the 2650.
SS51	Initialization	Describes the procedures for initializing the 2650 microprocessor, memory, and I/O devices to their described initial states.
MP52	Low-Cost Clock Generator Circuits	Describes several clock generator circuits that may be used with the 2650. These circuits are standard TTL logic elements (7400 series). They include RC, LC and crystal oscillator types.
*MP53	Address and Data Bus Interfacing Techniques	Provides several examples of interfacing the 2650 address and data busses with ROMs and RAMs, such as the 2608, 2606 and 2602.
*MP54	2650 Input/Output Structures and Interfaces	Examines the use of the 2650's versatile set of I/O instructions and the interface between the 2650 and I/O ports. A number of application examples for both serial and parallel I/O are given.

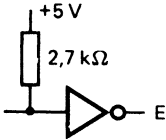
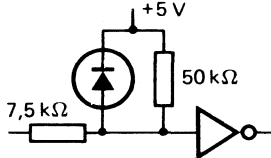
If any of these Application Notes is missing in your manual or if you require additional copies please contact your local Philips Organization.

ERRATA

A small number of errors have been found in the Application Notes issued so far.

Please correct your documentation according to the information below.

If you find mistakes or have any ideas for improvements please inform us, so that others can also benefit from it.

Application Note	Page	Error	Correction
AS53	5, line 56 7, line 83 11, line 181	CD 05 04 F8 02 BDRR,R0 04 FC LODI,R0	CD 02 05 58 02 BRNR,R0 05 FC LODI,R1
SP50	9	Add following note: "If noise problems are encountered when operating with a current loop TTY, the addition of a 4700 pF capacitor at the board location marked C4 is recommended (see figure 2)"	
	14	Teletype connection: pin 6, Receiver – pin 7, Receiver +	pin 1, Receiver – pin 2, Receiver +
	7	 +5V 2,7 kΩ E	 +5V 7,5 kΩ 50 kΩ
		TTY SERIAL IN	TTY SERIAL IN
	4	±15 V	±12 V
	7	pin 48 +15 V pin 49 –15 V	pin 48 +12 V pin 49 –12 V
	9	±15 V	±12 V
	13	±15 V	±12 V
	15	pin 48 +15 V pin 49 –15 V pin \bar{d} +15 V pin \bar{e} –15 V	pin 48 +12 V pin 49 –12 V pin \bar{d} +12 V pin \bar{e} –12 V
SP51	5, TABLE 2	ABUS 9 ABUS 10	ABUS 10 ABUS 9
SS50	6, line 200 8, line 294	75 10 CPSL RS 12 SPSU	75 18 CPSL RS + WC 92 LPSU

2650 APPLICATION REPORTS

As an *extra* service to you copies of the Philips Application Laboratory Reports listed below can be ordered. These reports are issued as a basis for the Application Notes to be sent to you free of charge later. However, if you think that the information is of immediate use for your current application, a copy can be ordered from your local Philips Organization.

No.	Title	Summary
✕ EDP 7519	Sorting Routines	Describes methods and programs for sorting single byte numbers, which are in a list in an incrementing order. Search and bubble sort methods are outlined.
✕ EDP 7706	Sorting Routines for Multiple Byte Numbers	Describes methods and programs for sorting signed and unsigned multiple byte numbers.
✕ MDP 7601	Look-up tables	Describes methods for programming look-up tables and gives suggestions on how to connect the hardware to implement the various uses of the tables.
✕ MDP 7604	Look-up and Search Routines for the 2650	This report completes MDP 7601 and describes specific routines for table look-up and search.
DPM 76103	Additional Facility for the PC1001/DS2000	A small modification of the PC1001/PC1500 prototyping card and some additional hardware are required to extend this prototyping system with a hardware break-point by setting the 15 lever switches to the required address.
✕ DPM 76104	A Software Method for Generating Cyclic Redundancy Check Characters	A flexible program is described which will generate sixteen check characters by performing CRC on any given data pattern. There is a choice of four code polynomials, but any other 16 degree polynomial can be adopted.
✕ EDP 7710	Keyboard Systems for the 2650 Microprocessor	Required hardware and software are described for keyboard systems with 8, 16, 32 and 63 keys.
PRR22-25-495	Memory Interface with the Signetics 2650 Microprocessor	Describes the hardware interface with a small size memory system with 1k8 fusible link PROM (82S115) and 256 bytes static RAM (2606-1 and 2612).
PRR22-25-491	Interfacing the Signetics 2650 Microprocessor with Memory in a Medium Size System	Describes the hardware interface with a medium size memory system with 2k8 fusible link PROM (82S115) and 4k8 static RAM (2602)
PRR22-25-493	Dynamic Memory Interface with the 2650 Microprocessor	Describes the hardware interface with a large size memory system with 4k8 fusible link PROM (82S115) and 16k8 Dynamic RAM (2680).
✕ EDP 7707	Interfacing a High Speed Reader and a High Speed Punch to the 2650	Describes hardware and software interface with the Digitronics Model 2540 high speed reader and the Facit model 4070 high speed punch.
✕ EDP 7602	Binary Floating Point Arithmetic Routines for the 2650	Describes binary floating point arithmetic routines for the 2650.

No.	Title	Summary
EDP 7603	BCD Floating Point Arithmetic Routines for the 2650	Describes BCD floating point arithmetic routines for the 2650.
EDP 7604	General Protocol for Data Exchange between a 2650 Microcomputer and Peripheral Devices	Describes general rules for the software and hardware interface for a 2650 microcomputer and peripheral devices.
EDP 7605	Interfacing a High Speed Punch and Reader to a 2650 Based System	Describes hardware and software interfaces with the Digitronics model 2540 high speed reader and the Facit model 4070 high speed punch. (Based on the protocol as described in EDP 7604.)
EDP 7606	Interfacing a CRT Display to a 2650 Microprocessor	Describes both hardware and software interface between a 2650 based microcomputer and a CRT display. (Based on the protocol as described in EDP 7604.)
EDP 7607	Seven Segment LED Display Drive with the 2650	Describes the conversion of hex or BCD characters to 7-segment code and the interface to single and multiple 7-segment LED displays.
EDP 7609	Software Package for a Diagnostic Memory Test in a 2650 Microcomputer	Describes a memory test routine able to diagnose and locate faults in the 2650 microcomputer RAM.
EDP 7612	Logarithmic Routine for the 2650 Microprocessor	Describes a routine for calculating the natural logarithm (ln) of BCD floating point numbers using the CORDIC algorithm.
EDP 7611	A Digital Cassette Recorder Interface for a 2650 Microprocessor Based System	Hardware and software interface is described to connect a digital cassette recorder to the 2650. Phase encoding according to ECMA 34 standard is used. Error detection is performed with a CRC subroutine.

MICROPROCESSOR PUBLICATIONS

The list of technical literature published by Signetics in support of its MOS and bipolar microprocessor product offerings continues to grow. Included in this list are application memos, data sheets, brochures, and technical manuals.

To obtain a copy of any of the publications listed below, please contact your local Philips Organization.

TECHNICAL MANUALS

2650 Microprocessor Manual (bound manual) — Contains the complete specifications for the 2650 microprocessor. Describes the instruction set, interface signals, the internal organization, and the electrical characteristics. Includes user guides to the 2650 Assembler Language and the 2650 Simulator.

2650 Registered Microprocessor Manual Set (loose-leaf) — Same as above with the addition of all Signetics microprocessor application memos with automatic updating service. Order No. 2650BM1000.

Signetics TWIN 2650 Assembly Language Manual — A user's guide to the 2650 Assembly Language for the TWIN Prototype Development System. Order No. TW09005000.

TWIN Operator's Guide — Describes all aspects of TWIN system operation, from unpacking, through switches and indicators, to the use of the various system development programs. Order No. TW09003000.

TWIN System Reference Manual — Describes each board in the TWIN system, with functional descriptions and a theory of operation at the block diagram level. A knowledge of microcomputer development systems and the 2650 microprocessor is assumed. Order No. TW09004000.

Designing with Microcomputers — An introductory text on microcomputer fundamentals for electronic circuit and system designers and managers.

DATA SHEETS

- TWIN Microcomputer Prototype Development System
- PC1001 Microprocessor Prototyping Card*
- DS2000 Microprocessor Demonstration System*
- PC2000 4K Memory Card*
- PC3000 Intelligent Typewriter Controller*
- KT9100 Microprocessor Prototyping Kit*
- AS1000/1100 2650 Assembler Version 3.2*
- SM1000/1100 2650 Simulator Version 1.2*
- PL1000 Signetics Higher Level Language (PL μ S)*
- PC1500/KT9500 Adaptable Board Computer (ABC) Prototyping System*
- 2651 Programmable Communications Interface (PCI) Integrated Circuit
- 2655 Programmable Peripheral Interface (PPI) Integrated Circuit

BROCHURES

- 2650 Introductory Brochure and Short Form Catalog
- Signetics TestWare Instrument (TWIN)

BIPOLAR MICROPROCESSORS

Contact your local Philips Organization for literature on our powerful 2- and 4-bit slices, the 3002 and 2901-1, the fast 8X300 8-bit fixed instruction set bipolar microprocessor and all the relevant support circuits.

* Included in 2650 Introductory Brochure and Short Form Catalog.

SIMPLE SUPPORT CIRCUITRY

The Signetics 2650 8-bit, n-channel microprocessor continues to gain wide acceptance throughout the industry as an easy to use but powerful micro-processor.

A completely static microcomputer system can be built with the 2650 microprocessor as its heart. You can easily interface logic circuits with the microprocessor since every input and output can handle one TTL load. And many of the multiple-sourced and support circuits can be connected without any extra interfacing — thus permitting you to design a low cost system.

The 2650 is a single-chip microprocessor made using ion-implanted, n-channel, silicon-gate process. It has a fixed command set of 75 instructions, operates on 8-bit parallel data and can address 32,768 bytes. A single +5 volt power supply and single-phase TTL clock are all you need to get the microprocessor up and running. All bus outputs of the 2650 are three-state and can drive either one 7400-type load, or four 74LS loads.

Both memory and input/output (I/O) lines operate asynchronously at any speed up to the maximum data transfer rate of the memory circuits without additional buffering. No external latching of data is needed.

Aside from the 40-pin microprocessor IC, there are many support circuits and development aids in the 2650 family. Some of the specialized interface circuits to be introduced include the 2651 Programmable Communication Interface (PCI), which accepts program instructions from the microprocessor and supports almost any serial-data communication mode. Another circuit, the 2655, is a Programmable Peripheral Interface (PPI) that contains three bidirectional 8-bit I/O ports and an 8-bit data bus to communicate with the processor.

Since all inputs and outputs of the 2650 are TTL compatible, standard logic circuits can be used for all interface requirements. The two specialized interface circuits mentioned earlier — the PCI and PPI — offer interfaces that are software alterable (rather than hardware alterable) for parallel and serial data applications.

The PCI (Model 2651) is a universal synchronous/asynchronous data communications controller that supports almost any serial-data communications link in full-duplex or half-duplex modes. It accepts serial data from a peripheral and converts it to parallel data for the 2650 and vice-versa. Inside the 2651 are a baud-rate generator, a modem controller, data-transmit and receive buffers and support control logic. The baud-rate generator has sixteen commonly used baud rates that are software selectable.

The transmitter and receiver sections of the 2651 can operate simultaneously and the baud-rate generator can accept external clocks or use its own internal clock for all timing. A 28-pin DIP houses the n-channel MOS device and only a 5 V supply is needed for circuit operation.

The PPI (Model 2655) contains three 8-bit quasi-bidirectional ports for I/O in a 40-pin DIP. All three ports are internally multiplexed to feed onto the 8-bit-wide bidirectional data bus of the 2650. Each port of the PPI can be software controlled to act as an input, output or bidirectional bus. The PPI can be programmed to function in five major operating modes: static, strobed, bidirectional, serial or serial/timer.

One port of the 2655 can act as a serial I/O. A 3-MHz programmable timer or event counter is also available on the serial port to aid in timing external events. All lines are TTL-compatible.

To use either of these circuits, just set up the control words in your program and load the program into the 2650 memory. You can even change the port's function in mid-program, depending upon your application.

HARD/SOFTWARE DEVELOPMENT AIDS

Signetics offers 2650 users several development aids for both hardware and software:

The 2650PC1001: A microprocessor prototyping card that contains a complete microcomputer on a single printed-circuit card. On the board are the 2650 micro-processor, a control and R/W memory, two I/O ports, a clock and all necessary buffering and interface circuits.

The 2650PC2000: A 4K byte memory card that is compatible with the PC1001. It contains 32, 21L02 1k x 1 static RAMs. Decoding is provided to select any block of 1k x 8 and to distinguish cards in a multcard system.

The 2650DS2000: This is a complete microprocessor demonstration system that can accept one PC1001 or PC1500 and one PC2000 or PC1600. It has a built-in power supply and serial interfaces for RS-232 and TTY inputs.

The 2650 KT9100: This is a microprocessor prototyping kit that contains the 2650 microprocessor and enough support circuits to permit the development of a small system.

The 2650PC1500/KT9500: The Adaptable Board Computer is a modular microcomputer that contains the microprocessor, memory, I/O ports and support circuitry. It also permits user-designed circuits to be directly wired on the board. Two forms of the ABC system are available: the PC1500 fully assembled version and the KT9500 kit.

The 2650PC1600: This is a resident assembler. It accepts a program written in 2650 Assembly Language as an input, and produces a paper tape containing a hexadecimal translation of the program. This hexadecimal tape has a format suitable for input to the PC1001 or ABC1500 prototyping boards, via the PIPBUG control program which is included on both these boards. The PC1600 also fits into the DS2000 Demo Base.

For software support and development debugging several different programs are available:

Assembler: The 2650 assembly language (PIPASM) is a symbolic language designed to simplify the writing of programs for the 2650. It is written in FORTRAN IV and is modular — it can be executed in an overlay mode if the processor memory can't handle the entire program. Two passes are used to generate the symbol table, issue error messages, produce a program listing and a computer-readable object listing. Two versions are available: the AS1000 for 32-bit machines and the AS1100 for 16-bit machines. Also available on TYMSHARE, GE, and NCSS timesharing services.

Simulator: The 2650 simulator (PIPSIM) is a FORTRAN IV program that may be used to simulate the execution of your program without using the 2650. PIPSIM maintains its own internal FORTRAN storage registers, to describe the 2650 program, its registers, the ROM/RAM configuration and input data. There are two versions available: the SM1000 for 32-bit machines and the SM1100 for 16-bit units. Also available on TYMSHARE, GE, and NCSS timesharing services.

Signetics Higher Level Language (PL μ S): A PL-type microprocessor programming language which the programmer uses to replace many lines of machine code with a single statement. The PL μ S compiler is available in 32-bit format and also on TYMSHARE and NCSS timesharing services.

TWIN PROTOTYPE DEVELOPMENT SYSTEM

TWIN, a powerful and unique prototyping and development system, was recently added to Signetics growing list of product offerings.

TWIN consists of interdependent subsystems, each contributing to the total task of implementing user microprocessor applications from initial concept to actual hardware operation. The system closely resembles a general-purpose minicomputer during the initial stages of product development, and allows source programs to be entered, edited and assembled into object programs. Object programs may be executed simply as programs, or as part of a user's product emulation. When the programs have been run and debugged to the user's satisfaction, the TWIN system is capable of programming PROM devices for inclusion in the user's prototype hardware.

Initially announced in Europe early last year, TWIN has gained acceptance as the industry standard for prototype development systems. Because of its dual-CPU architecture, TWIN is capable of supporting virtually any eight or sixteen-bit microprocessor, including all of Signetics main-thrust MOS and bipolar microprocessors. One CPU, called the "master", controls and supervises all system resources. The other, called the "slave", supports all user-defined development functions.

Since the "master" and "slave" need not be the same microprocessor, the TWIN system will never become obsolete. Only the slave CPU must match the user's selected microprocessor for design-in applications.

Prior to the emergence of TWIN, microprocessor users requiring a development system have had to purchase one with the certainty that if they decided to change or upgrade to a new processor, it would be necessary to purchase another development system.

A typical TWIN system consists of the program development computer, a CRT terminal for data entry and display, a dual-drive floppy disk unit for mass storage and initial program loading, and a TestWare In-Circuit Emulator cable, called TWICE, to connect the TWIN system to the user's prototype hardware. As options, additional disk drivers may be added and a line printer added for hard copy output of data. A TTY may also be used for data entry or hard copy output.

TWIN is provided with a full range of supporting software, including a disk-based operating system, a text editor, a resident assembler, and extensive debugging and diagnostic capabilities.

The TWIN PROM programming capability lets the user program his memory with the object programs created by the system in the earlier phase of development, thus simulating most of the final hardware. The Debug software package lets the user trace program execution, examining the contents of RAM at selected locations and checking CPU status and I/O operations. Thus the complete range of user needs is met, beginning with a user program on paper and ending with final execution in hardware.

The TWIN system comes in two configurations. Super TWIN is a fully configured microcomputer development system that incorporates a CRT, printer, floppy disk unit, TWICE cable, and the dual-CPU development computer. Basic TWIN is essentially the same system without a CRT terminal or printer.

System features include a dual memory expandable from 16K to 64K bytes, 16-bit address and instruction busses, RS-232 and current-loop interfaces with transmission speeds ranging from 110 to 1200 baud, the TWICE cable, and all system software and supporting documentation.

MICROPROCESSOR COURSES

A series of one, two and three-day courses have been arranged and will be given in Eindhoven, The Netherlands, and will cover all aspects of the Signetics 2650 MOS and 8X300 bipolar microprocessors. The course language will be English. Also contact your local Philips Organization for courses held locally in your own language.

1977 PROGRAM

April	5	— Introduction to Microcomputers
April	7	— Designing with Microprocessors
April	19,20	— 8X300 Intensive Workshop
April	26,27,28	— 2650 Intensive Workshop
May	3,4,5	— 2650 System Design Workshop
May	10,11,12	— PL μ S Course
May	23,24	— TWIN System User Course
June	7	— Introduction to Microcomputers
June	8	— Designing with Microprocessors
June	14,15,16	— 2650 Intensive Workshop

Our preliminary course program for the second half of this year is as follows:

August	16	— Introduction to Microcomputers
August	18	— Designing with Microprocessors
August	23,24,25	— 2650 Intensive Workshop
September	5,6,7	— 2650 System Design Workshop
September	14,15,16	— PL μ S Course
September	20,21	— 8X300 Intensive Workshop
September	22,23	— TWIN System User Course
October	4,5,6	— 2650 Intensive Workshop
October	19,20,21	— PL μ S Course
October	25,26	— 8X300 Intensive Workshop
October	27,28	— TWIN System User Course

DESCRIPTION OF THE COURSES

Course: Introduction to Microcomputers 1-day Course

This basic course is intended for those engineers, salesmen, and managers who are not familiar with logic design. As a background, the course presents the developments that have made microprocessors possible and focuses on the advantages of microprocessor-based design and the trade-offs between microprocessor-based solutions and the more conventional ones. The use of microprocessors from three different viewpoints — design, marketing and production — is described and the course reviews the fundamental concepts of the development cycle.

Course: Designing with Microprocessors 1-day Course

This course has a two-fold objective. That of familiarization of engineers and programmers with microprocessor fundamentals, and demonstrating the application of Signetics 2650 microprocessors to system design. A real-life design problem — an intelligent typewriter system — is posed and solved. This design example also serves to illustrate the important differences between microprocessor and random logic techniques, and illustrates the simplicity of using the Signetics 2650 microprocessor.

Course: 2650 Intensive Workshop 3-day Course

This intensive workshop of lectures and laboratory work is intended primarily for logic designers. Divided into several sections, the course describes the 2650 instruction repertoire including instruction formats and addressing, the software development cycle, interface requirements and the design of interface circuits. The objective of this course is to provide participants with the knowledge and experience necessary to apply the 2650 microprocessor to the solution of real-life design problems. Accordingly, practical work also assumes a major role in this course.

Course: 2650 System Design Workshop 3-day Course

For those who have completed the three-day intensive workshop, or for those familiar with the 2650 but lack design experience, the workshop offers mainly practical work. Problem solving with a microcomputer system, standard hardware interfacing methods, hardware system design and the program development are all included. This is a course only for those with a good 2650 background, but lack experience in tackling a design problem. Taking both courses gives complete capability to produce one's own system designs.

Course: TWIN System User Course 2-day Course

This course is principally for engineers and programmers familiar with the 2650 microprocessor and its instruction repertoire. The course develops a practical understanding of Signetics TWIN Prototype Development System. This system includes a development computer, dual floppy disk unit, display terminal with keyboard, high speed printer and in-circuit emulator TWICE. Laboratory work enables participants to execute a hardware/software development cycle.

Course: 8X300 Intensive Workshop
2-day Course

This course is intended for users of the bipolar 8X300 microprocessor and provides a theoretical and practical background to 8X300 hardware, software and interface circuits. Each participant has the personal use of an 8X300 system development computer for course work. Participants can gain the knowledge necessary for using the 8X300 to solve real-life design problems.

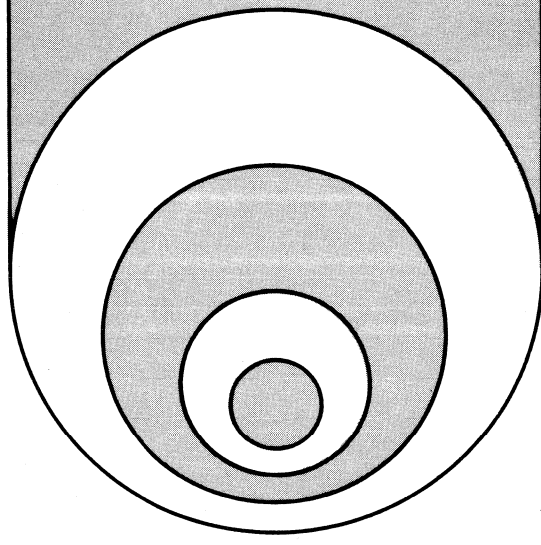
Course: PL μ S Course
3-day Course

The PL μ S course (Programming Language for Micro Systems) is an introduction to high level language programming for hardware-oriented designers. Trade-offs between high level and assembly languages are discussed, including basic concepts of high level language programming. High level language enables a designer to develop machine language code with less effort and fewer statements than with assembly language.

Due to the large amount of interest in our training courses, an early enrolment through your Philips Organization is strongly advised.

signetics

MICROPROCESSOR



SERIAL INPUT/OUTPUT.....AS50

2650 MICROPROCESSOR APPLICATION MEMO

INTRODUCTION

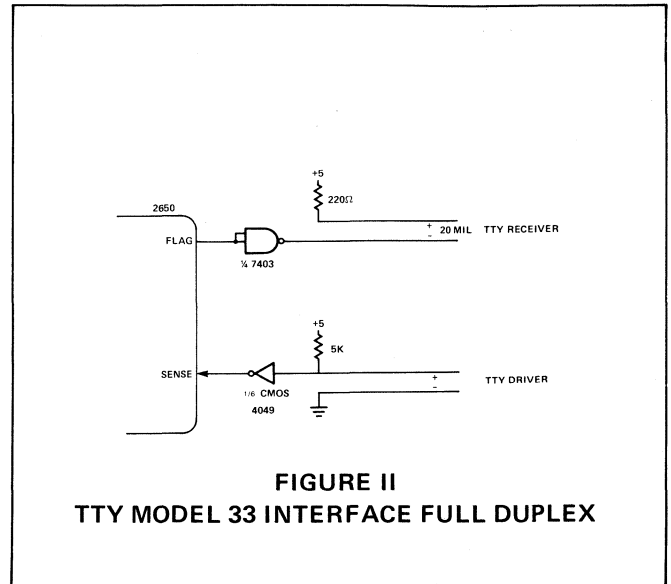
The Sense/Flag capability of the Signetics 2650 microprocessor can be used for serial I/O interfaces. The Sense input pin is directly connected to a bit in the microprocessor's Program Status Word. A high level on the Sense pin appears as a binary one while a low level appears as a binary zero. The Sense bit in the PSW can be stored or tested by the program. The Flag bit in the PSW is a simple latch that drives the Flag output pin. A program can set the Flag bit to a binary one, which causes a high level, one TTL load on the flag output pin. Setting the Flag bit to binary zero causes a low level on the Flag output pin.

APPLICATIONS

The most common use for the Sense/Flag capability would be in interfacing to a keyboard based terminal where the data is received or transmitted as bit serial. All bit manipulation and timings such as 8-bit serial-to-parallel conversion can be done by software running on the 2650. The software works by storing or setting the two bits in the Program Status Word which reflect or control the levels at the pins of the chip. External hardware is required simply to interface with line levels. No clock synchronization or address decoding hardware is necessary, since the Sense and Flag pins are independent of the normal I/O bus structure.

Two examples of device interfaces and software are given below; for a 1200 baud RS232-type CRT terminal and for a 110 baud Teletype. Figure 1 shows the RS232 interface. Half of the Signetics 8T15 dual line driver is used to transmit to the terminal from the Flag pin, while half of a

Signetics 8T16 dual line receiver is used to receive from the device into the Sense pin. The interface to a Teletype model 33 is shown in Figure II. A TTL open collector gate is used to provide the 20 milli-amp loop to the TTY



**FIGURE II
TTY MODEL 33 INTERFACE FULL DUPLEX**

receiver. For receiving from the TTY a CMOS gate is used to provide the necessary noise immunity.

SOFTWARE

All definitions of baud rate, character formats, and line characteristics are done in the software. For these examples, communication is asynchronous bit-serial over a full duplex line. Figure III shows the format of a 8-bit character (seven bits plus parity) headed by a start bit and followed by stop bits. The line levels are:

- low = start bit or binary zero
- high = stop bit or binary one

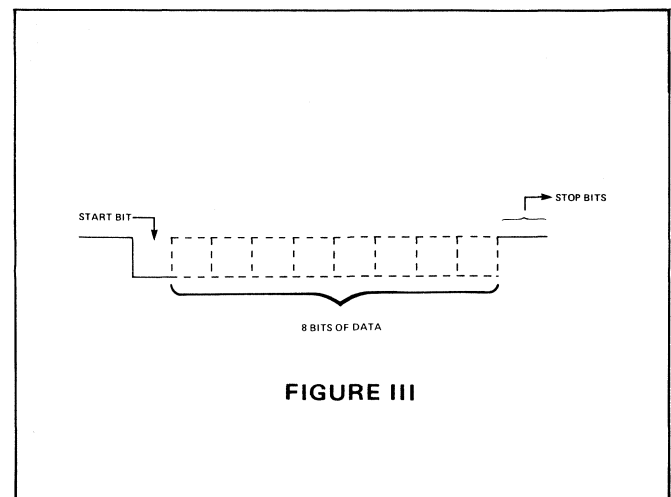
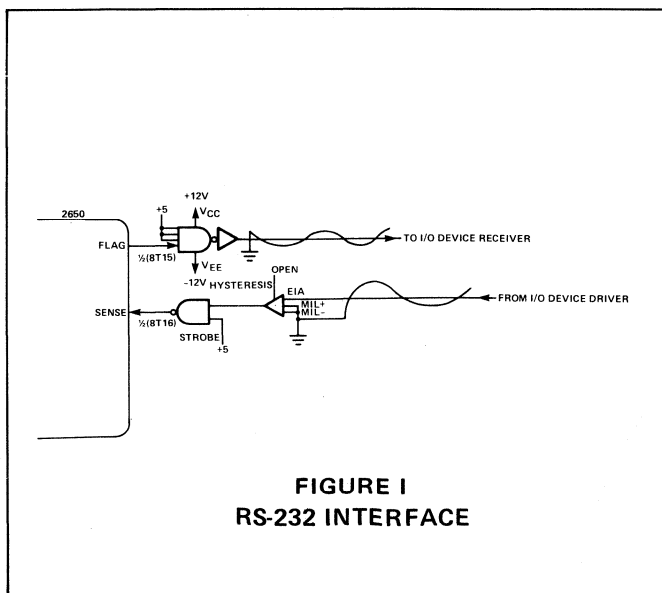


FIGURE III



**FIGURE I
RS-232 INTERFACE**

```

PIP ASSEMBLER VERSION 2 LEVEL 1
PAGE 1
LINE ADDR LABL B1 B2 B3 B4 ERRUR SOURCE
1 0001 EQU 1
2 0002 EQU 2
3 0003 EQU 3
4 0004 EQU 4
5 0005 EQU 5
6 0006 EQU 6
7 0007 EQU 7
8 0008 EQU 8
9 0009 EQU 9
10 0010 EQU 10
11 0011 EQU 11
12 0012 EQU 12
13 0013 EQU 13
14 0014 EQU 14
15 0015 EQU 15
16 0016 EQU 16
17 0017 EQU 17
18 0018 EQU 18
19 0019 EQU 19
20 0020 EQU 20
21 0021 EQU 21
22 0022 EQU 22
23 0023 EQU 23
24 0024 EQU 24
25 0025 EQU 25
26 0026 EQU 26
27 0027 EQU 27
28 0028 EQU 28
29 0029 EQU 29
30 0030 EQU 30
31 0031 EQU 31
32 0032 EQU 32
33 0033 EQU 33
34 0034 EQU 34
35 0035 EQU 35
36 0036 EQU 36
37 0037 EQU 37
38 0038 EQU 38
39 0039 EQU 39
40 0040 EQU 40
41 0041 EQU 41
42 0042 EQU 42
43 0043 EQU 43
44 0044 EQU 44
45 0045 EQU 45
46 0046 EQU 46
47 0047 EQU 47
48 0048 EQU 48
49 0049 EQU 49
50 0050 EQU 50
51 0051 EQU 51
52 0052 EQU 52
53 0500 0500 76 40 CH10 PPSU
54 0502 0502 75 08 CH10 CPSU
55 0504 0504 05 08 CH10 LODI,R1
56 0506 0506 06 08 CH10 LODI,R2
57 0508 0508 1A 7D TEST BCTR,LT
58 0509 0509 3F 05 29 TEST DLY
59 0510 0510 3F 05 2D BIT BSTA,UN
60 0511 0511 12 80 BIT CPSU
61 0512 0512 44 80 BIT ANDI,R0
62 0513 0513 51 80 BIT RRR,R1
63 0514 0514 01 80 BIT IOH2
64 0515 0515 01 80 BIT SYN2
65 0516 0516 01 80 BIT R1
66 0517 0517 01 80 BIT R1
67 0518 0518 01 80 BIT SYN2
68 0519 0519 74 40 BIT BCTR,LT
69 0520 0520 74 40 BIT CPSU
70 0521 0521 1B 02 BIT BCTR,UN
71 0522 0522 76 40 BIT PPSU
72 0523 0523 76 40 BIT PPSU
73 0524 0524 04 59 BIT BURR,R2
74 0525 0525 3F 05 2D BIT BSTA,UN
75 0526 0526 45 7F BIT ANDI,R1
76 0527 0527 17 7F BIT RETC,UN
77 0528 0528 17 7F BIT END
78 0529 0529 04 3A * TIMING DELAY FOR 1200 BAUD RS232 TERMINAL
79 0530 0530 1B 02 DLY LODI,R0
80 0531 0531 08 02 DLY BCTR,UN
81 0532 0532 08 02 DLY LODI,R0
82 0533 0533 08 02 DLY BURR,R0
83 0534 0534 08 02 DLY BURR,R0
84 0535 0535 08 02 DLY BURR,R0
85 0536 0536 08 02 DLY BURR,R0
86 0537 0537 08 02 DLY BURR,R0
87 0538 0538 08 02 DLY BURR,R0
88 0539 0539 08 02 DLY BURR,R0
89 0540 0540 08 02 DLY BURR,R0
90 0541 0541 08 02 DLY BURR,R0
91 0542 0542 08 02 DLY BURR,R0
92 0543 0543 08 02 DLY BURR,R0
93 0544 0544 08 02 DLY BURR,R0
94 0545 0545 08 02 DLY BURR,R0
95 0546 0546 08 02 DLY BURR,R0
96 0547 0547 08 02 DLY BURR,R0
97 0548 0548 08 02 DLY BURR,R0
98 0549 0549 08 02 DLY BURR,R0
99 0550 0550 08 02 DLY BURR,R0
100 0551 0551 08 02 DLY BURR,R0

```

```

PIP ASSEMBLER VERSION 2 LEVEL 1
PAGE 2
LINE ADDR LABL B1 B2 B3 B4 ERRUR SOURCE
53 0532 0532 20 TDLA EUMZ R0
54 0533 0533 08 7E TDLY BURR,R0 $
55 0534 0534 08 7E TDLY BURR,R0 $
56 0535 0535 08 7E TDLY BURR,R0 $
57 0536 0536 08 7E TDLY BURR,R0 $
58 0537 0537 08 7E TDLY BURR,R0 $
59 0538 0538 08 7E TDLY BURR,R0 $
60 0539 0539 08 7E TDLY BURR,R0 $
61 0540 0540 08 7E TDLY BURR,R0 $
62 0541 0541 08 7E TDLY BURR,R0 $
63 0542 0542 08 7E TDLY BURR,R0 $
64 0543 0543 08 7E TDLY BURR,R0 $
65 0544 0544 08 7E TDLY BURR,R0 $
66 0545 0545 08 7E TDLY BURR,R0 $
67 0546 0546 08 7E TDLY BURR,R0 $
68 0547 0547 08 7E TDLY BURR,R0 $
69 0548 0548 08 7E TDLY BURR,R0 $
70 0549 0549 08 7E TDLY BURR,R0 $
71 0550 0550 08 7E TDLY BURR,R0 $
72 0551 0551 08 7E TDLY BURR,R0 $
73 0552 0552 08 7E TDLY BURR,R0 $
74 0553 0553 08 7E TDLY BURR,R0 $
75 0554 0554 08 7E TDLY BURR,R0 $
76 0555 0555 08 7E TDLY BURR,R0 $
77 0556 0556 08 7E TDLY BURR,R0 $
78 0557 0557 08 7E TDLY BURR,R0 $
79 0558 0558 08 7E TDLY BURR,R0 $
80 0559 0559 08 7E TDLY BURR,R0 $
81 0560 0560 08 7E TDLY BURR,R0 $
82 0561 0561 08 7E TDLY BURR,R0 $
83 0562 0562 08 7E TDLY BURR,R0 $
84 0563 0563 08 7E TDLY BURR,R0 $
85 0564 0564 08 7E TDLY BURR,R0 $
86 0565 0565 08 7E TDLY BURR,R0 $
87 0566 0566 08 7E TDLY BURR,R0 $
88 0567 0567 08 7E TDLY BURR,R0 $
89 0568 0568 08 7E TDLY BURR,R0 $
90 0569 0569 08 7E TDLY BURR,R0 $
91 0570 0570 08 7E TDLY BURR,R0 $
92 0571 0571 08 7E TDLY BURR,R0 $
93 0572 0572 08 7E TDLY BURR,R0 $
94 0573 0573 08 7E TDLY BURR,R0 $
95 0574 0574 08 7E TDLY BURR,R0 $
96 0575 0575 08 7E TDLY BURR,R0 $
97 0576 0576 08 7E TDLY BURR,R0 $
98 0577 0577 08 7E TDLY BURR,R0 $
99 0578 0578 08 7E TDLY BURR,R0 $
100 0579 0579 08 7E TDLY BURR,R0 $

```

TOTAL ASSEMBLER ERRORS = 0

FIGURE IV

The internal logic of the program shown in Figure IV (the program listing) is to sense each incoming bit of the character and to output the bit in turn for the full duplex line. The Sense input is tested in the loop at 'TEST' for the transition to zero indicating the start bit. The program then delays one half of a bit time to the center of the start bit. At this point the echoing of the character starts by clearing the Flag bit which outputs the start bit transition. At 'BIT' the program then delays one full bit time to the center of the data bit. The Sense line is tested and that bit value is rotated into register one. The bit value is then used to set or clear the Flag bit for the echo. At 'NEXT' is the test

that controls the loop to get only eight bits. Figure V is a picture of the levels and timings when echoing a 'U'.

The bit timing is done by a subroutine which simply counts cycles for the appropriate baud rate. The example program shows both a 1200 baud delay at 'DLAY' and a 110 baud delay at 'TLAY'. The conversion from instruction cycles to milliseconds is based on a 1MHz clock rate. Clock stability is only moderately important since each character involves only nine sample times and each start bit redefines the base line for all timings.

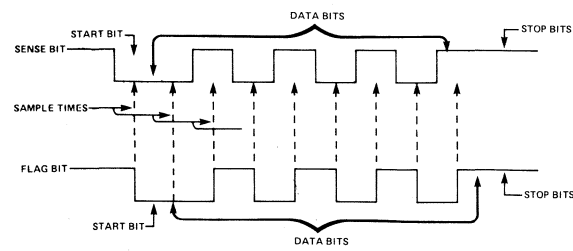
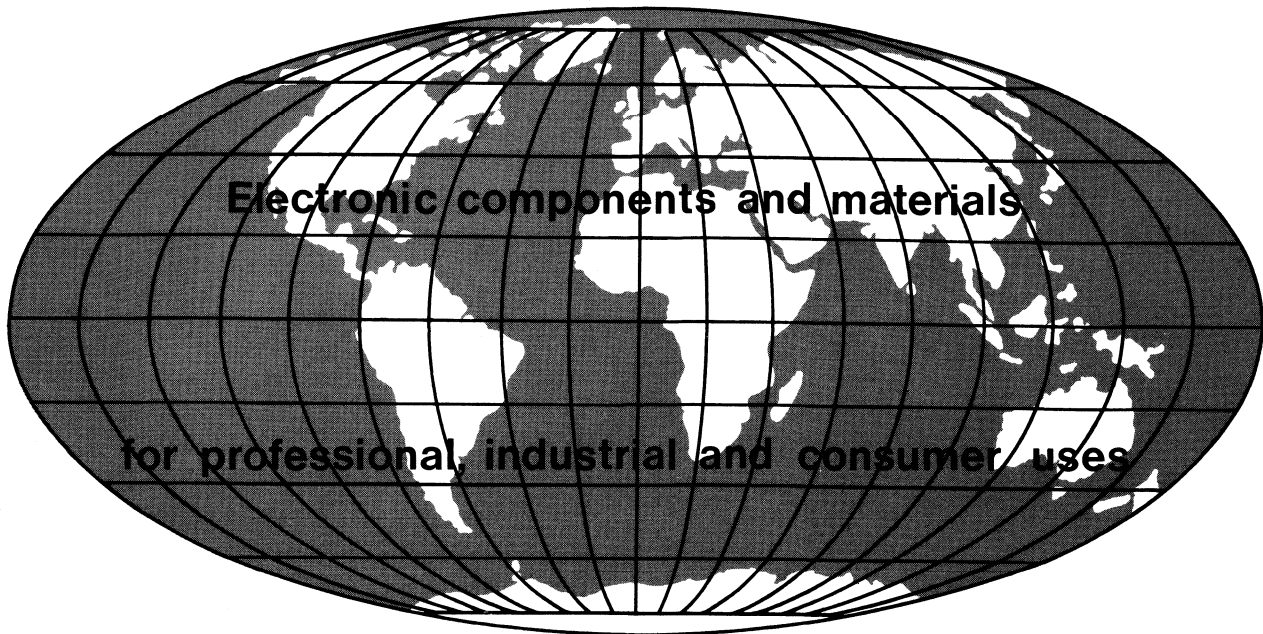


FIGURE V



from the world-wide Philips Group of Companies

EUROPEAN SALES OFFICES

Austria: Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.

Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.

Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.

Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.

France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.

Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.

Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.

Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.

Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.

Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.

Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.

Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.

Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.

Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.

Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.

Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoğlu, İSTANBUL.

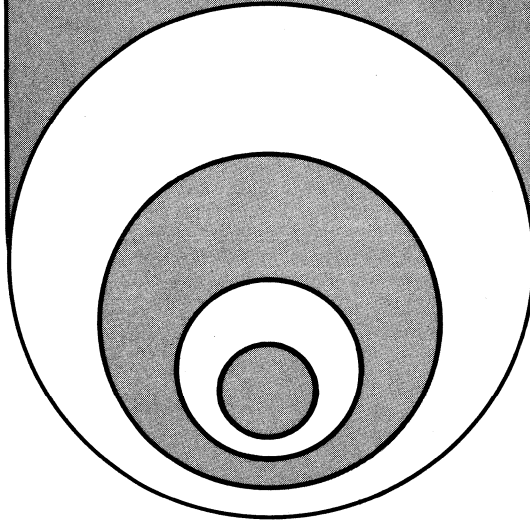
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

© N.V. Philips' Gloeilampenfabrieken

This information is furnished for guidance, and with no guarantees as to its accuracy or completeness; its publication conveys no licence under any patent or other right, nor does the publisher assume liability for any consequence of its use; specifications and availability of goods mentioned in it are subject to change without notice; it is not to be reproduced in any way, in whole or in part, without the written consent of the publisher.

signetics

**MOS
MICROPROCESSOR**



**BIT AND BYTE
TESTING
PROCEDURES
AS51**

SUMMARY

This applications memo describes several methods of testing the contents of the internal registers in the Signetics 2650 Microprocessor.

The following test examples are given:

- Specific bit(s) in a register.
- Positive, negative, or zero-contents of a register.
- Contents of a register compared with a value (equals, greater than, or less than).
- Interdigit-carry (IDC), overflow (OVF), and carry (C) flags in the program status word.

INTRODUCTION

As a result of an operation on register(s) of the 2650 register bank, five bits (bits 7, 6, 5, 2, and 0) in the Program Status Lower (PSL) portion of the Program Status Word (PSW) register can be affected.

7	6	5	4	3	2	1	0
CC1	CC0	IDC	RS	WC	OVF	COM	C

PROGRAM STATUS LOWER (PSL)

These bits are affected as follows:

CC1, CC0: Condition Code Bits

CONDITION CODE		RESULT OF		
		LOAD/STORE, ARITHMETIC, LOGICAL INSTRUCTIONS	COMPARE INSTRUCTION	SELECTIVE TESTS ON BITS (TMI, TPSU, & TPSL)
CC1	CC0			
0	0	Zero	Equal	All bits 1
0	1	Positive	Greater Than	----
1	0	Negative	Less Than	Not all bits 1

IDC: Interdigit Carry/Borrow Bit

The IDC bit is affected by arithmetic operations as well as rotation.

- 0 = Interdigit borrow/no interdigit carry
- 1 = Interdigit carry/no interdigit borrow

OVF: Overflow Bit. Arithmetic Operation

The overflow bit in arithmetic operations is set as follows:

$$\text{Operand 1} \pm \text{Operand 2} \rightarrow \text{Result}$$

SIGN			ADD OVF	SUB OVF
OPERAND 1	OPERAND 2	RESULT		
+	+	+	0	0
+	+	-	1	0
+	-	+	0	0
+	-	-	0	1
-	+	+	0	1
-	+	-	0	0
-	-	+	1	0
-	-	-	0	0

OVF: Overflow Bit. Rotate Operation

Condition: WC = 1; if WC = 0, the OVF bit is not affected.

The overflow bit is set as follows:

OPERAND SIGN		OVF
BEFORE ROTATE	AFTER ROTATE	
+	+	0
+	-	1
-	+	0
-	-	0

C: Carry/Borrow Bit

The Carry bit is affected by arithmetic operations as well as rotation.

- 0 = borrow/no carry
- 1 = carry/no borrow

BIT TESTING PROCEDURES

The bits of a register Rx (register zero Ro or any register R1, R2 or R3 in the selected register bank) can be tested as follows:

		BYTES	CYCLES
TEST FOR '0' IN BIT 3 OF Rx			
TMI, Rx	H'08'	1)	2 3
BCTR, 2	LBL *Branch if bit 3 is zero.		2 3
			4 6
or:			
ANDI, Rx	H'08'	2)	2 2
BCTR, 0	LBL *Branch if bit 3 is zero.		2 3
			4 5

While the second test is faster, it affects the contents of Rx.

BIT TESTING PROCEDURES (Continued)

TEST FOR '1' IN BIT 3 OF Rx

TMI, Rx	H'08'	1)	2	3
BCTR, 0	LBL	*Branch if bit 3 is one.	2	3
			4	6

or:

ANDI, Rx	H'08'	2)	2	2
BCTR, 0	LBL	*Branch if bit 3 is one.	2	3
			4	5

While the second test is faster, it affects the contents of Rx.

TEST FOR '0' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

TMI, Rx	H'4A'	1)	2	3
BCTR, 2	LBL	*Branch if one of the tested bits is zero.	2	3
			4	6

TEST FOR '1' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

ANDI, Rx	H'4A'	2)	2	2
BCTR, 0	LBL	*Branch if one of the tested bits is one.	2	3
			4	5

TEST FOR '0' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

ANDI, Rx	H'4A'	2)	2	2
BCTR, 0	LBL	*Branch if all tested bits are zero.	2	3
			4	5

TEST FOR '1' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

TMI, Rx	H'4A'	1)	2	3
BCTR, 0	LBL	*Branch if all tested bits are one.	2	3
			4	6

TEST FOR PATTERN IN Rx; e.g., x10xx01x

x = don't care

IORI, Rx	H'99'	2)	2	2
COMI, Rx	H'DB'		2	2
BCTR, 0	LBL	*Branch if pattern is true.	2	3
			6	7

- 1) Contents of register Rx kept
- 2) Contents of register Rx lost

BYTE TESTING PROCEDURES

TEST FOR POSITIVE, NEGATIVE AND ZERO

All of the tests described below must be preceded by an operation on Rx which updates the contents of the condition register, e.g., by instructions such as LOAD, ADD, AND, COMPARE, ROTATE, I/O, etc.

	CC	OPERATION
Test for (Rx) ≥ 0	00 or 01	BCTR, 2
Test for (Rx) > 0	01	BCTR, 1
Test for (Rx) = 0	00	BCTR, 0
Test for (Rx) < 0	10	BCTR, 2
Test for (Rx) ≤ 0	00 or 10	BCTR, 1

TESTS ON THE CONTENTS OF A REGISTER BY USING COMPARE INSTRUCTIONS

Logical compare: (COM = 1 in PSL)

Comparison is made between two 8-bit unsigned binary numbers.

Arithmetic compare: (COM = 0 in PSL)

Comparison is made between two 8-bit signed numbers.

After execution of the logic or arithmetic compare instruction, the condition register (CC) is set to a specific value and tested as follows:

REGISTER-TO-REGISTER COMPARE		
Instruction used: COMZ Rx		
RESULT	CC	TEST
(Ro) ≥ (Rx)	00 or 01	BCTR, 2
(Ro) > (Rx)	01	BCTR, 1
(Ro) = (Rx)	00	BCTR, 0
(Ro) < (Rx)	10	BCTR, 2
(Ro) ≤ (Rx)	00 or 10	BCTR, 1

REGISTER TO CONSTANT OR MEMORY LOCATION		
Instructions used: COMI, Rx DATA COMR, Rx RELATIVE LOCATION OF DATA COMA, Rx LOCATION OF DATA		
RESULT V=VALUE	CC	TEST
(Rx) ≥ V	00 or 01	BCTR, 2
(Rx) > V	01	BCTR, 1
(Rx) = V	00	BCTR, 0
(Rx) < V	10	BCTR, 2
(Rx) ≤ V	00 or 10	BCTR, 1

Whenever a compare instruction is used, the IDC, OVF, or C bits in the PSL are *not* affected.

TEST ON OVERFLOW (OVF in PSL)

The overflow bit is affected whenever arithmetic or rotate instructions are executed.

The *OVF bit* is set during an addition whenever the two operands have the same sign and the result has a different sign. During a subtraction, the *OVF bit* is set when the operands differ in sign and the result has a different sign than the first operand.

Examples: (+A) + (+B) = (-C) OVF
 (-A) + (-B) = (+C) OVF
 (+A) - (-B) = (-C) OVF
 (-A) - (+B) = (+C) OVF

Test: TPSL H'04' *OVF test
 BCTR, 0 LBL *Branch if OVF = set

The *OVF bit* is set during rotate instructions with WC = 1 whenever the sign changes from positive to negative. If WC = 0, then rotate instructions do not affect the OVF bit.

Example:

RRR, Rx *Rotate right
 TPSL H'04' *Test OVF bit
 BCTR, 0 LBL *Branch if OVF = set

TEST ON CARRY (C in PSL)

The carry bit is set to 1 by an add instruction that generates a carry and a sub-instruction that does *not* generate a borrow.

Example:

ADDITION

LODI, Rx H'88'
 ADDI, Rx H'99'
 TPSL H'01' *Test carry
 BCTR, 0 LBL *Branch if carry

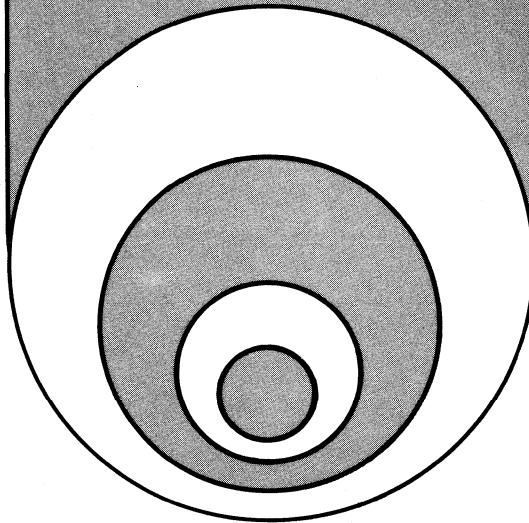
SUBTRACTION

LODI, Rx H'40'
 SUBI, Rx H'30'
 TPSL H'01' *Test borrow
 BCTR, 0 LBL *Branch if *no* borrow

When a rotate instruction is executed with WC = 1, the carry bit is also affected. Refer to the Signetics 2650 Microprocessor manual for a description of this operation.

signoties

**MOS
MICROPROCESSOR**



**GENERAL
DELAY
ROUTINES
AS52**

SUMMARY

In microprocessing applications, delay times are often required. A typical example is a delay time for a serial Teletypewriter interface. While delay times can be generated by counters, monostables, multivibrators, and other hardware, it is often simpler and more economical to use a short software routine.

This applications memo describes several ways of writing software delay time routines for the Signetics 2650 microprocessor. Time restrictions and formulas for calculating the delay time are given for each routine.

DELAY ROUTINES

In general, a delay can be implemented by setting a counter with a number N and decrementing this number by one until it is zero. If decrementing the number takes one clock period, then the total delay time is N clock periods.

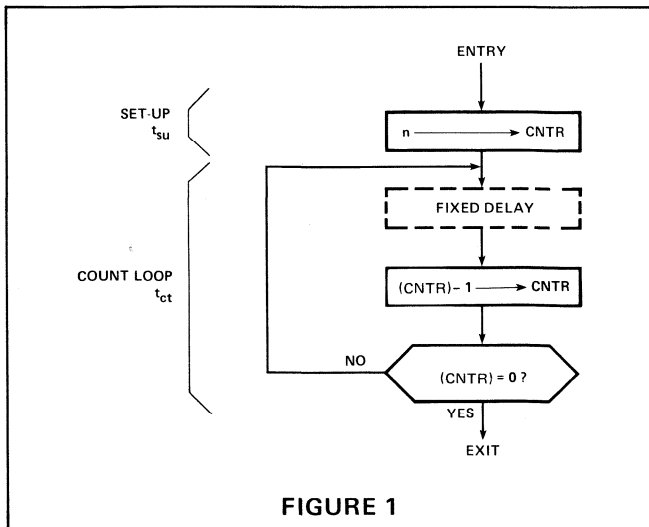
In the 2650 microprocessor, the internal registers may be used as counters. The most useful instructions for decrementing are the "Branch on Decrementing Register" (BDRR and BDRA) instructions, which also test the content of a register for zero.

Figure 1 illustrates a flowchart of a delay routine. This routine consists of a setup part and a count loop. The count loop will be executed n times and the setup only once. Hence, the delay time is:

$$t_d = t_{su} + n \cdot t_{ct}$$

It is possible to increase the delay time by increasing n or by making t_{ct} longer. The latter can be done by inserting a fixed delay such as a No Operation (NOP) instruction in the count loop.

DELAY ROUTINE FLOWCHART



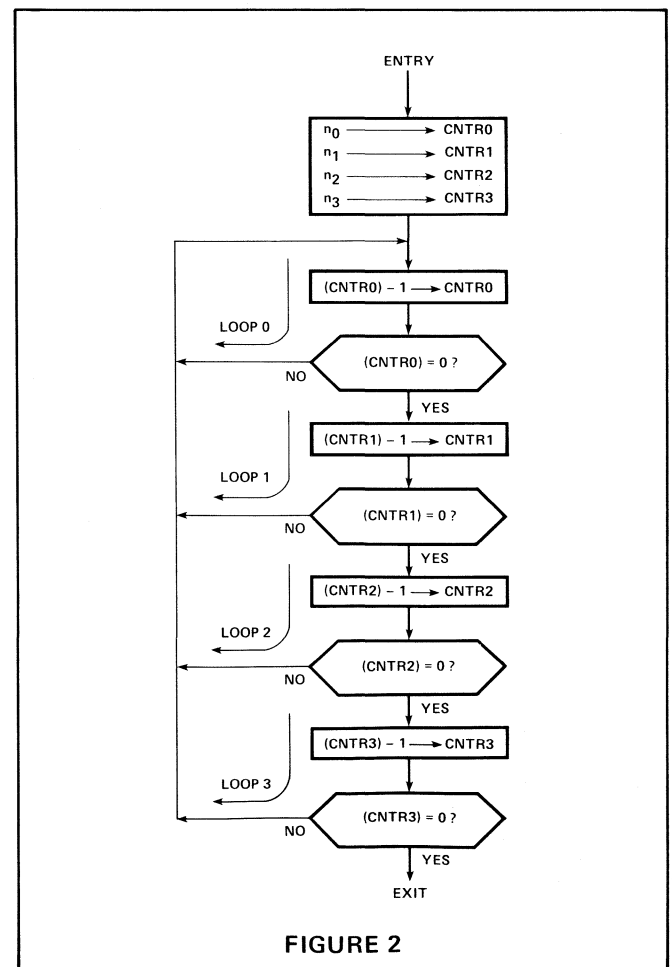
The program of the routine shown in Figure 1 is as follows:

LODI, Rx n	Load n into register Rx	6 cp*
LOOP NOP	No operation; fixed delay of 6 cp	6 cp
BDRR, Rx LOOP	Decrement Rx; branch to loop if the result is not zero	9 cp

*cp = clock periods

With one NOP, the delay time is: $t_d = (6 + 15 \cdot n)$ cp. Without the NOP, the delay time is: $t_d = (6 + 9 \cdot n)$ cp. The maximum delay time is obtained when Rx is loaded with zero, since Rx will cycle through all the 256 possible states. When Rx = R0, the LODI, R0 0 instruction can be replaced by the EORZ R0 instruction, which saves one byte of code.

DELAY ROUTINE WITH FOUR REGISTERS



GENERAL DELAY ROUTINES ■ AS52

Another possible way of increasing the delay time is to repeat the count loop of Figure 1 several times. This can be done by repeating the instructions or by counting the repetitions of the count loop in another register. For example, this latter method can be expanded to include four internal registers. A flowchart of a delay routine using this technique is illustrated in Figure 2.

The number of times the processor executes the different loops shown in Figure 2 are:

$$\begin{aligned} \text{loop 3} & n_3 \\ \text{loop 2} & n_2 + (n_3 - 1) 256 \\ \text{loop 1} & n_1 + (n_2 - 1) 256 + (n_3 - 1) 256^2 \\ \text{loop 0} & n_0 + (n_1 - 1) 256 + (n_2 - 1) 256^2 + (n_3 - 1) 256^3 \end{aligned}$$

Hence, the delay time of this routine is:

$$t_d = [24 + \{n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256 + 256^2) + n_3 + (n_3 - 1) (256 + 256^2 + 256^3)\} 9] \text{ cp}$$

(If Rx is loaded with a zero, then $n = 256$ in the formula):

Table 1 shows six different delay routine programs along with specifications for each program. The delay time for these routines can be computed from the following equations.

Routine	Delay Time
a	$t_d = (6 + 9 \cdot n_0) \text{ cp}$
b	$t_d = (6 + 15 \cdot n_0) \text{ cp}$
c	$t_d = (2310 + 9 \cdot n_0) \text{ cp}$
d	$t_d = \{12 + [n_0 + n_1 + (n_1 - 1) 256] 9\} \text{ cp}$
e	$t_d = \{18 + [n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256^2 + 256)] 9\} \text{ cp}$
f	$t_d = \{24 + [n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256^2 + 256) + n_3 + (n_3 - 1) (256^3 + 256^2 + 256)] 9\} \text{ cp}$

TABLE 1

ROUTINE	POSSIBLE DELAY TIME (cp)		DELAY STEP (cp)	NUMBER OF BYTES	NUMBER OF REGISTERS	PROGRAM
	MIN*	MAX				
a	15	2310	9	4	1	LODI, R0 n_0 LOOP BDRR, R0 LOOP
b	21	3846	15	5	1	LODI, R0 n_0 LOOP NOP BDRR, R0 LOOP
c	2319	4614	9	6	1	LODI, R0 n_0 LOP 1 BDRR, R0 LOP 1 LOP 2 BDRR, R0 LOP 2
d	30	592.140	9	8	2	LODI, R0 n_0 LODI, R1 n_1 LOOP BDRR, R0 LOOP BDRR, R1 LOOP
e	45	$\approx 151.6 \times 10^6$ **	9	12	3	LODI, R0 n_0 LODI, R1 n_1 LODI, R2 n_2 LOOP BDRR, R0 LOOP BDRR, R1 LOOP BDRR, R2 LOOP
f	60	$\approx 38.8 \times 10^9$ ***	9	16	4	LODI, R0 n_0 LODI, R1 n_1 LODI, R2 n_2 LODI, R3 n_3 LOOP BDRR, R0 LOOP BDRR, R1 LOOP BDRR, R2 LOOP BDRR, R3 LOOP

* cp = clock period. For 1MHz clock 1 cp = 1 μ s.

** For 1MHz clock this is about 2.5 minutes.

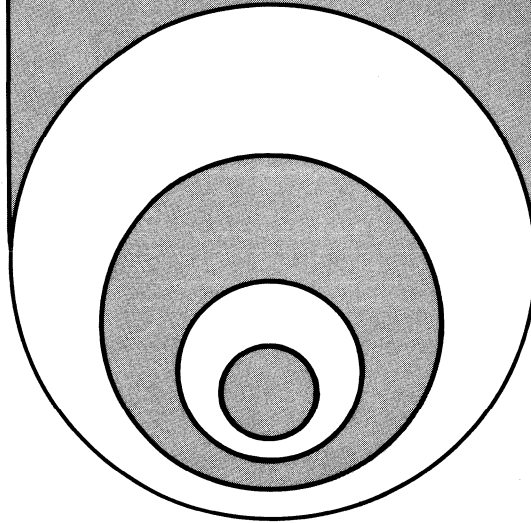
*** For 1MHz clock this is about 10.46 hours.

© N.V. Philips' Gloeilampenfabrieken

This information is furnished for guidance, and with no guarantees as to its accuracy or completeness; its publication conveys no licence under any patent or other right, nor does the publisher assume liability for any consequence of its use; specifications and availability of goods mentioned in it are subject to change without notice; it is not to be reproduced in any way, in whole or in part, without the written consent of the publisher.

signetics

**MOS
MICROPROCESSOR**



**BINARY ARITHMETIC
ROUTINES.....AS53**

2650 MICROPROCESSOR APPLICATIONS MEMO

INTRODUCTION

Binary arithmetic routines, like addition, subtraction, multiplication, and division, are often used in microprocessor-based systems. This applications memo provides several suggested examples for processing binary arithmetic routines on the 2650 microprocessor. These examples include:

- **SIGNED BINARY ADDITION/SUBTRACTION**
Two-byte operands giving a two-byte result.
- **UNSIGNED BINARY MULTIPLICATION**
One-byte operands giving a two-byte result.
Two-byte operands giving a four-byte result.
- **SIGNED BINARY MULTIPLICATION**
One-byte operands giving a two-byte result.
Two-byte operands giving a four-byte result.
- **BINARY DIVISION – UNSIGNED AND SIGNED**
Two-byte dividend and quotient with one-byte divisor and remainder.

In these examples, emphasis is placed on minimizing program memory requirements rather than on processing speed. The different branch instructions and the indexing features of the Signetics 2650 proved useful in minimizing memory requirements.

1. BINARY ADDITION/SUBTRACTION FOR TWO-BYTE SIGNED INTEGERS

FUNCTION:

Performs the addition or subtraction of two 2-byte signed integers giving a two-byte result.

$(OPR1, OPR1 + 1) +/- (OPR2, OPR2 + 1) \longrightarrow$
 $RSLT, RSLT + 1$

PARAMETERS:

Input: OPR1, OPR1 + 1 contains augend/subtrahend
OPR2, OPR2 + 1 contains addend/minuend
COM-flag in PSL indicates addition/subtraction:
COM = 0 addition
COM = 1 subtraction

Output: RSLT, RSLT + 1 contains sum/difference.
The condition code CC is set to the proper value of the two byte result.
OPR1, OPR2 and RSLT are MS-bytes.

SPECIAL REQUIREMENTS

None

Refer to Figures 1.1 and 1.2 for flowchart and program listing.

		HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'			
	X	X								
PSU	F	II	SP							
PSL	CC	IDC	RS	WC	OVF	COM	C			
	X	X		X	X		X			

RAM REQUIRED (BYTES): 6

ROM REQUIRED (BYTES): 45

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

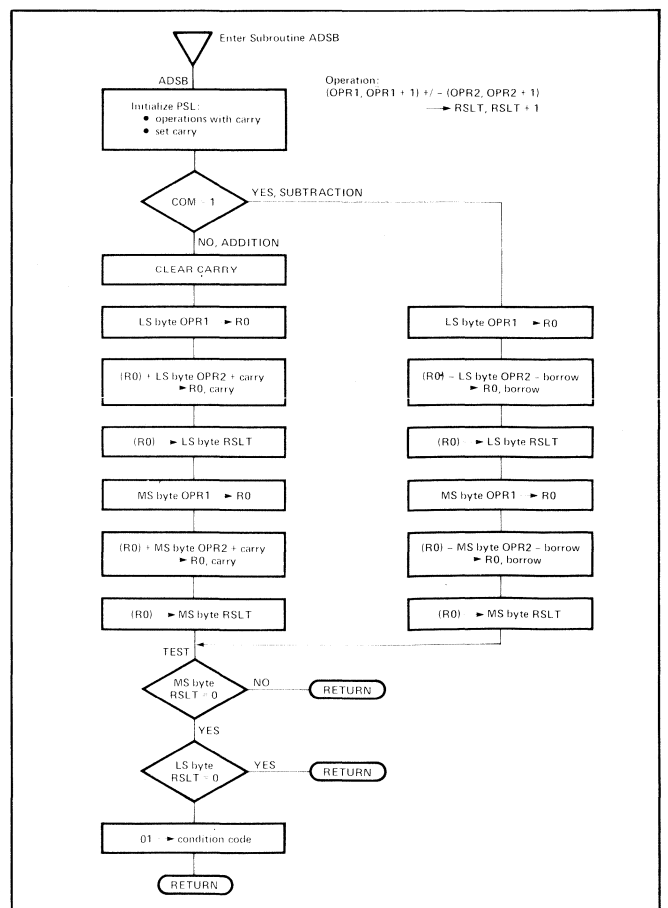


FIGURE 1-1 Flowchart for Double Precision Addition/Subtraction

```

1      * PD760010
2      *****
3      * BINARY DOUBLE PRECISION ADDITION/SUBTRACTION
4      *****
5      * OPERATION:
6      * (OPR1,OPR1+1)+/--(OPR2,OPR2+1)-->RSLT,RSLT+1
7      * OPR1,OPR2,RSLT ARE MOST SIG BYTES
8      * COM IN PSL IS USED AS ADD/SUB FLAG
9      *   COM=0 IS ADD; COM=1 IS SUBTRACT
10     * AFTER ADD/SUB THE CC,OVF,AND C BITS IN PSL
11     *   ARE VALID FOR THE RESULT
12     *
13     * DEFINITION OF SYMBOLS
14     *
15     0000      R0 EQU 0          PROCESSOR REGISTERS
16     0001      R1 EQU 1
17     0002      R2 EQU 2
18     0003      R3 EQU 3
19     0080      CC1 EQU H'00'     PSL: MSB OF CONDITION CODE
20     0040      CC0 EQU H'40'     LSB OF CONDITION CODE
21     0008      WC EQU H'08'     1=WITH,0=WITHOUT CARRY
22     0002      COM EQU H'02'    1=LOGICAL,0=ARITH COMP
23     0001      C EQU H'01'      CARRY/BORROW
24     0000      Z EQU 0          BRANCH COND: ZERO
25     0003      UN EQU 3         UNCONDITIONAL
26     0000      ON EQU 0         ALL BITS ARE 1
27     *
28     ORG      H'500'           START OF SUBROUTINE
29     *
30     0500 0500 77 09      ADSB PPSL WC+C      ARITH WITH CARRY;SET CARRY
31     0502      05 02      LODI,R1 2          LOAD INDEX REGISTER
32     0504      B5 02      TPSEL COM
33     0506      18 0F      BCTR,ON LPSB      BRANCH IF SUBTRACTION
34     0508      75 01      CPSL C          ADDITION,CLEAR CARRY
35     050A 050A 0D 45 2D    LPAD LODA,R0 OPR1,R1,-  BYTE OF FIRST OPERAND TO R0
36     050D      8D 65 2F    ADDA,R0 OPR2,R1  ADD BYTE OF SECOND OPERAND
37     0510      CD 65 31    STRA,R0 RSLT,R1  STORE RESULT
38     0513      59 75      BRNR,R1 LPAD      BRANCH IF NOT DONE
39     0515      1B 0B      BCTR,UN TEST
40     0517 0517 0D 45 2D    LPSB LODA,R0 OPR1,R1,-  BYTE OF FIRST OPERAND TO R0
41     051A      AD 65 2F    SUBA,R0 OPR2,R1  SUB BYTE OF SECOND OPERAND
42     051D      CD 65 31    STRA,R0 RSLT,R1  STORE RESULT
43     0520      59 75      BRNR,R1 LPSB      BRANCH IF NOT DONE
44     0522 0522 98 08      TEST BCFR,Z RTRN      RETURN IF MS BYTE NOT ZERO
45     0524      0C 05 32    LODA,R0 RSLT+1
46     0527      14          RETC,Z          RETURN IF LS BYTE ALSO ZERO
47     0528      75 00      CPSL CC1        SET CC. TO #1 (POSITIVE)
48     052A      77 40      PPSL CC0
49     052C 052C 17          RTRN RETC,UN
50     *
51     052D      OPR1 RES 2          LOCATION OF: FIRST OPERAND
52     052F      OPR2 RES 2          SECOND OPERAND
53     0531      RSLT RES 2         RESULT
54     END

```

FIGURE 1-2

2. BINARY MULTIPLICATION FOR ONE-BYTE UNSIGNED INTEGERS

FUNCTION:

One byte by one byte multiplication for unsigned integers, giving a two-byte result.

$(OPR1) \times (OPR2) \rightarrow RSLT, RSLT + 1$

PARAMETERS:

Input OPR1 contains multiplier
OPR2 contains multiplicand

Output: RSLT contains high-order product-byte.
RSLT + 1 contains low-order product-byte.

SPECIAL REQUIREMENTS:

None

Refer to Figures 2.1 and 2.2 for flowchart and program listing.

REGISTERS	HARDWARE AFFECTED							
	R0	R1	R2	R3	R1'	R2'	R3'	
PSU	F	II	SP					
PSL	CC	IDC	RS	WC	OVF	COM	C	
	X	X		X	X		X	

RAM REQUIRED (BYTES): 4

ROM REQUIRED (BYTES): 29

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE
NESTING LEVELS: None

ASSEMBLER/COMPILER USED: _PIPHASM

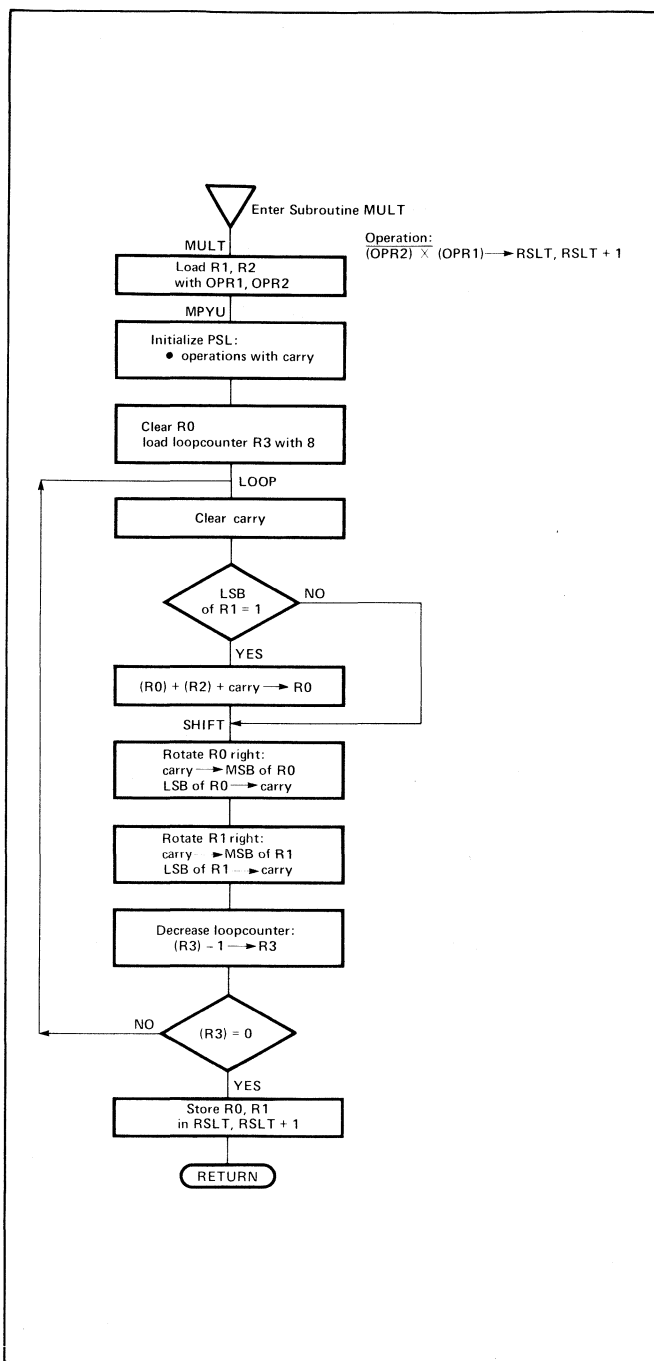


FIGURE 2-1 Flowchart for Unsigned Multiplication (One-Byte Operands; Two-Byte Result)


```

1          *      PD760030      *
2          *****
3          *      BINARY MULTIPLICATION FOR 2 UNSIGNED INTEGERS
4          *****
5          *
6          *      MULTIPLIER IS IN OPR1
7          *      MULTIPLICAND IS IN OPR2
8          *      RESULT WILL BE STORED IN RSLT,RSLT+1 (RSLT = MS BYTE)
9          *
10         *
11         *
12         *
13         *      SYMBOL DEFINITIONS
14         0000      R0      EQU      0
15         0001      R1      EQU      1
16         0002      R2      EQU      2
17         0003      R3      EQU      3
18         0001      R4      EQU      1
19         0002      R5      EQU      2
20         0003      R6      EQU      3
21         0003      UN      EQU      3      UNCONDITIONAL BRANCHING
22         0000      ON      EQU      0
23         0002      LT      EQU      2
24         0000      Z      EQU      0
25         0001      P      EQU      1
26         0002      N      EQU      2
27         0008      WC      EQU      8
28         0001      C      EQU      1
29         0040      F      EQU      H'40'
30         0004      OVF     EQU      4
31         0002      COM     EQU      2
32         *
33         *      R/W MEMORY
34         *
35         *      ORG H'500'
36         0500      OPR1     RES      2
37         0502      OPR2     RES      2
38         0504      RSLT     RES      4
39         *
40         *
41         *
42         *      ORG H'600'
43         0600 0600 0D 05 00      MULT      LODA,R1      OPR1      GET OPERAND IN R1
44         0603          0E 05 02      LODA,R2      OPR2      GET OPERAND IN R2
45         0606 0606 77 08          MPYU      PPSL      WC      ARITH
46         0608          20          EORZ      R0          CLEAR R0
47         0609          07 08          LODI,R3      8      LOAD LOOP COUNTER R3
48         060B 060B 75 01      LOOP      CPSL      C      CLEAR CARRY
49         060D          F5 01          TMI,R1      H'01'
50         060F          98 01          BCFR,ON     SHFT
51         0611          82          ADDZ      RZ
52         0612 0612 50          SHFT      RRR,R0
53         0613          51          RRR,R1
54         0614          FB 75          BDRR,R3     LOOP      BRANCH TO LOOP IF NOT READY
55         0616          CC 05 04      STRA,R0     RSLT     SAVE RESULT IN RESULT AREA
56         0619          CD 05 04      STRA,R1     RSLT+1   SAVE RESULT IN RESULT AREA
57         061C          17          RETC,UN

```

FIGURE 2-2

3. BINARY MULTIPLICATION FOR TWO-BYTE UNSIGNED INTEGERS

FUNCTION:

Two byte by two byte multiplication for unsigned integers, giving a four byte result.

$$(OPR2, OPR2 + 1) \times (OPR1, OPR1 + 1) \longrightarrow RSLT, RSLT + 1, RSLT + 2, RSLT + 3$$

PARAMETERS:

Input: (OPR1, OPR1 + 1) contains multiplier
(OPR2, OPR2 + 1) contains multiplicand

Output: RSLT, RSLT + 1, RSLT + 2, RSLT + 3 contains product.
OPR1, OPR2, and RSLT are most-significant bytes.

SPECIAL REQUIREMENTS:

None

Refer to Figures 3.1 and 3.2 for flowchart and program listing.

		HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'		
	X	X		X					
PSU	F	II	SP						
PSL	CC	IDC	RS	WC	OVF	COM	C		
	X	X		X	X		X		

RAM REQUIRED (BYTES): 8

ROM REQUIRED (BYTES): 57

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

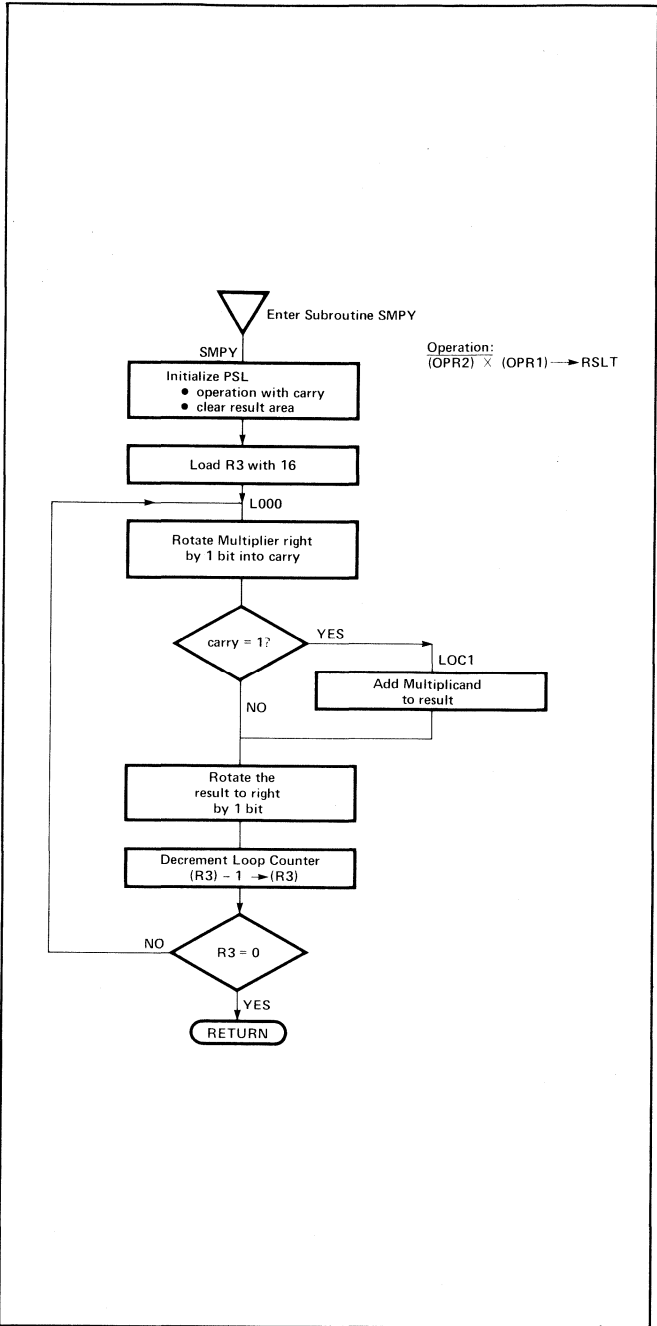


FIGURE 3-1 Flowchart for Unsigned Multiplication (Two-Byte Operands; Four-Byte Result)

```

58          * PD760031
59          *
60          * BINARY MULTIPLICATION FOR 2 TWO-BYTE INTEGERS
61          *
62          *
63          * MULTIPLIER IS IN OPR1 , OPR1+1
64          * MULTIPLICAND IS IN OPR2 ,OPR2+1
65          * RESULT WILL BE IN RSLT ,RSLT+1 ,RSLT+2 ,RSLT+3
66          * ORG H'790'
67          *
68 0790 0790 77 08 SMPY PPSL WC SET MODE
69 0792 20 EORZ R0
70 0793 CC 05 04 STRA,R0 RSLT CLEAR RESULT
71 0796 CC 05 05 STRA,R0 RSLT+1 CLEAR RESULT +1
72 0799 07 10 LODI,R3 16 LOAD COUNT
73 079B 079B 05 FE LOOO LODI,R1 -2 TO GET 254
74 079D 75 01 CPSL C CLEAR CARRY
75 079F 079F 0D 64 02 LOCO LODA,R0 OPR1-256+2,R1 FOR INDEXING INTO OPR1
76 07A2 50 RRR,R0 ROTATE RIGHT WITH C
77 07A3 CD 64 02 STRA,R0 OPR1-256+2,R1
78 07A6 D9 77 BIRR,R1 LOCO ROTATE 2ND TIME
79          * THIS ROTATES MULTIPLIER BY 1 BIT TO GET THE LSB
80          * INTO CARRY
81 07A8 20 EORZ R0 CLEAR R0
82 07A9 D0 RRL,R0 GET CARRY INTO LSB
83 07AA F8 02 BDRR,R0 LOC1
84 07AC 1B 0D BCTR,UN LOC4
85          *
86 07AE 07AE 05 02 LOC1 LODI,R1 2 GET INDEX
87 07B0 07B0 0D 65 03 LOC2 LODA,R0 RSLT-1,R1 ADD MULTIPLICAND TO PRODUCT
88 07B3 8D 65 01 ADDA,R0 OPR2-1,R1
89 07B6 CD 65 03 STRA,R0 RSLT-1,R1
90 07B9 F9 75 BDRR,R1 LOC2 FINISH THE ADD
91          *
92          *
93 07BB 07BB 05 FC LOC4 LODI,R1 -4 ROTATE THE PRODUCT TO RIGHT
94 07BD 07BD 0D 64 08 LOC5 LODA,R0 RSLT-256+4,R1
95 07C0 50 RRR,R0 ROTATE RESULT
96 07C1 CD 64 08 STRA,R0 RSLT-256+4,R1
97 07C4 D9 77 BIRR,R1 LOC5
98 07C6 FB 53 BDRR,R3 LOOO FINISH THE LOOP
99 07C8 17 RETC,UN

```

FIGURE 3-2

4. BINARY MULTIPLICATION FOR ONE-BYTE SIGNED INTEGERS

FUNCTION:

One byte by one byte multiplication for signed integers giving a two-byte result.

$(OPR1) \times (OPR2) \rightarrow RSLT, RSLT + 1$

The Booth algorithm is used (see Figure 4.1).

PARAMETERS:

Input: OPR1 contains multiplier
OPR2 contains multiplicand

Output: RSLT contains high-order product byte.
RSLT + 1 contains low-order product byte.

SPECIAL REQUIREMENTS:

None

Refer to Figures 4.1 and 4.2 for flowcharts and to Figure 4.3 for program listing.

		HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'			
	X	X	X	X						
PSU	F	II	SP							
PSL	CC	IDC	RS	WC	OVF	COM	C			
	X	X		X	X		X			

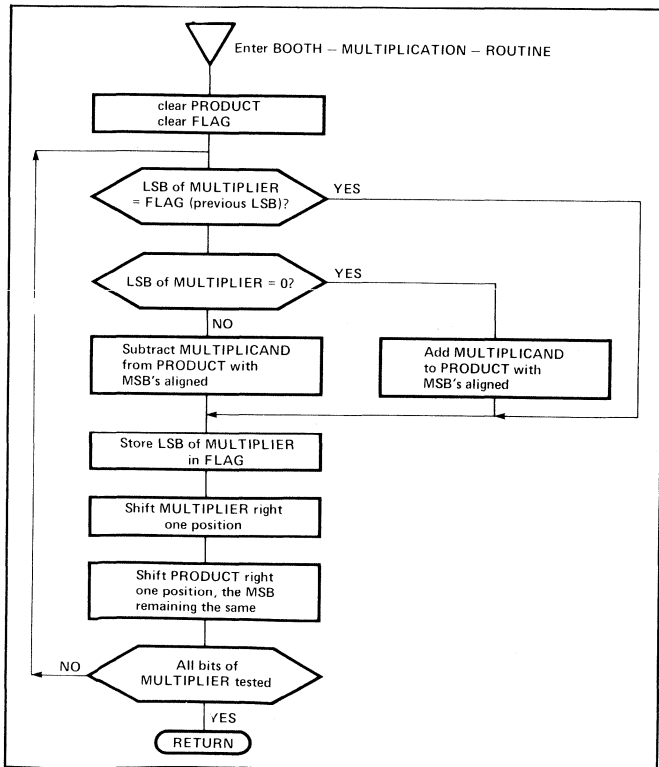


FIGURE 4-1 Flowchart of Booth Algorithm
Multiplicand \times Multiplier \rightarrow Product

RAM REQUIRED (BYTES):	4
ROM REQUIRED (BYTES):	51
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

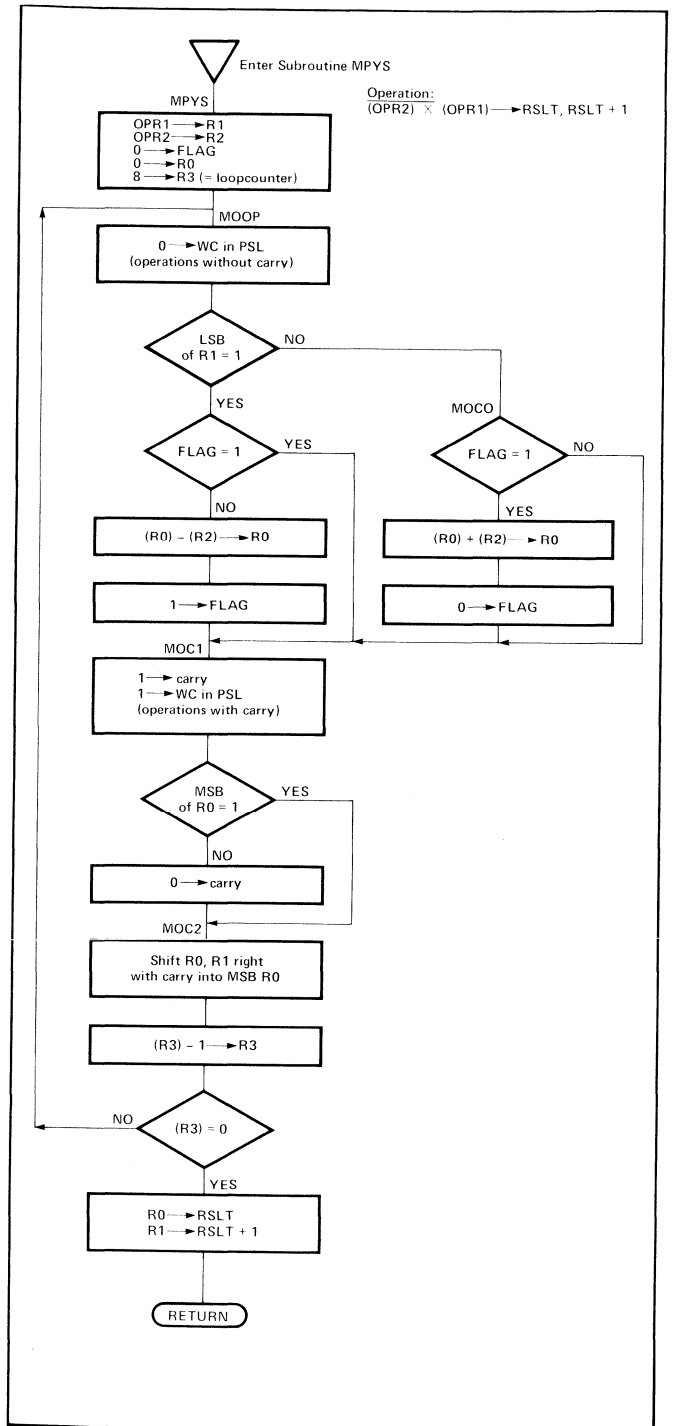


FIGURE 4-2 Flowchart for Signed Multiplication Using Booth Algorithm (One-Byte Operands; Two-Byte Result)

```

100 * PD760032
101 *****
102 * BINARY MULTIPLICATION USING BOOTH-ALGORITHM
103 * FOR 2 ONE-BYTE SIGNED INTEGERS.
104 *****
105 * FIRST OPERAND IS IN OPR1
106 * SECOND OPERAND IS IN OPR2 (OPR2) ≠ H'80'
107 * PRODUCT WILL BE IN RSLT,RSLT+1
108 *
109 ORG H'800'
110 0000 0800 74 40 MPYS CPSU F CLEAR FLAG IN PSU
111 0002 0D 05 00 LODA,R1 OPR1 GET 1ST OPERAND
112 0005 0E 05 02 LODA,R2 OPR2 GET 2ND OPERAND
113 0008 07 08 LODI,R3 8 LOAD LOOP COUNTER R3
114 000A 20 EORZ R0 CLEAR R0
115 000B 080B 75 08 MOOP CPSL WC CLEAR WC IN PSL
116 000D F5 01 TMI,R1 H'01'
117 000F 98 09 BCFR,ON MOC0 LSB OF R1 SET?
118 0011 B4 40 TPSU F YES
119 0013 18 0C BCTR,ON MOC1 FLAG =1?
120 0015 A2 SUBZ R2 NO,SUBTRACT WITHOUT BORROW
121 0016 76 40 PPSU F SET FLAG
122 0018 1B 07 BCTR,UN MOC1 BRANCH TO DOUBLE SHIFT
123 001A 081A B4 40 MOC0 TPSU F LSB OF R1 WAS 0
124 001C 98 03 BCFR,ON MOC1 FLAG =1?
125 001E 82 ADDZ R2 YES,ADD WITHOUT CARRY
126 001F 74 40 CPSU F CLEAR FLAG
127 0021 0821 77 09 MOC1 PPSL WC+C SET C AND WC
128 0023 60 IORZ R0
129 0024 1A 02 BCTR,N MOC2 MSB OF R0 SET?
130 0026 75 01 CPSL C NO,CLEAR CARRY
131 0028 0828 50 MOC2 RRR,R0 SHIFT R0 R1 RIGHT
132 0029 51 RRR,R1 MSB OF R0 IS SAME
133 002A FB 5F BDRR,R3 MOOP BRANCH TO LOOP IF NOT READY
134 002C CC 05 04 STRA,R0 RSLT STORE RESULT
135 002F CD 05 05 STRA,R1 RSLT+1
136 0032 17 RETC,UN EXIT SUBROUTINE MPYS
137 *

```

FIGURE 4-3

5. BINARY MULTIPLICATION FOR TWO-BYTE SIGNED INTEGERS

FUNCTION:

Two byte by two byte multiplication for signed integers giving a four byte result.

$$(OPR1, OPR1 + 1) \times (OPR2, OPR2 + 1)$$

→ RSLT, RSLT + 1, RSLT + 2, RSLT + 3.

The Booth algorithm (Figure 4.1) is used.

PARAMETERS:

Input: OPR1, OPR1 + 1 contains multiplicand
OPR2, OPR2 + 1 contains multiplier

Output: RSLT, RSLT + 1, RSLT + 2, RSLT + 3 contains product.

OPR1, OPR2, and RSLT are most-significant bytes.

SPECIAL REQUIREMENTS

None

Refer to Figure 5.1 for flowchart and to Figure 5.2 for program listing.

		HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'			
	X	X	X	X						
PSU	F	II	SP							
PSL	CC	IDC	RS	WC	OVF	COM	C			
	X	X		X	X		X			

RAM REQUIRED (BYTES): 8

ROM REQUIRED (BYTES): 71

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

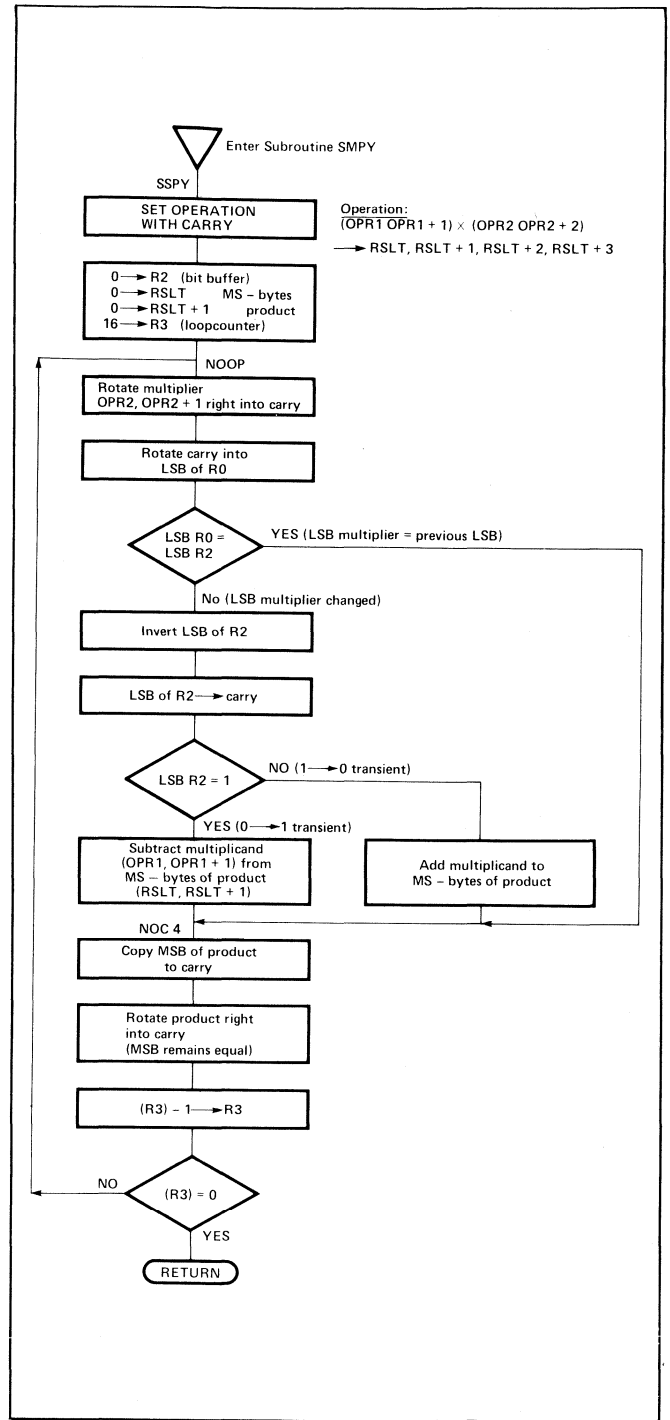


FIGURE 5-1 Flowchart for Signed Multiplication Using Booth Algorithm (Two-Byte Operands; Four-Byte Result)

```

138      * PD760033
139      *****
140      * BINARY MULTIPLICATION FOR TWO BYTE SIGNED INTEGERS
141      *****
142      * MULTIPLICAND IS IN LOCATIONS OPR1,OPR1+1
143      * MULTIPLIER IS IN LOCATIONS OPR2,OPR2+1
144      *
145      * RESULT WILL BE STORED IN RSLT,RSLT+1,RSLT+2,RSLT+3
146      *
147      * AFTER MULTIPLICATION THE MULTIPLICAND IS UNCHANGED
148      * THE MULTIPLIER IS DESTROYED
149      * THE MULTIPLICAND MUST BE UNEQUAL H'0000'
150      *****
151 0033 0033 77 00      SSPY  PPSL  WC          ARITH AND ROTATE WITH C
152 0035      20          EORZ  R0          CLEAR R0
153 0036      C2          STRZ  R2          CLEAR R2
154 0037      CC 05 04   STRA,R0 RSLT      CLEAR 2 MSBYTES OF PRODUCT
155 003A      CC 05 05   STRA,R0 RSLT+1
156 003D      07 10     LODI,R3 16          LOAD LOOP COUNTER R3
157 003F 003F 05 FE     NOOP  LODI,R1 -2         LOAD INDEX REG WITH 254
158 0041 0041 0D 64 04  NOC0   LODA,R0 OPR2-256+2,R1 ROTATE MULTIPLIER
159 0044      50          RRR,R0 INTO CARRY
160 0045      CD 64 04   STRA,R0 OPR2-256+2,R1
161 0048      D9 77     BIRR,R1 NOC0          BRANCH IF NOT DONE
162 004A      20          EORZ  R0          CLEAR R0
163 004B      D0          RRL,R0 ROTATE CARRY IN LSB OF R0
164 004C      22          EORZ  R2          LSB OF R0 BECOMES 1 FOR CHANGE
165 004D      18 19     BCTR,Z  NOC4          BRANCH IF NO CHANGE
166 004F      22          EORZ  R2          INVERT LSB OF R2
167 0050      C2          STRZ  R2          RESTORE NEW R2
168 0051      50          RRR,R0 LSB OF R2 INTO CARRY OR BORROW
169 0052      05 02     LODI,R1 2          LOAD INDEX
170 0054 0054 0D 45 04  NOC1   LODA,R0 RSLT,R1,-  LOAD BYTE OF RSLT IN R0
171 0057      F6 01     TMI,R2 1
172 0059      18 05     BCTR,ON NOC2          BRANCH TO SUBTRACT IF LSB R2=1
173 005B      8D 65 00   ADDA,R0 OPR1,R1      ADD BYTE MPLCND TO RSLT
174 005E      1B 03     BCTR,UN NOC3
175 0060 0060 AD 65 00  NOC2   SUBA,R0 OPR1,R1      SUB BYTE MPLCND FROM RSLT
176 0063 0063 CD 65 04  NOC3   STRA,R0 RSLT,R1      RESTORE INTERMEDIATE RSLT
177 0066      59 6C     BRNR,R1 NOC1          BRANCH IF ADD SUBTRACT NOT READY
178      *
179 0068 0068 0C 05 04  NOC4   LODA,R0 RSLT
180 006B      D0          RRL,R0
181 006C      04 FC     LODI,R0 -4          LOAD INDEX
182 006E 006E 0D 64 00  NOC5   LODA,R0 RSLT-256+4,R1 FETCH MS BYTE PRODUCT
183 0071      50          RRR,R0 ROTATE RSLT,PROD+1 ETC TO RIGHT
184 0072      CD 64 00   STRA,R0 RSLT-256+4,R1 KEEPING MSB SAME
185 0075      D9 77     BIRR,R1 NOC5          BRANCH IF NOT DONE
186 0077      FB 46     BDRR,R3 NOOP          BRANCH IF LOOP NOT READY
187 0079      17          RETC,UN RETURN TO MAIN PROGRAM
188      END

```

FIGURE 5-2

6. BINARY DIVISION

A. UNSIGNED INTEGERS
TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

FUNCTION:

Division of a two byte dividend by a one byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$$\frac{(DVDN, DVDN + 1)}{(DVSR)} \rightarrow \begin{cases} DVDN, DVDN + 1 & \text{(quotient)} \\ R1 & \text{(remainder)} \end{cases}$$

PARAMETERS:

Input: DVDN, DVDN + 1 contains dividend
DVSR contains divisor
DVDN is most-significant byte

Output: DVDN, DVDN + 1 contains quotient
R1 contains remainder
DVDN is most-significant byte.
Dividend is destroyed after execution of division.

SPECIAL REQUIREMENTS:

None

Refer to Figure 6.1 for flowchart and to Figure 6.2 for program listing.

		HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'			
	X	X	X	X						
PSU	F	II	SP							
PSL	CC	IDC	RS	WC	OVF	COM	C			
	X	X		X	X	X	X			

RAM REQUIRED (BYTES):	3
ROM REQUIRED (BYTES):	45
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

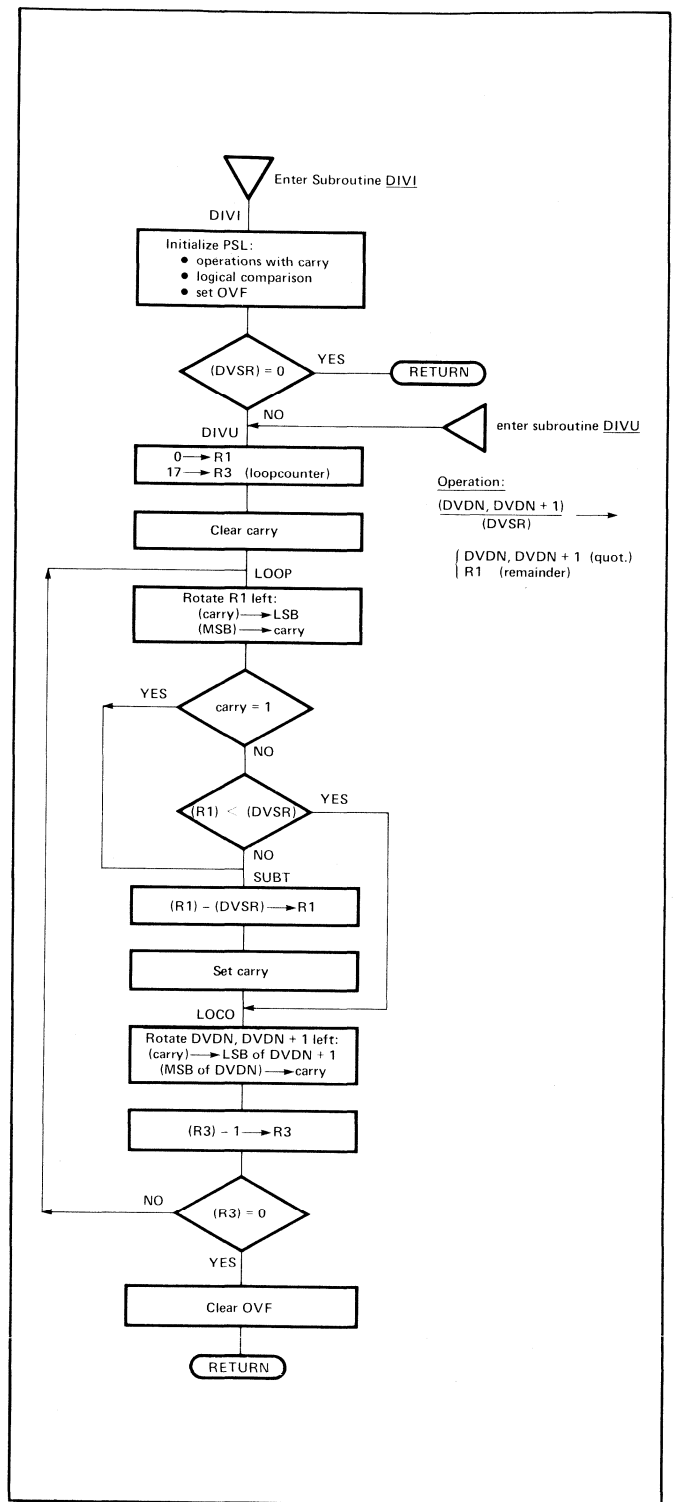


FIGURE 6-1 Flowchart for Unsigned Division (Dividend or Quotient: Two-Bytes; Divisor or Remainder: One-Byte)


```

1          *          PD760040          *
2          *-----*
3          *   BINARY DIVISIONS FOR INTEGERS
4          *-----*
5          * DIVIDEND IS IN DVDN,DVDN+1 16 BITS
6          * DIVISOR IS IN DVSR          8 BITS
7          * QUOTIENT WILL BE IN DVDN,DVDN+1 16 BITS
8          * AFTER DIVISION, DIVIDEND WILL BE DESTROYED
9          * R1 WILL HOLD REMAINDER
10         * OVF=1 IMPLIES OVERFLOW
11         *
12         *
13         *          SYMBOL DEFINITIONS
14         0000      R0      EQU      0
15         0001      R1      EQU      1
16         0002      R2      EQU      2
17         0003      R3      EQU      3
18         0001      R4      EQU      1
19         0002      R5      EQU      2
20         0003      R6      EQU      3
21         0003      UN      EQU      3          UNCONDITIONAL BRANCHING
22         0001      C       EQU      1
23         0000      ON      EQU      0
24         0002      LT      EQU      2
25         0000      Z       EQU      0
26         0000      EQ      EQU      0
27         0001      P       EQU      1
28         0002      N       EQU      2
29         0008      WC      EQU      8
30         0004      OVF     EQU      4
31         0002      COM     EQU      2
32         *
33         *          ORG      H'500'          UNSIGNED DIVISION SUBROUTINE
34         *
35         0500 0500 77 0E      DIVI      PPSL      WC+OVF+COM      ARITH ROTATE WITH CARRY
36         0502          0C 06 02      LODA,R0      DVSR          FETCH DIVISOR
37         0505          14          RETC,Z          RETURN WITH OVF =1 IF DVSR =0
38         *
39         0506 0506 05 00      DIVU      LODI,R1      0          CLR R1
40         0508          07 11      LODI,R3      17         LOAD LOOP COUNTER R3
41         050A          75 01      CPSL      C          CLEAR CARRY
42         050C 050C D1          LOOP      RRL,R1          ROTATE CARRY IN LSB OF R1
43         050D          B5 01      TPSL      C
44         050F          18 05      BCTR,ON      SUBT          GO TO SUBTRACT IF CARRY =1
45         0511          ED 06 02      COMA,R1      DVSR
46         0514          1A 07      BCTR,LT      LOC0          IF R1<DVSR,NO SUBTRACTION
47         0516 0516 77 01      SUBT      PPSL      C          CLR BORROW
48         0518          AD 06 02      SUBA,R1      DVSR          SUBTR DVSR FROM REMAINDER
49         051B          77 01      PPSL      C          SET CARRY
50         051D 051D 06 02      LOC0      LODI,R2      2          LOAD INDEX REGISTERR
51         051F 051F 0E 46 00      LOC1      LODA,R0      DVDN,R2,-      ROTATE QUOTIENT BIT
52         0522          D0          RRL,R0          DVDN,DVDN+1 AND MSB OF
53         0523          CE 66 00      STRA,R0      DVDN,R2      DVDN INTO CARRY
54         0526          5A 77      BRNR,R2      LOC1          BRANCH IF ROTATE NOT READY
55         0528          FB 62      BDRR,R3      LOOP          BRANCH IF DIVISION NOT READY
56         052A          75 04      CPSL      OVF          CLEAR OVF IN PSL
57         052C          17          RETC,UN          RETURN TO MAIN PROGRAM
58         *

```

FIGURE 6-2

B. SIGNED INTEGERS
TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

FUNCTION:

Division of a two-byte dividend by a one-byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$$\frac{(DVDN, DVDN + 1)}{(DVSR)} \rightarrow \begin{cases} DVDN, DVDN + 1 & \text{(quotient)} \\ R1 & \text{(remainder)} \end{cases}$$

PARAMETERS:

Input: DVDN, DVDN + 1 contains dividend
 DVSR contains divisor
 DVDN is most-significant byte.

Output: DVDN, DVDN + 1 contains quotient
 R1 contains remainder
 DVDN is most-significant byte.
 Dividend is destroyed after execution of division;
 negative divisor becomes positive

SPECIAL REQUIREMENTS:

Software: Unsigned division subroutine
 Refer to Figure 6.3 for flowchart and to Figure 6.4 for program listing.

HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'	
	X	X	X	X				
PSU	F	II	SP					
PSL	CC	IDC	RS	WC	OVF	COM	C	
	X	X		X	X	X	X	

RAM REQUIRED (BYTES): _____ 4 _____

ROM REQUIRED (BYTES): _____ 61 _____

EXECUTION TIME: _____ Variable _____

MAXIMUM SUBROUTINE NESTING LEVELS: _____ 1 _____

ASSEMBLER/COMPILER USED: _____ PIPHASM _____

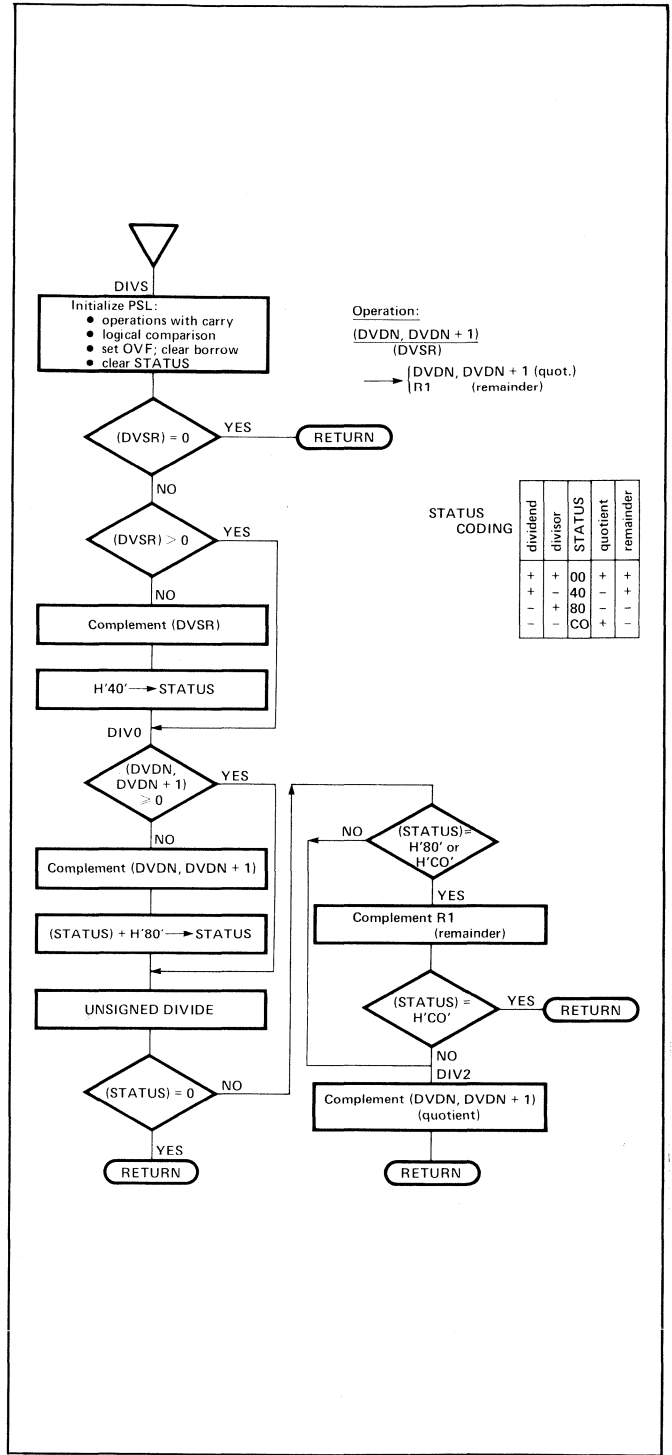


FIGURE 6-3 Flowchart for Signed Division (Dividend & Quotient: 2 Bytes; Divisor & Remainder: 1 Byte)

```

59      * PD760041
60      * *****
61      * SIGNED DIVISION
62      * *****
63      *
64      * NEGATIVE DIVIDEND AND OR DIVISOR ARE COMPLEMENTED
65      * PRIOR TO EXECUTION OF DIVISION
66      *
67      * SIGNS ARE CODED IN STATUS:
68      * STATUS CODING:DVDN DVSR STAT QUOT RMDR
69      *      +      +      00      +      +
70      *      +      -      40      -      +
71      *      -      +      80      -      -
72      *      -      -      C0      +      -
73      * DIVIDEND MUST BE UNEQUAL H'0000' (NO CORRECT OVF)
74      * NEGATIVE SIGN OF DIVISOR IS LOST AFTER EXECUTION.
75 052D 052D 77 00      DIVS      PPSL      WC+OVF+C      ARITH ROTATE WITH CARRY ETC
76 052F      20      EORZ      R0
77 0530      C1      STRZ      R1      CLEAR R1
78 0531      0E 06 02      LODA,R2      DVSR      FETCH DIVISOR IN R2
79 0534      14      RETC,Z      RETURN WITH OVF SET IF DVSR= 0
80 0535      19 06      BCTR,P      DIV0      BRANCH IF DIVISOR >0
81 0537      A2      SUBZ      R2      TAKE 2S COMPLEMENT OF DVSR
82 0538      CC 06 02      STRA,R0      DVSR      RESTORE DIVISOR
83 053B      05 40      LODI,R1      H'40'      LOAD STATUS IN R1
84 053D 053D 0E 06 00      DIV0      LODA,R2      DVDN      FETCH MS BYTE OF DIVIDEND
85 0540      9A 04      BCFR,N      DIV1      BRANCH IF DIVIDEND NOT<0
86 0542      3B 18      BSTR,UN      CMPL      TAKE 2S COMPLEMENT OF DIVIDEND
87 0544      85 80      ADDI,R1      H'80'      UPDATE STATUS
88 0546 0546 CD 06 03      DIV1      STRA,R1      STAT      SAVE STATUS
89 0549      3F 05 06      BSTA,UN      DIVU      CALL UNSIGNED DIVISION
90 054C      0F 06 03      LODA,R3      STAT      LOAD STATUS IN R3
91 054F      14      RETC,Z      RETURN IF BOTH DVDN AND DVSR NOT<0
92 0550      19 07      BCTR,P      DIV2      BRANCH IF DVDN WAS NOT <0 AND DVSR<0
93 0552      77 01      PPSL      C      CLEAR BORROW
94 0554      20      EORZ      R0      CLEAR R0
95 0555      A1      SUBZ      R1      TAKE 2 S COMPLEMENT OF REMAINDER
96 0556      C1      STRZ      R1      RESTORE REMAINDER IN R1
97 0557      D3      RRL,R3      SHIFT R3 LEFT
98 0558      16      RETC,N      RETURN IF BOTH DVDN,DVSR<0
99 0559 0559 3B 01      DIV2      BSTR,UN      CMPL      TAKES 2S COMPL. OF QUOTIENT
100 055B      17      RETC,UN      RETURN TO MAINPROGRAM
101
102
103      * SUBROUTINE TO TAKE 2S COMPL
104      * OF (DVDN,DVDN+1)
105      *
106 055C 055C 77 01      CMPL      PPSL      C      CLEAR BORROW
107 055E      07 02      LODI,R3      2      LOAD INDEX REG
108 0560 0560 20      CMP0      EORZ      R0      CLR R0
109 0561      AF 46 00      SUBA,R0      DVDN,R3,-      COMPLEMENT BYTE
110 0564      CF 66 00      STRA,R0      DVDN,R3      RESTORE RESULT
111 0567      5B 77      BRNR,R3      CMP0      BRANCH IF NOT DONE
112 0569      17      RETC,UN
113
114      0600      DVDN      RES      2      DIVIDEND AND QUOTIENT
115      0602      DVSR      RES      1      DIVISOR
116      0603      STAT      RES      1      STATUS REG
117
      END

```

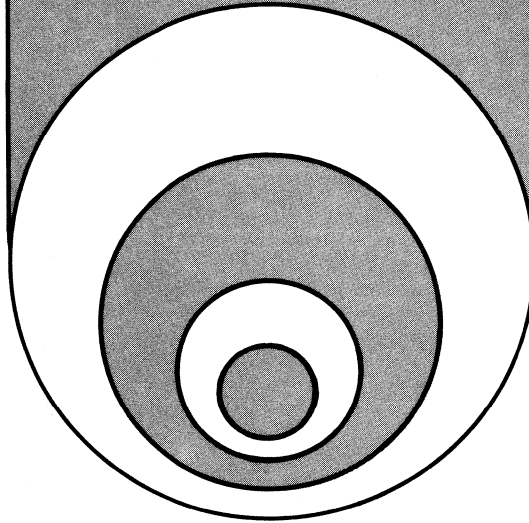
FIGURE 6-4

Signetics 2650 Microprocessor application memos currently available:

AS50	Serial Input/Output
AS51	Bit and Byte Testing Procedures
AS52	General Delay Routines
AS54	Conversion Routines
SP50	2650 Evaluation Printed Circuit Board Level System (PC1001)
SP51	2650 Demo Systems
SP52	Support Software for use with the NCSS Timesharing System
SP53	Simulator, Version 1.2
SP54	Support Software for use with the General Electric Mark III Timesharing System
SS50	PIPBUG
SS51	Absolute Object Format (Revision 1)
MP51	2650 Initialization
MP52	Low Cost Clock Generator Circuits

signetics

**MOS
MICROPROCESSOR**



CONVERSION ROUTINES. . . .AS54

INTRODUCTION

Conversion routines like binary to BCD, BCD to binary, and BCD to ASCII are often used in microprocessor based systems. This applications memo describes routines for converting:

- Eight-bit unsigned binary to BCD.
- Sixteen-bit signed binary to BCD.
- Signed BCD to binary conversion 1 (using an addition method).
- Signed BCD to binary conversion 2 (using a multiplication method).
- Signed BCD to ASCII
- ASCII to BCD
- Hexadecimal to ASCII
- ASCII to Hexadecimal

1. EIGHT-BIT UNSIGNED BINARY-TO-BCD CONVERSION

FUNCTION:

Converts an unsigned binary number to a BCD number (3 digits).

(BINN) $\xrightarrow{\text{Conversion}}$ R0, R1

A multiplication method is used.

PARAMETERS:

Input: BINN contains the binary number (8 bits unsigned).

Output: Registers R0, R1 contain the BCD result (3 BCD digits).

R0 is the most-significant byte.

The maximum BCD result is 256 decimal.

Refer to figures 1.1 and 1.2 for flowchart and program listing.

REGISTERS	HARDWARE AFFECTED							
	R0	R1	R2	R3	R1'	R2'	R3'	
PSU	F	II	SP					
PSL	CC	IDC	RS	WC	OVF	COM	C	
	X	X		X	X	X	X	

RAM REQUIRED (BYTES):	1
ROM REQUIRED (BYTES):	28
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

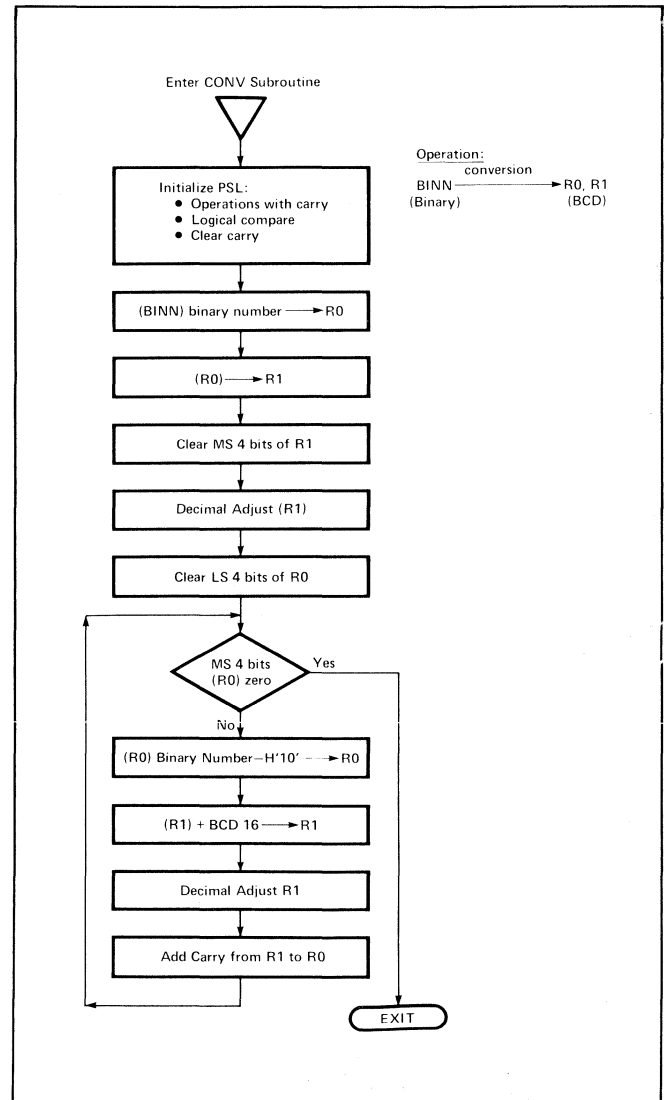


FIGURE 1-1 Flowchart for Eight-Bit Unsigned Binary-to-BCD Conversion (Multiplication Method)

```

1          * PD760050
2          *+++++
3          * 8 BIT UNSIGNED BINARY TO BCD CONVERSION
4          *+++++
5          *
6          *THIS ROUTINE CONVERTS AN 8 BIT UNSIGNED BINARY
7          *NUMBER INTO AN UNSIGNED BCD NUMBER.
8          *
9          *BINARY NUMBER IS IN BINN.
10         *BCD NUMBER (AFTER CONVERSION) IS IN R0,R1.
11         * HUNDREDS IN R0
12         * TENS,UNITS IN R1.
13         *
14         *DEFINITIONS OF SYMBOLS:
15         *
16         0000 R0 EQU 0 PROCESSOR-REGISTERS
17         0001 R1 EQU 1
18         0008 WC EQU H'08' PSL: 1=WITH, 0=WITHOUT CARRY
19         0002 COM EQU H'02' 1=LOGIC, 0=ARITH.COMPARE
20         0001 C EQU H'01' CARRY:BORROW
21         0003 UN EQU 3 BRANCH COND.: UNCONDITIONAL
22         0002 LT EQU 2 LESS THAN
23         *
24         *
25         ORG H'600'
26         *
27         0600 BINN RES 1 BINARY NUMBER.
28         *
29         ORG H'500' START ADDRESS OF ROUTINE.
30         *
31         *
32         0500 0500 77 0A CONV PPSL WC+COM INITIALISATION:
33         0502 75 01 CPSL C WITH CARRY,LOGICAL COMPARE
34         0504 0C 06 00 LODA,R0 BINN CLEAR CARRY FLAG IN PSL.
35         0507 C1 STRZ R1 8 BIT BIN.NUMBER -> R0.
36         0508 45 0F ANDI,R1 H'0F' (R0) -> R1.
37         050A 85 66 ADDI,R1 H'66' CLEAR MS 4 BITS BIN. NUMBER
38         050C 95 DAR,R1 PREPARE R1 FOR DECIMAL ADJUST.
39         *
40         050D 44 F0 ANDI,R0 H'F0' CLEAR LS 4 BITS.
41         *
42         050F 050F E4 10 LOOP COMI,R0 H'10'
43         0511 1A 09 BCTR,LT EXIT IF MS 4 BITS ZERO THEN RETRUN.
44         0513 A4 0F SUBI,R0 H'10'-1 SUBTRACT 1 FROM MS 4 BITS
45         0515 85 7B ADDI,R1 H'16'+H'66'-1 ADD BCD 16 AND PREPARE
46         0517 95 DAR,R1 FOR DECIMAL ADJUST.
47         0518 84 00 ADDI,R0 0 ADD CARRY TO MS BCD DIGIT
48         051A 1B 73 BCTR,UN LOOP BRANCH AGAIN
49         *
50         051C 051C 40 EXIT HALT END OF CONVERSION.
51         END

```

FIGURE 1-2 Program Listing for Eight-Bit Unsigned Binary-to-BCD Conversion

2. SIXTEEN-BIT SIGNED BINARY-TO-BCD CONVERSION

FUNCTION:

Converts a signed 16-bit binary number to a signed BCD number.

Subtraction of base numbers is used.

PARAMETERS:

Input: BINN, BINN+1 contain the signed binary number.

BINN is the most-significant byte.

Binary number is destroyed after conversion.

Output: BCDD, BCDD+1, BCDD+2 contain the BCD result.

BCDD contains the sign and the most-significant BCD digit.

The minimum BCD result is -32768 decimal.

The maximum BCD result is +32767 decimal.

Refer to figures 2.1 and 2.2 for flowchart and program listing.

		HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'		
	X	X	X	X					
PSU	F	II	SP						
PSL	CC	IDC	RS	WC	OVF	COM	C		
	X	X		X	X		X		

RAM REQUIRED (BYTES):	5
ROM REQUIRED (BYTES):	106
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

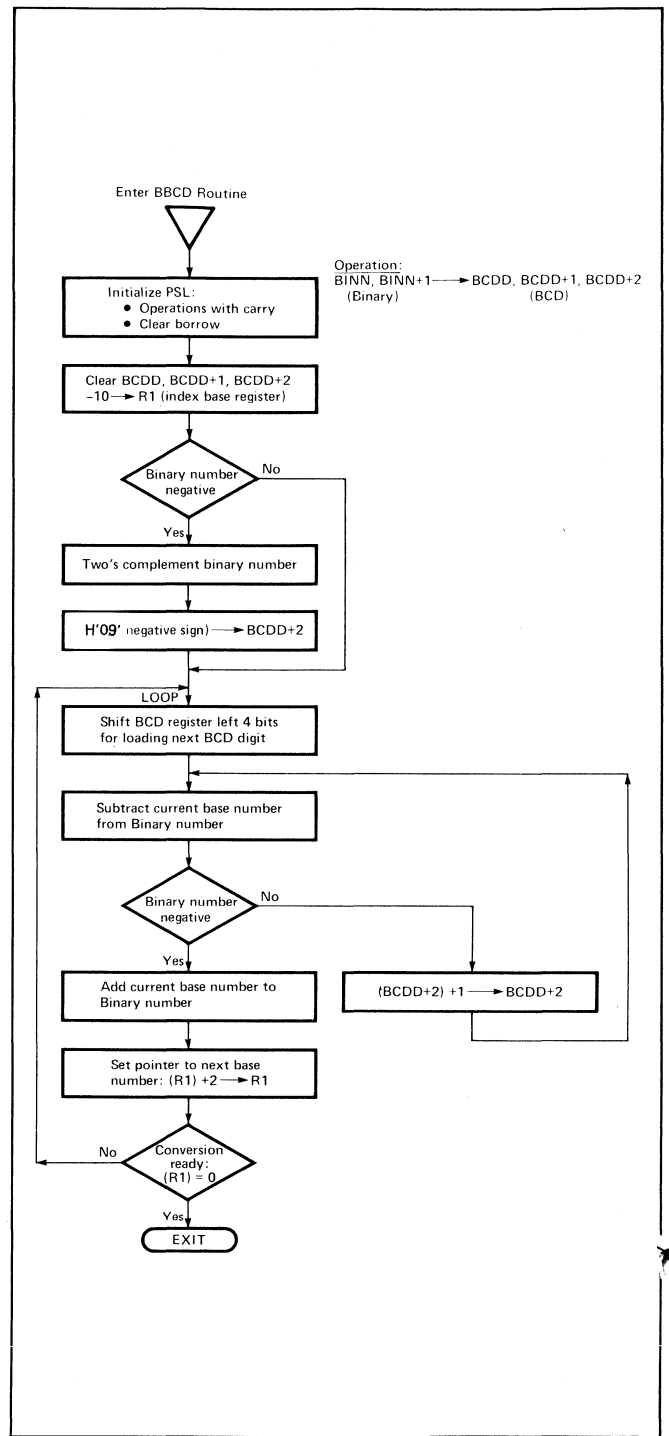


FIGURE 2-1 Flowchart for Signed Binary-to-BCD Conversion


```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20      0000
21      0001
22      0002
23      0003
24      0000
25      0001
26      0002
27      0003
28
29
30
31
32      0600
33      0602
34      0605 0605 27 10
35      0607      03 E8
36      0609      00 64
37      060B      00 0A
38      060D      00 01
39      000A
40
41
42
43      0500 0500 77 09
44
45      0502      20
46      0503      07 03
47      0505 0505 CF 46 02
48      0508      5B 78
49      050A      05 F6
50
51      050C      0E 06 00
52      050F      9A 10
53      0511 0511 06 02
54      0513 0513 20
55      0514      AE 46 00
56      0517      CE 66 00
57      051A      5A 77
58      051C      04 09
59      051E      CC 06 04
60
61      0521 0521 75 01
62      0523      04 04
63      0525 0525 07 03
64      0527 0527 0F 46 02
65      052A      D0
66      052B      CF 66 02
67      052E      5B 77
68      0530      FA 73
69
70      0532 0532 85 02
71      0534      06 02
72      0536      77 01
73      0538 0538 0E 46 00
74      053B      AD 45 0F
75      053E      CE 66 00
76      0541      5A 75
77      0543      1A 09
78      0545      0C 06 04
79      0548      82
80      0549      CC 06 04
81      054C      1B 64
82
83      054E 054E 06 02
84      0550 0550 0E 46 00
85      0553      8D 45 11
86      0556      CE 66 00
87      0559      5A 75
88      055B      05 03
89      055D      59 42
90      055F 055F 43
91

```

```

* PD760051
*****
* BINARY TO BCD CONVERSION
*****
*
*THIS ROUTINE CONVERTS A SIGNED BINARY NUMBER
*(16 BITS) INTO A SIGNED BCD NUMBER
*(24 BITS: SIGN + 5 BCD DIGITS).
*
*THE BINARY NUMBER IS IN BINN,BINN+1.
*THE BCD NUMBER IS IN BCDD,BCDD+1,BCDD+2.
*BINN AND BCDD ARE MOST SIGNIFICANT BYTES.
*MS NIBBLE OF BCDD=0 FOR POSITIVE BINARY NUMBERS.
*MS NIBBLE OF BCDD=9 FOR NEGATIVE BINARY NUMBERS.
*
*SUBTRAHENDS ARE PLACED IN REGISTER BASE (10 BYTES)
*
*DEFINITION OF SYMBOLS:
*
R0 EQU 0 PROCESSOR-REGISTERS
R1 EQU 1
R2 EQU 2
R3 EQU 3
WC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
C EQU H'01' CARRY+BORROW
N EQU 2 BRANCH COND.: NEGATIVE
UN EQU 3 UNCONDITIONALLY
*
*
ORG H'600' START ADDRESS
*
BINN RES 2 BINARY NUMBER MEMORY LOCATION
BCDD RES 3 BCD REGISTER
BASE DATA H'27,10' 10000
DATA H'03,E8' 1000
DATA H'00,64' 100
DATA H'00,0A' 10
DATA H'00,01' 1
LEN EQU 9-BASE LENGTH BASE REGISTER
ORG H'500' START ADDR. OF PROGRAM
*
BBCD PPSL WC+C ARITHMETIC+ROTATE WITH CARRY:
CLEAR BORROW.
*
EORZ R0 INITIALISATION: CLEAR R0.
LODI,R3 3
LOC0 STRA,R0 BCDD,R3,- CLEAR 3 BYTES OF BCD REGISTER.
BRNR,R3 LOC0
LODI,R1 -LEN LENGTH OF BASE REGISTER.
*
LDDA,R2 BINN MS 4 BITS BINARY NUMBER.
BCFR,N LOOP IF POS. GO TO LOOP
COMP LODI,R2 2 LOAD INDEX REGISTER.
LOC1 EORZ R0 TWO'S COMPLEMENT BY
SUBA,R0 BINN,R2,- SUBTRACTING FROM ZERO.
STRA,R0 BINN,R2
BRNR,R2 LOC1 RETURN IF NOT READY.
LODI,R0 H'09' NEGATIVE SIGN INDICATION.
STRA,R0 BCDD+2 SIGN IN LSB OF BCD REGISTER.
SHIFT BCD REG. LEFT 4 TIMES.
*
LOOP CPSL C CLEAR CARRY FOR ROTATE.
LODI,R2 4 BIT COUNT.
LP2 LODI,R3 3 INDEX BYTE SHIFT.
LP1 LODA,R0 BCDD,R3,- BCD DIGIT INTO R0.
RRL,R0 CARRY (PREVIOUS MS BIT)-> LSB
STRA,R0 BCDD,R3 AND MS BIT -> CARRY.
BRNR,R3 LP1
BDRR,R2 LP2
*
SUBL ADDI,R1 2 RESTORE BASE INDEX.
LODI,R2 2 INDEX REGISTER
PPSL C CLEAR BORROW
LOC2 LODA,R0 BINN,R2,- LOAD BINN AND SUBTRACT
SUBA,R0 BASE-256+LEN+R1,- CORRESPONDING
STRA,R0 BINN,R2 BASE DIGIT
BRNR,R2 LOC2
BCTR,N CORR IF BINN NEG. THEN CORRECTION.
LDDA,R0 BCDD+2
ADDZ R2 ADD 1 TO LSB OF BCD NUMBER
STRA,R0 BCDD+2 C=1 IN PSL AND (R2)=0
BCTR,UN SUBL
*
CORR LODI,R2 2 INDEX COUNT
LOC3 LODA,R0 BINN,R2,- ADD CORRESPONDING BASE BYTE TO
ADDA,R0 BASE-256+LEN+2,R1,- BINARY NUMBER.
STRA,R0 BINN,R2
BRNR,R2 LOC3 RETURN IF NOT READY
ADDI,R1 3 UPDATE BASE POINTER:C=1IN PSL
BRNR,R1 LOOP RETURN IF CONVERSION NOT READY
EXIT HALT END OF CONVERSION
END

```

FIGURE 2-2 Program Listing for Signed Binary-to-BCD Conversion

3. SIGNED BCD-TO-BINARY CONVERSION 1

FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

Addition of base numbers is used.

PARAMETERS:

Input: BCDD, BCDD+1, BCDD+2 contain the BCD number.

BCDD contains the sign plus the most-significant BCD digit.

The range of BCD numbers is: $-32768 < \text{BCD Number} < +32767$.

BCDD is destroyed after the conversion.

Output: BINN, BINN+1 contain the signed binary number.

BINN is the most-significant byte.

Refer to figures 3.1 and 3.2 for flowchart and program listing.

		HARDWARE AFFECTED							
REGISTERS		R0	R1	R2	R3	R1'	R2'	R3'	
PSU		F	II	SP					
PSL		CC	IDC	RS	WC	OVF	COM	C	
		X	X		X	X		X	

RAM REQUIRED (BYTES):	5
ROM REQUIRED (BYTES):	86
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

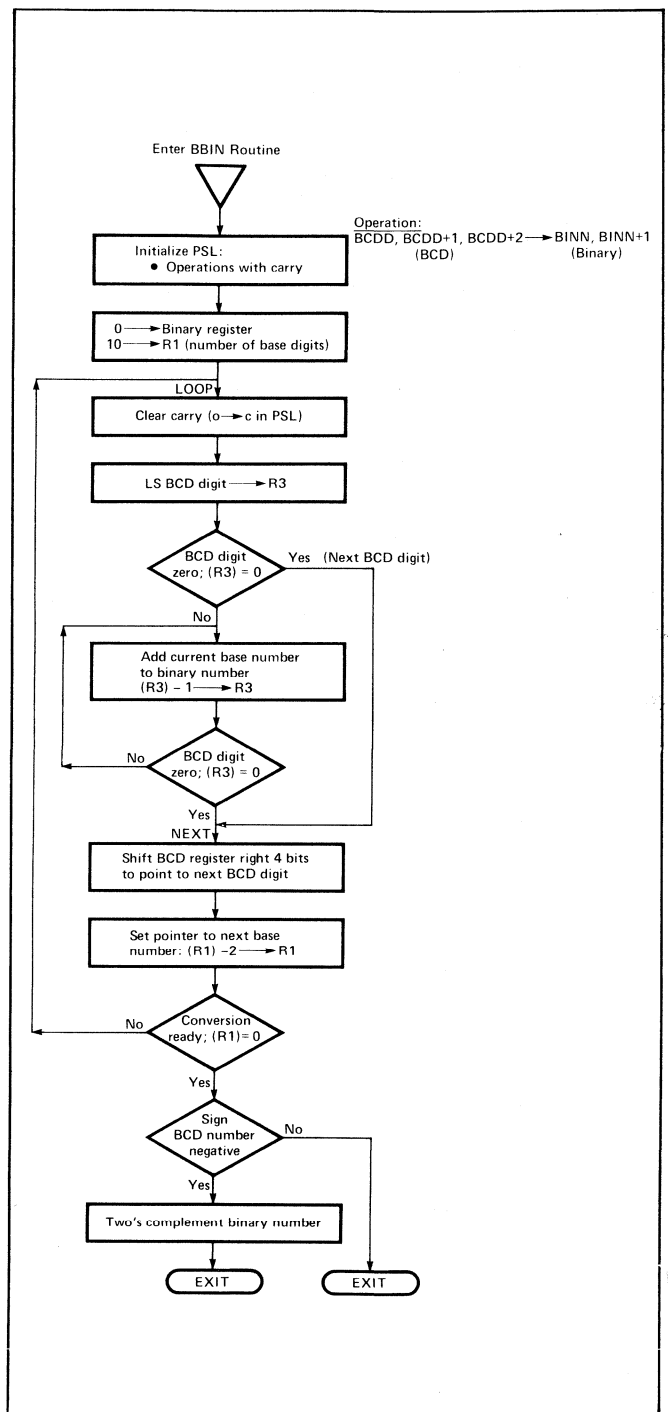


FIGURE 3-1: Flowchart for signed BCD-to-Binary Conversion

```

1          * PD760052
2          *+++++*****
3          * BCD TO BINARY CONVERSION
4          *+++++*****
5          *
6          * THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
7          * (24 BITS: SIGN+5 BCD DIGITS) INTO A SIGNED
8          * BINARY NUMBER (16 BITS).
9          * -32768 <BCD NUMBER <+32767
10         *   BCD NUMBER IS LOST AFTER CONVERSION.
11         *
12         * THE BINARY NUMBER IS IN BINN,BINN+1.
13         * THE BCD NUMBER IS IN BCDD,BCDD+1,BCDD+2 (R0-A4).
14         * THE BASE NUMBERS ARE IN BASE,- -BASE+9 (R0,R4).
15         * BINN AND BCDD ARE MOST SIGNIFICANT BYTES.
16         *
17         * PRINCIPLE OF CONVERSION IS:
18         * BINN = A0.R0+ A1.R1+ A2.R2+ A3.R3+ A4.R4
19         * A0 -A4 = NUMBER OF DIGITS OF BCD NUMBER.
20         * R0 -R4 = BASE NUMBERS FOR CONVERSION.
21         *
22         * DEFINITIONS OF SYMBOLS:
23         0000 R0 EQU 0          PROCESSOR-REGISTERS
24         0001 R1 EQU 1
25         0002 R2 EQU 2
26         0003 R3 EQU 3
27         0008 WC EQU H'08'      PSL: 1=WITH, 0=WITHOUT CARRY
28         0001 C EQU H'01'      CARRY: BORROW
29         0000 Z EQU 0          BRANCH COND: ZERO
30         0000 ON EQU 0         ALL BITS ARE 1
31         0008 SIGN EQU H'08'   TO TEST BCD. NUMBER
32         000A LEN EQU 10       INDEX NUMBER (LENGTH BASE REG)
33         *
34         *   ORG H'600'
35         *
36         0600 BINN RES 2        BINARY NUMBER
37         0602 BCDD RES 3        BCD NUMBER
38         0605 0605 27 10      BASE DATA H'27,10' 10000
39         0607 0607 03 E8      DATA H'03,E8' 1000
40         0609 0609 00 64      DATA H'00,64' 100
41         060B 060B 00 0A      DATA H'00,0A' 10
42         060D 060D 00 01      DATA H'00,01' 1
43         *
44         0450 0450 77 08      ORG H'450' START OF PROGRAM
45         0452 0452 20          BBIN PPSL WC ARITHMETIC+ROTATE WITH CARRY
46         0453 0453 CC 06 00   EORZ R0 CLEAR R0
47         0456 0456 CC 06 01   STRA,R0 BINN CLEAR BINARY REGISTERS
48         0459 0459 05 0A      LODI,R1 LEN INDEX FOR BASE DIGITS
49         045B 045B 75 01      LOOP CPSL C CLEAR CARRY
50         045D 045D 0F 06 04   LODA,R3 BCDD+2 LOAD LS BCD DIGIT IN R3
51         0460 0460 47 0F      ANDI,R3 H'0F' CLEAR MS 4 BITS
52         0462 0462 18 11      BCTR,Z NEXT IF ZERO GO TO NEXT
53         0464 0464 06 02      LOC1 LODI,R2 2 LOAD INDEX
54         0466 0466 0E 46 00   LOC2 LODA,R0 BINN,R2,-
55         0469 0469 8D 46 05   ADDA,R0 BASE,R1,- ADD BASE DIGIT TO BIN. NUMBER
56         046C 046C CE 66 00   STRA,R0 BINN,R2
57         046F 046F 5A 75      BRNR,R2 LOC2
58         0471 0471 85 02      ADDI,R1 2 RESTORE BASE POINTER
59         0473 0473 FB 6F      BDRR,R3 LOC1 IF NOT READY RETURN TO LOC1
60         *
61         0475 0475 06 04      NEXT LODI,R2 4 BIT COUNT
62         0477 0477 07 FB      LP2 LODI,R3 -3 INDEX FOR BYTE COUNT
63         0479 0479 0F 65 05   LP1 LODA,R0 BCDD-256+3,R3 BCD DIGIT INTO R0
64         047C 047C 50          RRR,R0 CARRY (PREVIOUS LS BIT) -> MSB
65         047D 047D CF 65 05   STRA,R0 BCDD-256+3,R3 AND LS BIT -> CARRY.
66         0480 0480 DB 77      BIRR,R3 LP1 NEXT BCDD BYTE
67         0482 0482 75 01      CPSL C
68         0484 0484 FA 71      BDRR,R2 LP2 NEXT SHIFT OF BCD REG. BIT
69         0486 0486 F9 00      BDRR,R1 *+2 UPDATE BASE POINTER WITHOUT
70         0488 0488 F9 51      BDRR,R1 LOOP AFFECTING C FLAG IN PSL AND
71         * GO TO LOOP IF NOT READY
72         *
73         048A 048A F4 08      TMI,R0 SIGN
74         048C 048C 98 0D      BCF,ON EXIT IF SIGN POS. THEN READY.
75         048E 048E 77 01      COMP PPSL C CLEAR BORROW
76         0490 0490 06 02      LODI,R2 2 NUMBER OF DIGITS
77         0492 0492 20          LP3 EORZ R0 TWO'S COMPLEMENT BY
78         0493 0493 AE 46 00   SUBA,R0 BINN,R2,- SUBTRACTION FROM ZERO
79         0496 0496 CE 66 00   STRA,R0 BINN,R2
80         0499 0499 5A 77      BRNR,R2 LP3
81         *
82         049B 049B 40          EXIT HALT END OF CONVERSION
83         *   END

```

FIGURE 3-2 Program Listing for Signed BCD-to-Binary Conversion

4. SIGNED BCD-TO-BINARY CONVERSION 2

FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

A multiplication method is used.

PARAMETERS:

Input: BCDD, BCDD+1 contain the BCD number.
 BCDD contains the sign plus the most-significant BCD digit.
 The range of BCD numbers is: $-32768 < \text{BCD Number} < +32767$

Output: BINN, BINN+1 contain the signed binary number.
 BINN is the most-significant byte.

Refer to figures 4.1 and 4.2 for flowchart and program listing.

HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

RAM REQUIRED (BYTES):	6
ROM REQUIRED (BYTES):	87
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

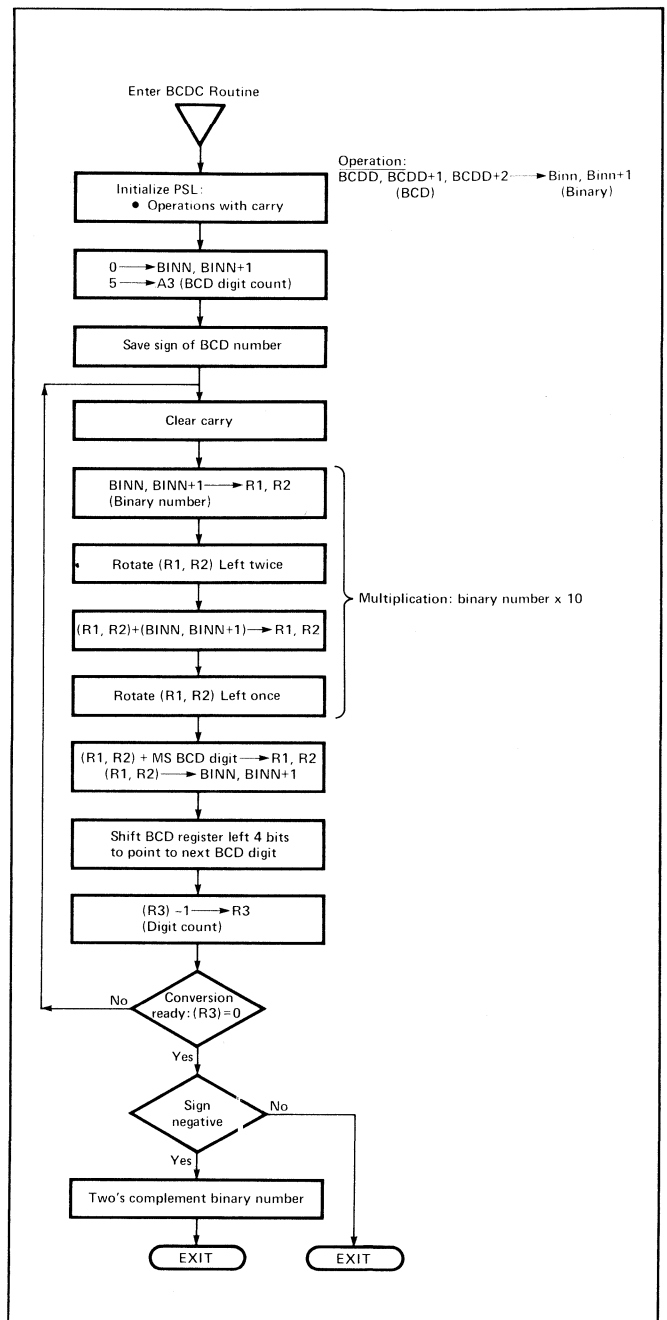


FIGURE 4-1: Flowchart for signed BCD-to-Binary Conversion (Multiplication Method).

```

1          * PD760053
2          *+++++*****
3          * BCD TO BINARY CONVERSION
4          *+++++*****
5          *
6          * THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
7          * (24 BITS: SIGN + 5 BCD DIGITS) INTO A SIGNED
8          * BINARY NUMBER (16 BITS).
9          * -32768 < BCD NUMBER < +32767
10         * BCD NUMBER IS LOST AFTER CONVERSION
11         *
12         * PRINCIPLE:
13         * BIN.=((((A*10)+B)*10)+C)*10)+D)*10)+E
14         * ABCDE= BCD NUMBER
15         *
16         * MULTIPLICATION BY 10 IS DONE BY:
17         * LOAD R2,R1 WITH BIN. NUMBER, SHIFT LEFT TWICE,
18         * ADD BIN. NUMBER TO R2,R1, SHIFT LEFT ONCE,
19         * STORE R2,R1 IN BINN,BINN+1 AS RESULT
20         *
21         * DEFINITION OF SYMBOLS:
22         0000 R0 EQU 0 PROCESSOR-REGISTERS
23         0001 R1 EQU 1
24         0002 R2 EQU 2
25         0003 R3 EQU 3
26         0008 WC EQU H'08' PSL: 1=WITH, 0=WITHOUT CARRY
27         0001 C EQU H'01' CARRY: BORROW
28         0002 N EQU 2 COND: NEGATIVE
29         0005 NUM EQU 5 INDEX FOR NUMBER OF BCD DIGITS
30         *
31         * ORG H'600'
32         *
33         0600 BINN RES 2 BINARY NUMBER
34         0602 BCDD RES 3 BCD NUMBER AND SIGN
35         0605 SIGN RES 1 SAVE SIGN DIGIT
36         *
37         *
38         *
39         * ORG H'450' START OF PROGRAM
40         *
41         0450 0450 77 00 BCDC PPSL WC ARITH.: ROTATE WITH CARRY
42         0452 20 00 EORZ R0 CLEAR R0
43         0453 CC 06 00 STRA,R0 BINN CLEAR BINARY NUMBERS
44         0456 CC 06 01 STRA,R0 BINN+1
45         0459 07 05 LODI,R3 NUM BCD INDEX REGISTER
46         *
47         045B 0C 06 02 LODA,R0 BCDD SAVE SIGN OF BCD NUMBER IN
48         045E CC 06 05 STRA,R0 SIGN MEMORY LOC. SIGN
49         *
50         * MULTIPLY BINARY NUMBER BY 10
51         0461 0461 75 01 LOOP CPSL C CLEAR CARRY
52         0463 0D 06 00 LODA,R1 BINN LOAD BIN. NUMBER IN R1,R2
53         0466 0E 06 01 LODA,R2 BINN+1
54         0469 D2 RRL,R2 ROTATE REGISTERS R1,R2 LEFT 2
55         046A D1 RRL,R1
56         046B D2 RRL,R2
57         046C D1 RRL,R1
58         046D 0E 06 01 ADDA,R2 BINN+1 ADD BIN. NUMBER TO R1,R2
59         0470 0D 06 00 ADDA,R1 BINN
60         0473 D2 RRL,R2 SHIFT R1,R2 LEFT ONCE
61         0474 D1 RRL,R1
62         *
63         *
64         0475 0C 06 02 LODA,R0 BCDD LOAD MS BCD DIGIT IN R0
65         0478 44 0F ANDI,R0 H'0F' CLEAR MS 4 BITS
66         047A 82 ADDZ R2 ADD BCD TO BINARY NUMBER
67         047B 85 00 ADDI,R1 0 ADD CARRY TO MS BYTE
68         047D CD 06 00 STRA,R1 BINN STORE RESULT IN BINN,BINN+1
69         0480 CC 06 01 STRA,R0 BINN+1
70         *
71         * ROTATE BCD NUMBER 4 TIMES LEFT
72         * TO POINT TO NEXT BCD DIGIT
73         0483 05 04 LODI,R1 4 BIT COUNT
74         0485 0485 06 03 LP2 LODI,R2 3 INDEX FOR BYTE COUNT
75         0487 0487 0E 46 02 LP1 LODA,R0 BCDD,R2,-
76         048A 00 RRL,R0 SHIFT BCD BYTE LEFT
77         048B CE 66 02 STRA,R0 BCDD,R2
78         048E 5A 77 BRNR,R2 LP1 NEXT BYTE OF BCD REGISTER
79         0490 F9 73 BDRR,R1 LP2 NEXT BIT SHIFT
80         0492 FB 4D BDRR,R3 LOOP TO LOOP IF MULTIPLY NOT READY
81         *
82         *
83         0494 0C 06 05 LODA,R0 SIGN
84         0497 9A 0D BCFR,N EXIT IF SIGN POS. THEN READY
85         0499 77 01 PPSL C CLEAR CARRY
86         049B 06 02 LODI,R2 2 INDEX LOADING
87         049D 049D 20 LP3 EORZ R0 TWO'S COMPLEMENT BY
88         049E AE 66 00 SUBA,R0 BINN,R2 SUBTRACTING FROM ZERO
89         04A1 CC 06 00 STRA,R0 BINN
90         04A4 5A 77 BRNR,R2 LP3
91         *
92         *
93         04A6 04A6 40 EXIT HALT END OF CONVERSION
94         *
95         *
96         *
97         *
98         *
99         *
100        *
101        *
102        *
103        *
104        *
105        *
106        *
107        *
108        *
109        *
110        *
111        *
112        *
113        *
114        *
115        *
116        *
117        *
118        *
119        *
120        *
121        *
122        *
123        *
124        *
125        *
126        *
127        *
128        *
129        *
130        *
131        *
132        *
133        *
134        *
135        *
136        *
137        *
138        *
139        *
140        *
141        *
142        *
143        *
144        *
145        *
146        *
147        *
148        *
149        *
150        *
151        *
152        *
153        *
154        *
155        *
156        *
157        *
158        *
159        *
160        *
161        *
162        *
163        *
164        *
165        *
166        *
167        *
168        *
169        *
170        *
171        *
172        *
173        *
174        *
175        *
176        *
177        *
178        *
179        *
180        *
181        *
182        *
183        *
184        *
185        *
186        *
187        *
188        *
189        *
190        *
191        *
192        *
193        *
194        *
195        *
196        *
197        *
198        *
199        *
200        *
201        *
202        *
203        *
204        *
205        *
206        *
207        *
208        *
209        *
210        *
211        *
212        *
213        *
214        *
215        *
216        *
217        *
218        *
219        *
220        *
221        *
222        *
223        *
224        *
225        *
226        *
227        *
228        *
229        *
230        *
231        *
232        *
233        *
234        *
235        *
236        *
237        *
238        *
239        *
240        *
241        *
242        *
243        *
244        *
245        *
246        *
247        *
248        *
249        *
250        *
251        *
252        *
253        *
254        *
255        *
256        *
257        *
258        *
259        *
260        *
261        *
262        *
263        *
264        *
265        *
266        *
267        *
268        *
269        *
270        *
271        *
272        *
273        *
274        *
275        *
276        *
277        *
278        *
279        *
280        *
281        *
282        *
283        *
284        *
285        *
286        *
287        *
288        *
289        *
290        *
291        *
292        *
293        *
294        *
295        *
296        *
297        *
298        *
299        *
300        *
301        *
302        *
303        *
304        *
305        *
306        *
307        *
308        *
309        *
310        *
311        *
312        *
313        *
314        *
315        *
316        *
317        *
318        *
319        *
320        *
321        *
322        *
323        *
324        *
325        *
326        *
327        *
328        *
329        *
330        *
331        *
332        *
333        *
334        *
335        *
336        *
337        *
338        *
339        *
340        *
341        *
342        *
343        *
344        *
345        *
346        *
347        *
348        *
349        *
350        *
351        *
352        *
353        *
354        *
355        *
356        *
357        *
358        *
359        *
360        *
361        *
362        *
363        *
364        *
365        *
366        *
367        *
368        *
369        *
370        *
371        *
372        *
373        *
374        *
375        *
376        *
377        *
378        *
379        *
380        *
381        *
382        *
383        *
384        *
385        *
386        *
387        *
388        *
389        *
390        *
391        *
392        *
393        *
394        *
395        *
396        *
397        *
398        *
399        *
400        *
401        *
402        *
403        *
404        *
405        *
406        *
407        *
408        *
409        *
410        *
411        *
412        *
413        *
414        *
415        *
416        *
417        *
418        *
419        *
420        *
421        *
422        *
423        *
424        *
425        *
426        *
427        *
428        *
429        *
430        *
431        *
432        *
433        *
434        *
435        *
436        *
437        *
438        *
439        *
440        *
441        *
442        *
443        *
444        *
445        *
446        *
447        *
448        *
449        *
450        *
451        *
452        *
453        *
454        *
455        *
456        *
457        *
458        *
459        *
460        *
461        *
462        *
463        *
464        *
465        *
466        *
467        *
468        *
469        *
470        *
471        *
472        *
473        *
474        *
475        *
476        *
477        *
478        *
479        *
480        *
481        *
482        *
483        *
484        *
485        *
486        *
487        *
488        *
489        *
490        *
491        *
492        *
493        *
494        *
495        *
496        *
497        *
498        *
499        *
500        *
501        *
502        *
503        *
504        *
505        *
506        *
507        *
508        *
509        *
510        *
511        *
512        *
513        *
514        *
515        *
516        *
517        *
518        *
519        *
520        *
521        *
522        *
523        *
524        *
525        *
526        *
527        *
528        *
529        *
530        *
531        *
532        *
533        *
534        *
535        *
536        *
537        *
538        *
539        *
540        *
541        *
542        *
543        *
544        *
545        *
546        *
547        *
548        *
549        *
550        *
551        *
552        *
553        *
554        *
555        *
556        *
557        *
558        *
559        *
560        *
561        *
562        *
563        *
564        *
565        *
566        *
567        *
568        *
569        *
570        *
571        *
572        *
573        *
574        *
575        *
576        *
577        *
578        *
579        *
580        *
581        *
582        *
583        *
584        *
585        *
586        *
587        *
588        *
589        *
590        *
591        *
592        *
593        *
594        *
595        *
596        *
597        *
598        *
599        *
600        *
601        *
602        *
603        *
604        *
605        *
606        *
607        *
608        *
609        *
610        *
611        *
612        *
613        *
614        *
615        *
616        *
617        *
618        *
619        *
620        *
621        *
622        *
623        *
624        *
625        *
626        *
627        *
628        *
629        *
630        *
631        *
632        *
633        *
634        *
635        *
636        *
637        *
638        *
639        *
640        *
641        *
642        *
643        *
644        *
645        *
646        *
647        *
648        *
649        *
650        *
651        *
652        *
653        *
654        *
655        *
656        *
657        *
658        *
659        *
660        *
661        *
662        *
663        *
664        *
665        *
666        *
667        *
668        *
669        *
670        *
671        *
672        *
673        *
674        *
675        *
676        *
677        *
678        *
679        *
680        *
681        *
682        *
683        *
684        *
685        *
686        *
687        *
688        *
689        *
690        *
691        *
692        *
693        *
694        *
695        *
696        *
697        *
698        *
699        *
700        *
701        *
702        *
703        *
704        *
705        *
706        *
707        *
708        *
709        *
710        *
711        *
712        *
713        *
714        *
715        *
716        *
717        *
718        *
719        *
720        *
721        *
722        *
723        *
724        *
725        *
726        *
727        *
728        *
729        *
730        *
731        *
732        *
733        *
734        *
735        *
736        *
737        *
738        *
739        *
740        *
741        *
742        *
743        *
744        *
745        *
746        *
747        *
748        *
749        *
750        *
751        *
752        *
753        *
754        *
755        *
756        *
757        *
758        *
759        *
760        *
761        *
762        *
763        *
764        *
765        *
766        *
767        *
768        *
769        *
770        *
771        *
772        *
773        *
774        *
775        *
776        *
777        *
778        *
779        *
780        *
781        *
782        *
783        *
784        *
785        *
786        *
787        *
788        *
789        *
790        *
791        *
792        *
793        *
794        *
795        *
796        *
797        *
798        *
799        *
800        *
801        *
802        *
803        *
804        *
805        *
806        *
807        *
808        *
809        *
810        *
811        *
812        *
813        *
814        *
815        *
816        *
817        *
818        *
819        *
820        *
821        *
822        *
823        *
824        *
825        *
826        *
827        *
828        *
829        *
830        *
831        *
832        *
833        *
834        *
835        *
836        *
837        *
838        *
839        *
840        *
841        *
842        *
843        *
844        *
845        *
846        *
847        *
848        *
849        *
850        *
851        *
852        *
853        *
854        *
855        *
856        *
857        *
858        *
859        *
860        *
861        *
862        *
863        *
864        *
865        *
866        *
867        *
868        *
869        *
870        *
871        *
872        *
873        *
874        *
875        *
876        *
877        *
878        *
879        *
880        *
881        *
882        *
883        *
884        *
885        *
886        *
887        *
888        *
889        *
890        *
891        *
892        *
893        *
894        *
895        *
896        *
897        *
898        *
899        *
900        *
901        *
902        *
903        *
904        *
905        *
906        *
907        *
908        *
909        *
910        *
911        *
912        *
913        *
914        *
915        *
916        *
917        *
918        *
919        *
920        *
921        *
922        *
923        *
924        *
925        *
926        *
927        *
928        *
929        *
930        *
931        *
932        *
933        *
934        *
935        *
936        *
937        *
938        *
939        *
940        *
941        *
942        *
943        *
944        *
945        *
946        *
947        *
948        *
949        *
950        *
951        *
952        *
953        *
954        *
955        *
956        *
957        *
958        *
959        *
960        *
961        *
962        *
963        *
964        *
965        *
966        *
967        *
968        *
969        *
970        *
971        *
972        *
973        *
974        *
975        *
976        *
977        *
978        *
979        *
980        *
981        *
982        *
983        *
984        *
985        *
986        *
987        *
988        *
989        *
990        *
991        *
992        *
993        *
994        *
995        *
996        *
997        *
998        *
999        *
1000       *

```

FIGURE 4-2 Program Listing for Signed BCD-to-Binary Conversion

5. SIGNED BCD-TO-ASCII CONVERSION

FUNCTION:

Converts n BCD digits plus sign to n + 1 ASCII characters (sign included).

PARAMETERS:

Input: BCDD, BCDD+1, ----- BCDD+ (numb - 1)
 BCDD contains the sign plus the most-significant digit (2 BCD digits/byte).
 Numb is the number of BCD bytes.

Output: ASCII, ASCII+1, -----, ASCII + (num - 1) contains the signed result.
 ASCII contains the sign.
 ASCII+1 contains the most-significant byte.

Refer to figures 5.1 and 5.2 for flowchart and program listing.

HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

RAM REQUIRED (BYTES):	N Numb, N Num+1
ROM REQUIRED (BYTES):	53
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

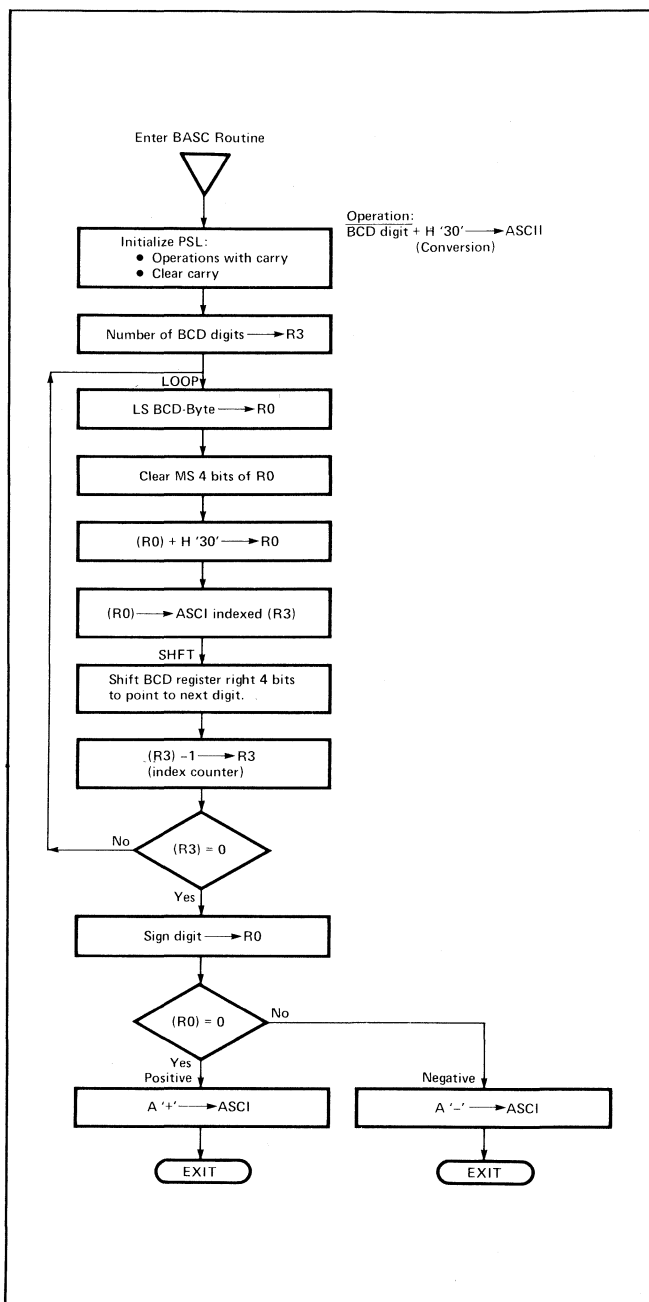


FIGURE 5-1 Flowchart for BCD-to-ASCII Conversion (signed)

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

```

*
* PD760054
*+++++
* BCD TO ASCII CONVERSION *
*+++++
*
* THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
* INTO ASCII CHARACTERS (SIGN INCLUDED).
* BCD FORMAT: SIGN + BCD DIGITS (TWO DIGITS:BYTES)
* THE NUMBER OF BCD DIGITS -> R3 = NUM
* THE NUMBER OF BCD BYTES -> R2 = NUMB
* BCD NUMBER IS IN BCDD,BCDD+1,---,BCDD+(N-1)
* ASCII CHARACTERS ARE IN ASCII,ASCII+1,---,ASCII+NUM
* (SIGN) (BCD DIGITS)
*
* DEFINITIONS OF SYMBOLS:
*
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
WC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
C EQU H'01' CARRY: BORROW
UN EQU 3 COND: UNCONDITIONAL
Z EQU 0 ZERO
*
* IN THIS EXAMPLE THE CONVERSION OF 5 BCD DIGITS
* IS PERFORMED.
*
NUMB EQU 3 NUMBER OF BCD BYTES
NUM EQU 5 NUMBER OF BCD DIGITS
*
*
ORG H'4E0'
*
BCDD RES NUMB RESERVE FOR BCD NUMBER
ASCII RES NUM+1 RESERVE FOR SIGN,ASCII DIGITS
*
ORG H'500' PROGRAM START HERE
*
BASC PPSL WC ARITHMETIC: ROTATE WITH CARRY
CPSL C CLEAR CARRY
LODI,R3 NUM INDEX REGISTER
*
LOOP LODA,R0 BCDD+NUMB-1 LOAD LS BCD DIGIT IN R0
ANDI,R0 H'0F' CLEAR MS 4 BITS
ADDI,R0 H'30' ASCII CHARACTER
STRA,R0 ASCII,R3 STORE ASCII CHARACTER
*
SHFT LODI,R1 4 BIT COUNT
LP2 LODI,R2 -NUMB INDEX FOR BYTE SHIFT
LP1 LODA,R0 BCDD-256+NUMB,R2
RRR,R0 CARRY (PREVIOUS LS BIT) -> MSB
STRA,R0 BCDD-256+NUMB,R2 AND LS BIT -> CARRY
BIRR,R2 LP1
CPSL C CLEAR CARRY
BDRR,R1 LP2
BDRR,R3 LOOP IF NOT READY GO TO LOOP
*
SIGN LODA,R0 BCDD+NUMB-1 SIGN -> R0
BCTR,Z POS
NEG LODI,R0 A'-1
STRA,R0 ASCII
BCTR,UN EXIT
POS LODI,R0 A'+1
STRA,R0 ASCII
*
EXIT HALT END OF CONVERSION
END

```

FIGURE 5-2 Program Listing for BCD-to-ASCII Conversion (Signed)

6. ASCII-TO-BCD CONVERSION

FUNCTION:

Converts n ASCII digits to n BCD digits.
 ASCII → BCD

PARAMETERS:

Input: ADIG, ADIG+1, ..., ADIG+(n - 1) contain ASCII digits.
 The most-significant digit is in ADIG (byte/digit).

Output: BCDD, BCDD+1, ..., BCDD + (n-1) contains BCD digits.
 The most-significant digit is in BCDD (2 digits/byte).

Refer to figures 6.1 and 6.2 for flowchart and program listing.

HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

RAM REQUIRED (BYTES):	nADIG + nBCDD
ROM REQUIRED (BYTES):	37
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

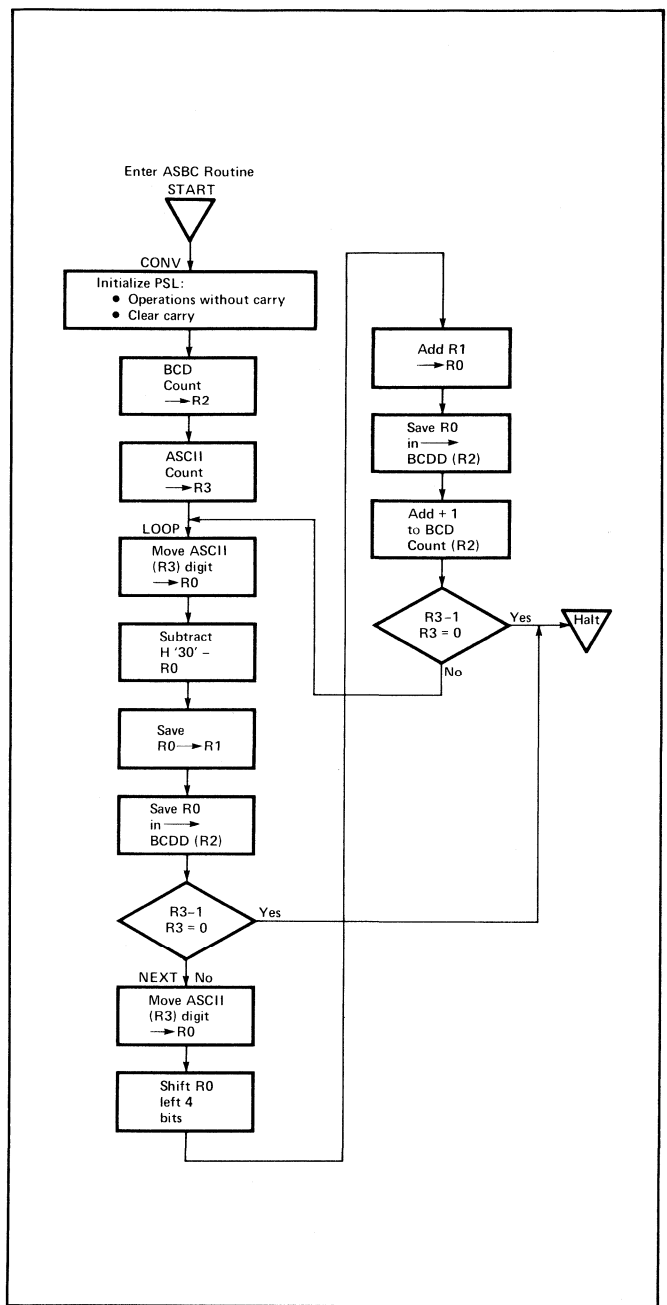


FIGURE 6-1 Flowchart for ASCII-to-BCD Conversion


```

1          * PD760055
2          ++++++
3          * ASCII TO BCD CONVERSION          +
4          ++++++
5          *
6          * THIS ROUTINE CONVERTS A STRING OF ASCII
7          * DIGITS TO A STRING OF BCD DIGITS.
8          *
9          * ADIG IS MS DIGIT ASCII
10         * BCDD IS MS DIGIT BCD
11         *
12         * DEFINITIONS OF SYMBOLS:
13 0000    R0 EQU 0          PROCESSOR-REGISTERS
14 0001    R1 EQU 1
15 0002    R2 EQU 2
16 0003    R3 EQU 3
17 0008    WC EQU H'08'    PSL: 1-WITH, 0-WITHOUT
18 0001    C EQU H'01'     CARRY: BORROW
19 0003    UN EQU 3        BR.COND: ALWAYS
20         *
21         * IN THIS EXAMPLE THE CONVERSION OF 5
22         * ASCII CHARACTERS IS PERFORMED.
23         *
24 0005    NUM EQU 5
25 0003    NUM1 EQU 3
26         *
27         *
28         * ORG H'750'    RAM DEFINITIONS
29 0750    ADIG RES NUM    ASCII BYTES RESERVED
30 0005    ACNT EQU $-ADIG ASCII DIGIT COUNT
31 0755    BCDD RES NUM1   BCD BYTES RESERVED
32 0003    BCNT EQU $-BCDD BCD BYTE COUNT
33         *
34         * ORG H'500'    START OF SUBROUTINE
35 0500 75 09    CONV CPSL WC+C    ARITH.WITHOUT:NO CARRY
36 0502 06 03    LODI,R2 BCNT      BCD COUNT -> R2
37 0504 07 05    LODI,R3 ACNT      ASCII COUNT ->R3
38 0506 0F 67 4F LOOP LODA,R0 ADIG-1,R3 R0 HAS ASCII DIGIT
39 0509 A4 30    SUBI,R0 H'30'     MAKE IT BCD
40 050B C1       STRZ R1           R0 ->R1
41 050C CE 67 54 STRA,R0 BCDD-1,R2 SAVE 1 BCD DIGIT
42 050F FB 03    BDRR,R3 NEXT      DECREMENT -NON ZERO BR
43 0511 1F 05 25 BCTA,UN BYE      CONVERSION COMPLETE
44 0514 0F 67 4F NEXT LODA,R0 ADIG-1,R3 NEXT ASCII DIGIT
45 0517 A4 30    SUBI,R0 H'30'     MAKE IT BCD
46 0519 D0       RRL,R0           SHIFT LEFT 4 BITS
47 051A D0       RRL,R0
48 051B D0       RRL,R0
49 051C D0       RRL,R0
50 051D 61       IORZ R1           INCLUSIVE OR LOW ORDER
51 051E CE 67 54 STRA,R0 BCDD-1,R2 STORE 2 BCD DIGITS
52 0521 FA 00    BDRR,R2 $+2      DECREMENT BCD COUNT
53 0523 FB 61    BDRR,R3 LOOP      DECREMENT-NON ZERO BR.
54 0525 40       BYE HALT         END OF ASCII -> BCD
55         END

```

FIGURE 6-2 Program Listing for ASCII-to-BCD Conversion

7. HEXADECIMAL-TO-ASCII CONVERSION

FUNCTION:

Converts a string of hexadecimal digits to a string of ASCII digits.

PARAMETERS:

Input: HEX, HEX+1, ..., HEX + (n - 1)
 HEX is the most-significant digit (2 Hex. digit/byte).

Output: ASCI, ASCI+1, ..., ASCI + (n - 1)
 ASCI is the most-significant digit.

Refer to figures 7.1 and 7.2 for flowchart and program listing.

HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1' 	R2' 	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	nHEX + nASCII
ROM REQUIRED (BYTES):	59
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

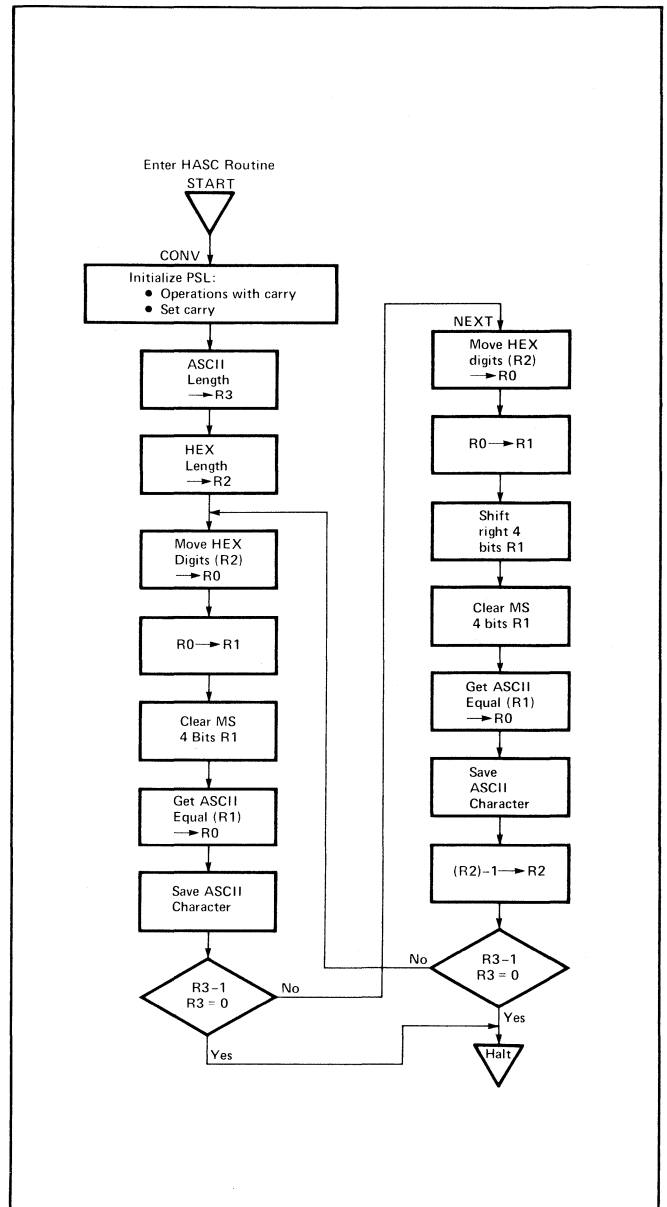


FIGURE 7-1 Flowchart for Hexadecimal-to-ASCII Conversion

```

1          * PD760056
2          ++++++
3          * HEXIDECIMAL TO ASCII CONVERSION +
4          ++++++
5          *
6          * THIS ROUTINE CONVERTS A STRING OF ASCII
7          * DIGITS TO A STRING OF HEX DIGITS.
8          *
9          * ASCII IS MS DIGIT ASCII.
10         * HEX IS MS DIGIT HEXIDECIMAL.
11         *
12         * DEFINITION OF SYMBOLS:
13 0000    R0 EQU 0 PROCESSOR-REGISTER
14 0001    R1 EQU 1
15 0002    R2 EQU 2
16 0003    R3 EQU 3
17 0008    WC EQU H'08' ARITHMETIC CARRY
18 0001    C EQU H'01' CARRY:BORROW
19 0003    UN EQU 3 UNCOND. BRANCH
20         *
21         * IN THIS EXAMPLE 3 HEXIDECIMAL
22         * CHARACTERS ARE CONVERTED.
23 0002    NUM EQU 2 HEX BYTE COUNT
24 0003    NUM1 EQU 3 ASCII BYTE COUNT
25         *
26         ORG H'600' RAM DEFINITIONS
27 0600    HEX RES NUM RESERVES HEX BYTES
28 0002    HLEN EQU $-HEX LENGTH OF HEX
29 0602    ASCII RES NUM1 RESERVES ASCII BYTES
30 0003    ALEN EQU $-ASCII LENGTH OF ASCII
31         *
32         ORG H'500' START OF ROUTINE
33 CONV PPSL WC+C ARITH.WITH; SET CARRY
34 0500 77 09 LODI,R3 ALEN R3= ASCII LENGTH
35 0502 07 03 LODI,R2 HLEN R2= HEX LENGTH
36 0504 06 02 CHEX LODA,R0 HEX-1,R2 GET HEX DIGITS
37 0506 0E 65 FF STRZ R1 R0 ->R1
38 0509 C1 ANDI,R1 H'0F' CLEAR MS 4 BITS
39 050A 45 0F LODA,R0 ANSI,R1 LOAD ASCII CORRESPONDI
40 050C 0D 65 2C STRA,R0 ANSI-1,R3 SAVE IT
41 050F CF 66 01 BDRR,R3 NEXT R3-1, R3(<) BRANCH
42 0512 FB 03 BCTA,UN BYE END OF CONVERSION
43 0514 1F 05 2B NEXT LODA,R0 HEX-1,R2 GET HEX DIGITS
44 0517 0E 65 FF STRZ R1 R0 -> R1
45 051A C1 RRR,R1 SHIFT RIGHT 4 BITS
46 051B 51 RRR,R1
47 051C 51 RRR,R1
48 051D 51 RRR,R1
49 051E 51 ANDI,R1 H'0F' CLEAR MS 4 BITS
50 051F 45 0F LODA,R0 ANSI,R1 LOAD ASCII CORRESPONDI
51 0521 0D 65 2C STRA,R0 ANSI-1,R3 SAVE IT
52 0524 CF 66 01 BDRR,R2 $+2 R2 - 1 CONT.
53 0527 FA 00 BDRR,R3 CHEX R3-1, R3(<) BRANCH
54 0529 FB 5B
55         *
56 052B 40 BYE HALT END OF CONVERSION
57         *
58 052C 30 31 32 33 ANSI DATA A'0123456789ABCDEF'
          34 35 36 37
          38 39 41 42
          43 44 45 46
59
60         *
          END

```

FIGURE 7-2 Program Listing for Hexadecimal-to-ASCII Conversion

8. ASCII-TO-HEXADECIMAL CONVERSION

FUNCTION:

Converts a string of ASCII digits to a string of hexadecimal digits. The conversion is done by table look-up. Non-numeric ASCII halts this routine. It may be changed to report non-numeric.

PARAMETERS:

Input: ASCII, ASCII+1, ..., ASCII + (n - 1)
 ASCII is the most-significant digit.

Output: HEX, HEX+1, ..., HEX + (n - 1)
 HEX is the most-significant digit (2 Hex. digits/byte)

Refer to figures 8.1 and 8.2 for flowchart and program listing.

HARDWARE AFFECTED								
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'	
	X	X	X	X				
PSU	F	II	SP					
PSL	CC	IDC	RS	WC	OVF	COM	C	
	X	X		X	X		X	

RAM REQUIRED (BYTES): nASCII + nHEX

ROM REQUIRED (BYTES): 68

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE
 NESTING LEVELS: 1

ASSEMBLER/COMPILER USED: PIPHASM

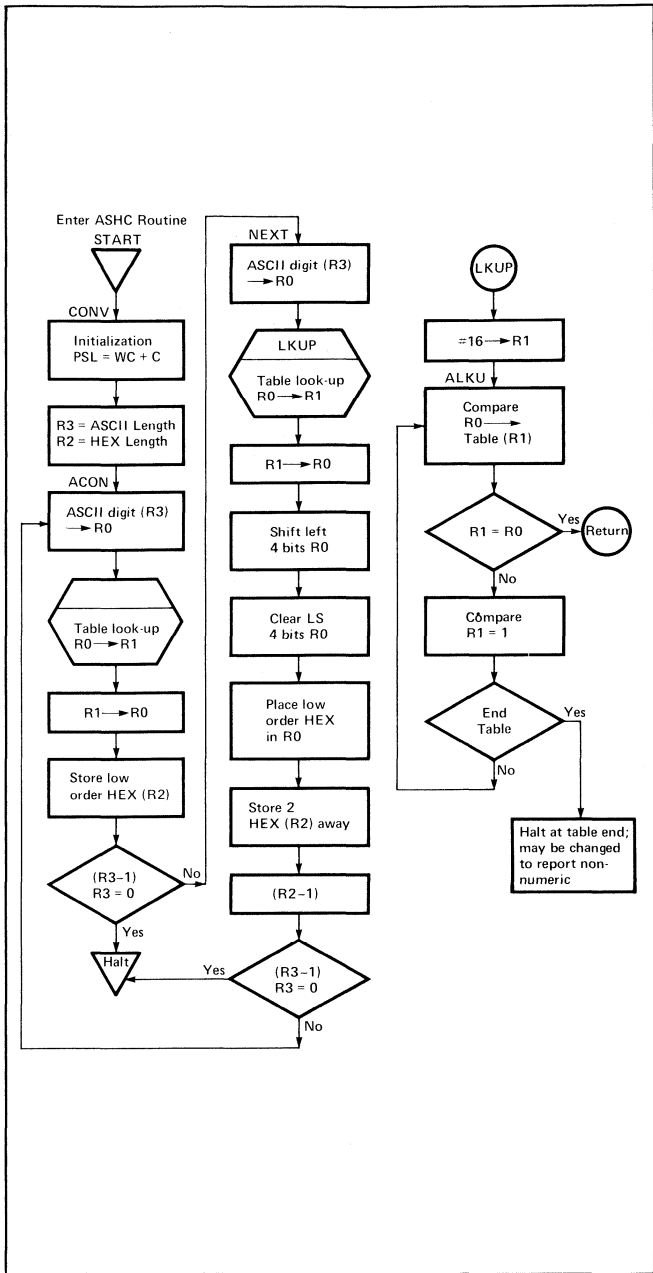


FIGURE 8-1 Flowchart for ASCII-to-Hexadecimal Conversion

```

1          * PD760057
2          ++++++
3          * ASCII TO HEX CONVERSION          *
4          ++++++
5          *
6          * THIS ROUTINE CONVERTS A STRING OF ASCII
7          * DIGITS TO A STRING OF HEXIDECIMAL DIGITS
8          * ASCII IS MS DIGIT ASCII
9          * HEX IS MS DIGIT HEXIDECIMAL
10         * CONVERSION DONE BY TABLE LOOKUP
11         * NON NUMERIC ASCII HALT ROUTINE
12         *
13         * DEFINITION OF SYMBOLS:
14 0000    R0 EQU 0          REGISTER-PROCESSOR
15 0001    R1 EQU 1
16 0002    R2 EQU 2
17 0003    R3 EQU 3
18 0008    WC EQU H'08'    ARITHMETIC CARRY
19 0001    C EQU H'01'    CARRY:BORROW
20 0003    UN EQU 3       BRANCH UNCOND.
21 0002    LT EQU 2       LESS THAN
22 0000    EQ EQU 0       EQUAL
23         *
24         * IN THIS EXAMPLE 3 ASCII DIGITS
25         * ARE CONVERTED TO HEXIDECIMAL
26         *
27 0003    NUM EQU 3       ASCII BYTE COUNT
28 0002    NUM1 EQU 2      HEX BYTE COUNT
29         *
30         *
31         *
32         ORG H'600'      RAM DEFINITIONS
33 0600    ASCII RES NUM   RESERVED ASCII BYTES
34 0003    ALEN EQU $-ASCII LENGTH OF ASCII
35 0603    HEX RES NUM1   RESERVED HEX BYTES
36 0002    HLEN EQU $-HEX LENGTH OF HEX
37         *
38         ORG H'500'      START OF ROUTINE
39 0500 77 09    CONV PPSL WC+C ARITH.WITH + CARRY SET
40 0502 07 03    LODI,R3 ALEN R3 = ASCII LENGTH
41 0504 06 02    LODI,R2 HLEN R2 = HEX LENGTH
42 0506 0F 65 FF ACON LODA,R0 ASCII-1,R3 GET ASCII DIGIT
43 0509 3B 08    BSTR,UN LKUP LOOKUP SUBROUTINE
44 050B 01       LODZ R1 R1 -> R0
45 050C CE 66 02 STRA,R0 HEX-1,R2 SAVE HEX CORRESPONDING
46 050F FB 1D    BDRR,R3 NEXT (R3-1), R3 (<) BRANCH
47 0511 1B 31    BCTR,UN BYE END OF CONVERSION
48         *
49 0513 05 10    LKUP LODI,R1 16 LOOP CONSTANT
50 0515 ED 45 1E ALKU COMA,R0 ANSI,R1,- COMPARE TO TABLE
51 0518 14       RETC,EQ RETURN - MATCH FOUND
52 0519 E5 01    COMI,R1 1 TEST END OF TABLE
53 051B 9A 78    BCFR,LT ALKU NO- LOOK AGAIN
54 051D 40       HALT ERROR - NON NUMERIC HE
55         *
56 051E 30 31 32 33 ANSI DATA A'0123456789ABCDEF'
57         *
58 052E 0F 65 FF NEXT LODA,R0 ASCII-1,R3 GET NEXT ASCII DIGIT
59 0531 3B 60    BSTR,UN LKUP LOOK UP SUBROUTINE
60 0533 01       LODZ R1 R1 -> R0
61 0534 D0       RRL,R0 SHIFT LEFT 4 BITS
62 0535 D0       RRL,R0
63 0536 D0       RRL,R0
64 0537 D0       RRL,R0
65 0538 44 F0    ANDI,R0 H'F0' CLEAR LS 4 BITS
66 053A 6E 66 02 IORA,R0 HEX-1,R2 COMBINE LOW ORDER
67 053D CE 66 02 STRA,R0 HEX-1,R2 SAVE 2 HEX DIGITS
68 0540 FA 00    BDRR,R2 $+2 (R2-1), CONTIUNE
69 0542 FB 42    BDRR,R3 ACON (R3-1), R3 (<) BRANCH
70         *
71 0544 40       BYE HALT END OF CONVERSION
72         END

```

FIGURE 8-2 Program Listing for ASCII-to-Hexadecimal Conversion

Signetics 2650 Microprocessor application memos currently available:

AS50	Serial Input/Output
AS51	Bit and Byte Testing Procedures
AS52	General Delay Routines
AS53	Binary Arithmetic Routines
AS54	Conversion Routines
SP50	2650 Evaluation Printed Circuit Board Level System (PC1001)
SP51	2650 Demo Systems
SP52	Support Software for use with NCSS Timesharing System
SP53	Simulator, Version 1.2
SP54	Support Software for use with the General Electric Mark III Timesharing System
SS50	PIPBUG
SS51	Absolute Object Format (Revision 1)
MP51	2650 Initialization
MP52	Low Cost Clock Generator Circuits
MP53	Address and Data Bus Interfacing Techniques



Electronic
components
and materials

PHILIPS

FIXED POINT DECIMAL

ARITHMETIC ROUTINES AS55

AN APPLICATION MEMO

signetics

INTRODUCTION

The numbers used in digital systems are usually expressed in binary notation. Some commonly used formats are:

- magnitudes only for unsigned numbers
- 1's complement and 2's complement for signed numbers.

However, binary numbers are difficult to interpret, and man-machine interface can be greatly improved by presenting numbers in decimal notation. Since virtually all digital systems operate on numbers in binary form (i.e., 1's and 0's), decimal numbers must be converted to binary during the input process, and reconverted to decimal notation during the output process. In cases where decimal input and/or output is required, the ideal solution would be a digital system capable of interpreting and processing decimal numbers.

This applications memo describes several methods of handling binary-coded-decimal (BCD) numbers with the Signetics 2650 microprocessor. Special provisions in the 2650 for these operations, including the Interdigit Carry (IDC) flag bit and the Decimal Adjust Register (DAR) instruction, are discussed. These provisions greatly simplify interfacing of the 2650 to decimal-oriented peripheral devices, such as CRT display terminals, printers, and keyboards. Basic arithmetic routines (add, subtract, multiply, and divide) for both signed integers and signed fixed-point numbers are given.

BCD NOTATION

In BCD notation, each decimal digit requires a 4-bit code as indicated below:

0 = 0000	5 = 0101
1 = 0001	6 = 0110
2 = 0010	7 = 0111
3 = 0011	8 = 1000
4 = 0100	9 = 1001

Codes 1010 through 1111 are not used.

Two decimal digits can be packed into one 8-bit byte—the size of a 2650 data word. The range within 1 byte is consequently 00₁₀ through 99₁₀. For instance, the number 15₁₀ is coded as 00010101.

CARRY (C) AND INTERDIGIT CARRY (IDC) FLAGS

The Program Status Lower (PSL) of the 2650's Program Status Word (PSW) register contains 2 carry flags: Carry (C) and Interdigit Carry (IDC). During execution of any arithmetic instruction, both flags are set or

reset depending on the result of the operation, as illustrated in Figure 1:

- The Carry (C) flag is set as a result of a carry (or no borrow) out of the most-significant-bit (bit 7) of the affected register Rx, and hence out of the most-significant BCD digit.
- The Interdigit Carry (IDC) flag is set as a result of a carry (or no borrow) out of bit 3, and hence out of the least-significant BCD digit.

DECIMAL ADJUST REGISTER (DAR) INSTRUCTION

If 2 BCD numbers are added or subtracted by means of binary arithmetic instructions, the result may not be a BCD number. For example:

$$23_{16} + 56_{16} = 79_{16};$$

but

$$18_{16} + 35_{16} = 4D_{16}.$$

Since the binary codes 1010 (A₁₆) through 1111 (F₁₆) are not used in BCD, the result of a binary arithmetic instruction may need a correction of (+6) in case of an add operation or (-6) in case of a subtract operation. The 2650 performs this correction by means of the Decimal Adjust Register (DAR) instruction. This 1-byte instruction condition-

ally adds a decimal 10 (2's complement negative 6 in a 4-bit binary number system) to either the high order 4-bits and/or the low order 4 bits of a specified register Rx, which may be any of the 2650's seven CPU registers.

The truth table of Figure 2 indicates the logical operation performed. The operation proceeds based on the values of the Carry (C) and Interdigit Carry (IDC) flags in the Program Status Word. The C and IDC remain unchanged by the execution of this instruction.

The WC (With/Without Carry) bit in PSL has no influence on the DAR instruction.

GENERAL SUBTRACTION RULES

In the case of subtraction, a correction of (-6) is required for the digit(s) which generate a borrow upon execution of the subtract instruction. This can be performed directly by the DAR instruction.

Single-Byte Operands/Result:

Subtraction of single-byte operands is done by performing the subtract instruction and then performing the DAR instruction; the borrow bit must be cleared initially. See Example A.

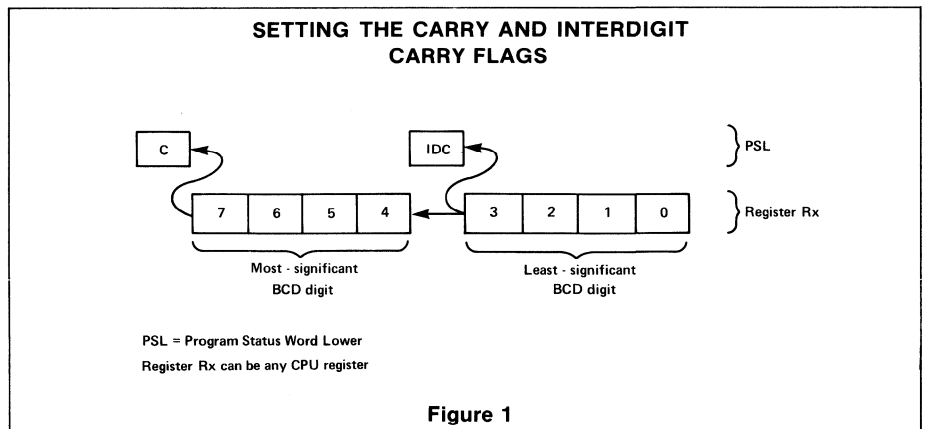


Figure 1

TRUTH TABLE FOR DAR INSTRUCTION

BEFORE: DAR, Rx				AFTER: DAR, Rx			
C	IDC	Rx		C	IDC	Rx	
		MSD	LSD			MSD	LSD
0	0	a	b	0	0	a+10 ₁₀	b+10 ₁₀
0	1	a	b	0	1	a+10 ₁₀	b
1	0	a	b	1	0	a	b+10 ₁₀
1	1	a	b	1	1	a	b

NOTE

IDC is not added to the upper digit in the 'a+10₁₀' operation.

Figure 2

If the With Carry (WC) bit in PSL is zero (no carry/borrow), the first instruction is not required.

Multiple-Byte Operands/Result:

When dealing with multiple-byte operands, arithmetic operations *including carry*, are required. Hence, the WC bit in PSL must be set to 1 prior to execution. If indexing is used, multiple-byte subtraction is simple, as illustrated in Example B.

NOTE: OPR1, OPR2 and RSLT are the most-significant bytes.

GENERAL ADDITION RULES

For addition, a correction of (+6) is required if the sum of the most-significant digits or least-significant digits exceeds 9. This is accomplished by first adding an offset of (+6) to each of the digits of the first operand (addition of H'66') and then adding the second operand.

If the sum of the least-significant digits did exceed 9, it now (including the (+6) correction) will exceed 15₁₀, (H'F'); an Interdigit Carry will be generated. If an IDC is generated, the result is correct and, as shown in Figure 2, the DAR instruction will have no effect on the sum. If not, the (+6) correction will be cancelled by adding 10 (equivalent to subtracting 6). Correction of the most-significant digit sum operates similarly, with the C bit controlling the final correction.

Single-Byte Operands/Result:

If the 2650 is conditioned for arithmetic without carry (WC = 0), addition can be performed as shown in Example C.

In the case of arithmetic with carry (WC = 1), it should be noted that the addition of the offset H'66' may generate a carry (if OPR1 = 99 and carry was set); this carry will be added during the addition of OPR2, giving an incorrect sum.

Multiple-Byte Operands/Result:

When using multiple-byte operands, linking of the bytes by means of the carry bit is required. Hence, arithmetic with carry must be performed (WC in PSL is set to 1). Because of the two successive additions (of the offset H'66' and of the second operand), the problem mentioned in the previous section can also arise here. Two straightforward solutions to this problem, listed below, are illustrated in the flowchart of Figure 3.

Method 1: In this method, each byte of the first operand is first increased by the offset H'66', after which addition of the second operand is performed. See Example D.

PPSL	C	CLEAR BORROW
LODA,R3	OPR1	FETCH FIRST OPERAND
SUBA,R3	OPR2	SUBTRACT SECOND OPERAND
DAR,R3		DECIMAL ADJUST RESULT
STRA,R3	RSLT	STORE RESULT

Example A

PPSL	WC+C	ARITHMETIC WITH CARRY, CLEAR BORROW
LODI,R3	LENG	LOAD INDEX REGISTER
DSUL LODA,R0	OPR1,R3,-	FETCH BYTE OF OPERAND1
SUBA,R0	OPR2,R3	SUBTRACT BYTE OF OPERAND2
DAR,R0		DECIMAL ADJUST RESULT
STRA,R0	RSLT,R3	STORE RESULTING BYTE
BRNR,R3	DSUL	CONTINUE LOOP IF NOT DONE

Example B

LODA,R3	OPR1	FETCH FIRST OPERAND
ADDI,R3	H'66'	ADD OFFSET FOR BCD ADD
ADDA,R3	OPR2	ADD SECOND OPERAND
DAR,R3		DECIMAL ADJUST RESULT
STRA,R3	RSLT	STORE RESULT

Example C

	CPSL	C	CLEAR CARRY
	PPSL	WC	ARITHMETIC WITH CARRY
	LODI,R3	LENG	LOAD INDEX REGISTER
ADD0	LODA,R0	OPR1,R3,-	FETCH BYTE OF OPERAND1
	ADDI,R0	H'66'	ADD OFFSET FOR BCD ADD
	STRA,R0	RSLT,R3	STORE INTERMEDIATE RESULT
	BRNR,R3	ADD0	BRANCH IF ALL BYTES NOT READY
	LODI,R3	LENG	LOAD INDEX REGISTER
ADD1	LODA,R0	RSLT,R3,-	FETCH BYTE OF INTERMEDIATE SUM
	ADDA,R0	OPR2,R3	ADD BYTE OF OPERAND2
	DAR,R0		DECIMAL ADJUST RESULT
	STRA,R0	RSLT,R3	STORE RESULT
	BRNR,R3	ADD1	BRANCH IF ALL BYTES NOT READY

Example D

Method 2: In this method, the complete addition is handled on a byte-by-byte basis. This means that the true interbyte-carry must be saved and restored, and the carry must be cleared at the appropriate time. This can be performed by using one additional register to retain the interbyte-carry. See Example E.

The second method is faster and requires fewer bytes of code (24 versus 30) but requires an additional register.

The program listing of Figure 5 summarizes the basic BCD addition and subtraction routines.

CPSL	C	CLEAR CARRY	
PPSL	WC	ARITHMETIC/ROTATE WITH CARRY	
LODI,R3	LENG	LOAD INDEX REGISTER	
LODI,R1	0	CLEAR INTERBYTE-CARRY REGISTER	
DADL	LODA,R0	OPR1,R3,-	FETCH BYTE OF OPERAND1
	ADDI,R0	H'66'	ADD OFFSET FOR BCD ADD
	RRR,R1		RESTORE INTERBYTE-CARRY TO CARRY
	ADDA,R0	OPR2,R3	ADD BYTE OF OPERAND2
	DAR,R0		DECIMAL ADJUST RESULT
	STRA,R0	RSLT,R3	STORE RESULT
	RRL,R1		SAVE INTERBYTE-CARRY IN R1, CLEAR C
	BRNR,R3	DADL	BRANCH IF NOT READY

Example E

ROUTINES FOR SIGNED INTEGER ARITHMETIC

There are several possible ways of representing signed decimal numbers. The best known are ten's complement notation and sign-magnitude notation. The sign-magnitude notation, illustrated in Figure 4, is used here because it is easy to interpret and lends itself to interfacing with peripherals. It is also simpler to use in multiplication, division, and in aligning and rounding routines. The numbers are stored in memory in the form of a sign followed by the absolute value of the number.

The length of the numbers is defined by the number of bytes (including the sign byte) they require. This parameter can be modified by changing the definition of LENG in the source program. Note that for clarity, each routine is written in a "stand-alone" form. If more than 1 routine is required in a program, considerable savings in the program space required can be realized by breaking out common operations as subroutines.

GENERAL ADDITION FOR MULTIPLE-BYTE, UNSIGNED BCD NUMBERS

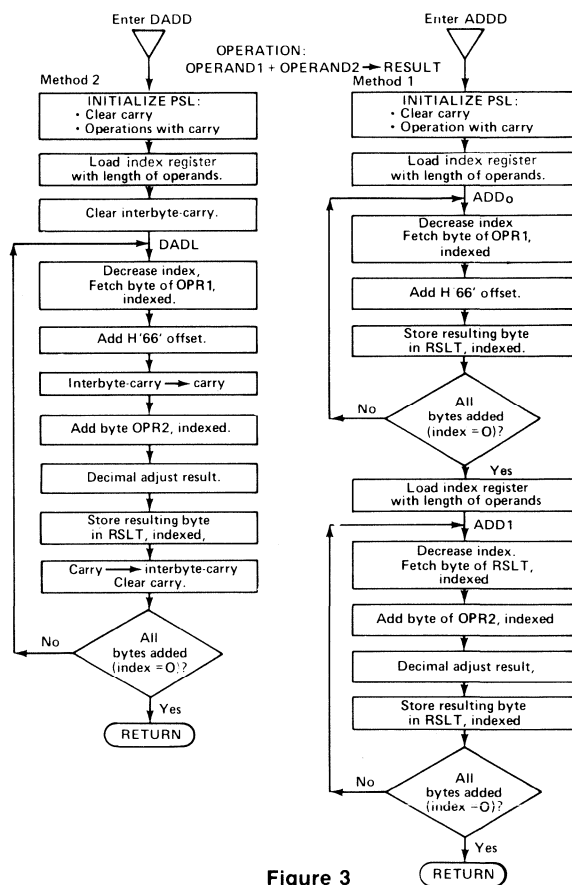


Figure 3

SIGN-MAGNITUDE NOTATION

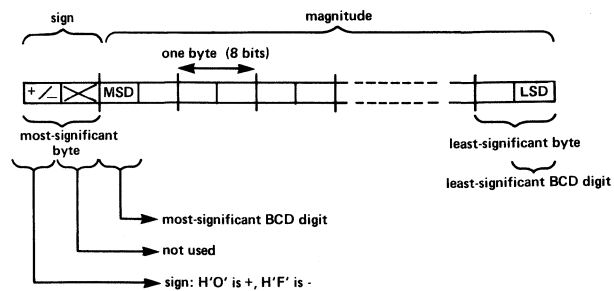


Figure 4

BCD ADDITION AND SUBTRACTION ROUTINES

```

TWIN ASSEMBLER VER 1.0                PAGE 0001
LINE ADDR OBJECT E SOURCE
0001          * PD760087
0002          * *****
0003          * DECIMAL ADDITION/SUBTRACTION FOR PACKED-BCD *
0004          * *****
0005          * OPERATION: OPERAND1 +/- OPERAND2 --> RESULT
0006          * OPERAND1 IS IN: OPR1,OPR1+1,OPR1+2,ETC.
0007          * OPERAND2 IS IN: OPR2,OPR2+1,OPR2+2,ETC.
0008          * RESULT IS IN: RSLT,RSLT+1,RSLT+2,ETC.
0009          * OPR1,OPR2 AND RSLT ARE MOST-SIGNIFICANT BYTES.
0010          * ALL NUMBERS ARE OF EQUAL LENGTH (IN BYTES).
0011          * LENGTH IS DEFINED BY: LENG
0012          *
0013          * DEFINITIONS OF SYMBOLS:
0014          *
0015 0000    R0 EQU 0          PROCESSOR REGISTERS
0016 0001    R1 EQU 1
0017 0002    R2 EQU 2
0018 0003    R3 EQU 3
0019 0008    MC EQU H'08'    PSL: 1-WITH, 0-WITHOUT CARRY
0020 0001    C EQU H'01'      CARRY/BORROW
0021 0003    UN EQU 3        BRANCH CONDITION: UNCONDITIONAL
0022          *
0023 0005    LENG EQU 5      LENGTH OF OPERANDS/RESULT IN BYTES
0024          *
0025 0000    ORG H'700'      PARAMETERS
0026          *
0027 0700    OPR1 RES LENG OPERAND1
0028 0705    OPR2 RES LENG OPERAND2
0029 070A    RSLT RES LENG RESULT
0030          *
0031 070F    ORG H'450'
0032          *
0033          * *****
0034          * ADDITION OF UNSIGNED, SINGLE-BYTE BCD NUMBERS *
0035          * *****
0036          * OPERATION: OPERAND1 + OPERAND2 --> RESULT
0037          *
0038 0450 0F0700  ADD LOAR,R3,OPR1  FETCH FIRST OPERAND
0039 0453 0766    ADDI,R3,H'66'     ADD OFFSET FOR BCD ADD
0040 0455 0F0705  ADDA,R3,OPR2     ADD SECOND OPERAND
0041 0458 97     DAR,R3            DECIMAL ADJUST RESULT
0042 0459 0F070A STRA,R3,RSLT  STORE RESULT
0043          *
0044          * *****
0045          * SUBTRACTION OF UNSIGNED, SINGLE-BYTE BCD NUMBERS *
0046          * *****
0047          * OPERATION: OPERAND1 - OPERAND2 --> RESULT
0048          *
0049 045C 0F0700  SUBT LOAR,R3,OPR1  FETCH FIRST OPERAND
0050 045F 0F0705  SUBA,R3,OPR2     SUBTRACT SECOND OPERAND
0051 0462 97     DAR,R3            DECIMAL ADJUST RESULT
0052 0463 0F070A STRA,R3,RSLT  STORE RESULT
0053          *
    
```

```

TWIN ASSEMBLER VER 1.0                PAGE 0002
LINE ADDR OBJECT E SOURCE
0055          * *****
0056          * ADDITION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0057          * *****
0058          * OPERATION: OPERAND1 + OPERAND2 --> RESULT
0059          *
0060 0466 7501    DADD CPSL C      CLEAR CARRY
0061 0468 7708    PPSL MC        ARITHMETIC/ROTATE WITH CARRY
0062 046A 0705    LOOI,R3 LENG   LOAD INDEX REGISTER
0063 046C 0500    LOOI,R1 0      CLEAR INTERBYTE-CARRY
0064 046E 0F4700  DADL LOAR,R0,OPR1,R3 - FETCH BYTE OF OPERAND1
0065 0471 0466    ADDI,R0,H'66'  ADD OFFSET FOR BCD ADD
0066 0473 51     RRR,R1         RESTORE INTERBYTE-CARRY TO C
0067 0474 0F6705  ADAR,R0,OPR2,R3  ADD BYTE OF OPERAND2
0068 0477 94     DAR,R0         DECIMAL ADJUST RESULT
0069 0478 0F670A STRA,R0,RSLT,R3  STORE RESULTING BYTE
0070 047B D1     RRL,R1         SAVE INTERBYTE-CARRY IN R1, CLEAR C
0071 047C 5B78    BRNR,R3,DADL  BRANCH IF NOT READY
0072          *
0073          * *****
0074          * ADDITION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0075          * ALTERNATE METHOD *
0076          * *****
0077          * OPERATION: OPERAND1 + OPERAND2 --> RESULT
0078          *
0079 047E 7501    ADDO CPSL C      CLEAR CARRY
0080 0480 7708    PPSL MC        ARITHMETIC WITH CARRY
0081 0482 0705    LOOI,R3 LENG   LOAD INDEX REGISTER
0082 0484 0F4700  ADDO LOAR,R0,OPR1,R3 - FETCH BYTE OF OPERAND1
0083 0487 0466    ADDI,R0,H'66'  ADD OFFSET FOR BCD-ADD
0084 0489 0F670A STRA,R0,RSLT,R3  STORE INTERMEDIATE RESULT
0085 048C 5B7E    BRNR,R3,ADD0   BRANCH IF ALL BYTES NOT READY
0086 048E 0705    LOOI,R3 LENG   LOAD INDEX REGISTER
0087 0490 0F470A  ADDI LOAR,R0,RSLT,R3 - FETCH BYTE OF INTERMEDIATE SUM
0088 0493 0F6705  ADAR,R0,OPR2,R3  ADD BYTE OF OPERAND2
0089 0496 94     DAR,R0         DECIMAL ADJUST RESULT
0090 0497 0F670A STRA,R0,RSLT,R3  STORE RESULT
0091 049A 5B74    BRNR,R3,ADD1   BRANCH IF ALL BYTES NOT READY
0092          *
0093          *
0094          * *****
0095          * SUBTRACTION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0096          * *****
0097          * OPERATION: OPERAND1 - OPERAND2 --> RESULT
0098          *
0099 049C 7709    DSUB PPSL MC+C   ARITHMETIC WITH CARRY, CLEAR BORROW
0100 049E 0705    LOOI,R3 LENG   LOAD INDEX REGISTER
0101 04A0 0F4700  DSUL LOAR,R0,OPR1,R3 - FETCH BYTE OF OPERAND1
0102 04A3 0F6705  SUBA,R0,OPR2,R3  SUBTRACT BYTE OF OPERAND2
0103 04A6 94     DAR,R0         DECIMAL ADJUST RESULT
0104 04A7 0F670A STRA,R0,RSLT,R3  STORE RESULTING BYTE
0105 04AA 5B74    BRNR,R3,DSUL  BRANCH IF NOT READY
0106          *
0107 0000          END 0
    
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 5

Program Title

DECIMAL ADDITION/SUBTRACTION FOR SIGNED INTEGERS (PACKED BCD)

Function

Addition or subtraction of 2 decimal integers in sign-magnitude notation. Operands and result are of equal length, as defined by LENG.

OPERAND1 +/- OPERAND2 → OPERAND2

Parameters

Input:

Length of numbers (in bytes) is defined by LENG.

OPR1, OPR1+1, OPR1+2, etc., contain augend or subtrahend.

OPR2, OPR2+1, OPR2+2, etc., contain addend or minuend.

Output:

OPR2, OPR2+1, OPR2+2, etc., contain sum or difference.

Overflow is detected.

OPERATION

Subtraction is performed by changing the sign of the second operand before entering the signed addition routine. Prior to adding or subtracting, the sign of the result must be determined. This requires a comparison of the magnitudes of both operands if they have opposite signs. In this case, the subtrahend and minuend for the operation are also designated by the comparison.

Refer to Figures 6 and 7 for flowchart and program listing.

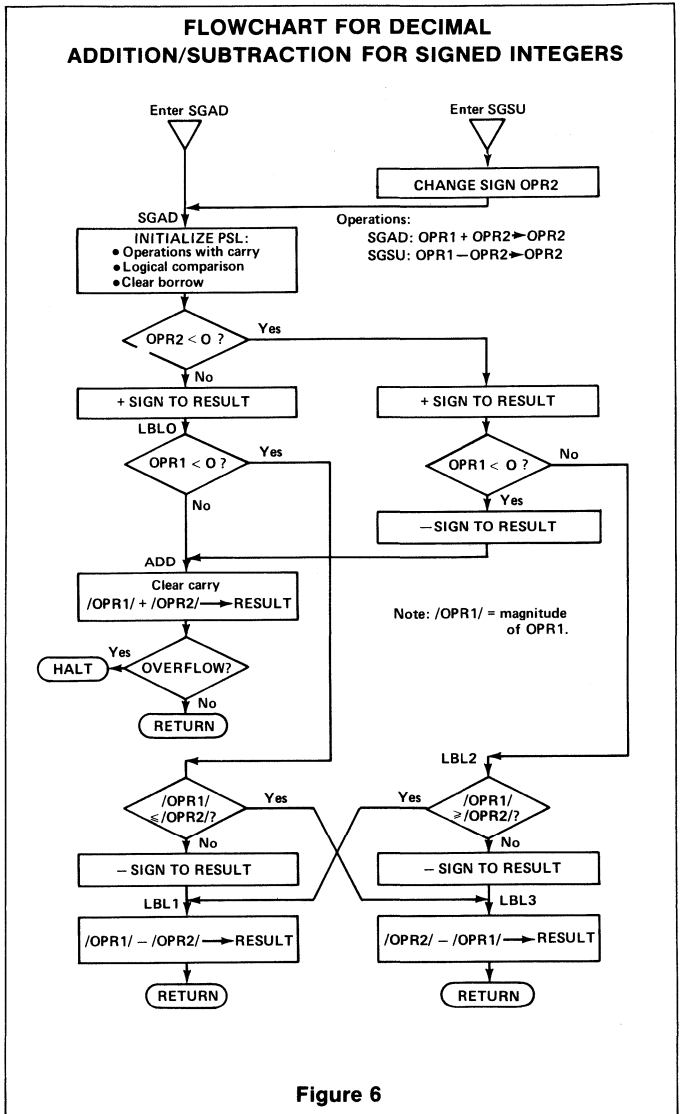


Figure 6

HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES):	2 X LENG
ROM REQUIRED (BYTES):	127
MAXIMUM SUBROUTINE NESTING LEVELS:	1
ASSEMBLER/COMPILER USED:	TWIN VER 1.0

DECIMAL ADDITION/SUBTRACTION FOR SIGNED INTEGERS

```
TWIN ASSEMBLER VER 1.0 PAGE 0001
LINE ADDR OBJECT E SOURCE
0001 * PD76006
0002 *****
0003 * DECIMAL ADDITION/SUBTRACTION FOR SIGNED-INTEGERS *
0004 * NUMBERS ARE IN PACKED BCD, SIGN-MAGNITUDE NOTATION *
0005 *****
0006 * OPERATION: OPRAND1 +/- OPRAND2 --> OPRAND2
0007 * OPRAND1 IS IN: OPR1, OPR1+1, OPR1+2, ETC.
0008 * OPRAND2 IS IN: OPR2, OPR2+1, OPR2+2, ETC.
0009 * SUM/DIFFERENCE IS IN: OPR2, OPR2+1, OPR2+2, ETC.
0010 * OPRAND2 IS DESTROYED AFTER ADD/SUBTRACT.
0011 * OPR1, OPR2 ARE MOST-SIGNIFICANT BYTES.
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG
0013 * ALLOWED RANGE: 1 < LENG < 255.
0014 * MS BYTE HOLDS SIGN INFORMATION: 'H'00' FOR +, 'H'F0' FOR -
0015 *
0016 * DEFINITIONS OF SYMBOLS
0017 *
0018 0000 R0 EQU 0 PROCESSOR REGISTERS
0019 0001 R1 EQU 1
0020 0002 R2 EQU 2
0021 0003 R3 EQU 3
0022 0008 WC EQU 'H'00' PSL: 1=WITH 0=WITHOUT CARRY
0023 0002 COM EQU 'H'02' 1=LOGIC, 0=ARITH COMPARE
0024 0001 C EQU 'H'01' CARRY/BORROW
0025 0000 Z EQU 0 BRANCH CONDITION: ZERO
0026 0002 N EQU 2 NEGATIVE
0027 0000 EQ EQU 0 EQUAL
0028 0001 GT EQU 1 GREATER THAN
0029 0002 LT EQU 2 LESS THAN
0030 0003 UN EQU 3 UNCONDITIONAL
0031 *
0032 * PARAMETERS *
0033 *
0034 0005 LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0035 *
0036 0000 ORG 'H'700'
0037 *
0038 0700 OPR1 RES LENG OPRAND1
0039 0705 OPR2 RES LENG OPRAND2/RESULT
0040 *
0041 070A ORG 'H'500'
0042 *
0043 *****
0044 * SUBROUTINE TO COMPARE OPRAND1 WITH OPRAND2 (UPDATE CC) *
0045 *****
0046 0500 0500 C012 LODI, R1, 0 CLEAR R1: MS BITS ARE USED TO SAVE CC DATA
0047 0502 0704 LODI, R3, LENG-1 LOAD INDEX REG
0048 0504 0F6700 COMB LODA, R0, OPR1, R3 FETCH BYTE OF OPRAND1
0049 0507 0F6705 COMA, R0, OPR2, R3 COMPARE WITH BYTE OF OPRAND2
0050 050A 1802 BCTR, EQ, COM1 BRANCH IF EQUAL
0051 050C 13 SPSL PSL TO R0
0052 050D C1 STRZ R1 SAVE PSL IN R1
0053 050E FE74 COM1 BARR, R3, COMB BRANCH IF ALL BYTES NOT TESTED
0054 0510 01 LODZ R1 UPDATE CC WITH STATUS COMPARE
0055 0511 17 RETC, UN RETURN
0056 *
```

```
TWIN ASSEMBLER VER 1.0 PAGE 0002
LINE ADDR OBJECT E SOURCE
0057 *
0058 *****
0059 * SUBTRACTION FOR SIGNED INTEGERS *
0060 *****
0061 0512 0C0705 S6SU LODA, R0, OPR2 FETCH SIGN OF OPRAND2
0062 0515 24F0 EOR1, R0, 'H'F0' CHANGE SIGN
0063 0517 0C0705 STRA, R0, OPR2 RESTORE SIGN OF OPRAND2
0064 *****
0065 * ADDITION FOR SIGNED INTEGERS *
0066 *****
```

```
0067 051A 7708 S6AD PPSL WC+COM+C OPERATIONS WITH CARRY,
0068 * LOGICAL COMPARE, CLEAR BORROW
0069 051C 20 EORZ R0 CLEAR R0
0070 051D 0D0705 LODA, R1, OPR2 FETCH SIGN OF OPRAND2
0071 0520 0C0705 STRA, R0, OPR2 CLEAR SIGN OF OPRAND2 (-RESULT)
0072 0523 9A23 BCFR, N, LBL0 BRANCH IF OPR2 NOT NEGATIVE
0073 0525 0C0700 LODA, R0, OPR1 FETCH SIGN OF OPRAND1
0074 0528 9A3C BCFR, N, LBL2 BRANCH IF OPR1 NOT NEGATIVE
0075 052A 04F0 LODI, R0, 'H'F0' FETCH MINUS SIGN
0076 052C 0C0705 STRA, R0, OPR2 STORE IN MS-BYTE RESULT
0077 *
0078 052F 7501 ADD CPSL, C OPR1 + OPR2 --> OPR2,
0079 * CLEAR CARRY.
0080 0531 0704 LODI, R3, LENG-1 LOAD INDEX REGISTER
0081 0533 0500 LODI, R1, 0 CLEAR INTERBYTE-CARRY
0082 0535 0F6700 ADD0 LODA, R0, OPR1, R3 FETCH BYTE OF OPRAND1
0083 0538 0466 ADDI, R0, 'H'66' ADD OFFSET
0084 053A 51 RRR, R1 INTERBYTE-CARRY TO CARRY
0085 053B 0F6705 ADDA, R0, OPR2, R3 ADD BYTE OF OPRAND2
0086 053E 94 DAR, R0 DECIMAL ADJUST RESULT
0087 053F CF6705 STRA, R0, OPR2, R3 STORE RESULTING BYTE
0088 0542 D1 RRL, R1 CARRY (-INTERBYTE-CARRY) TO R1
0089 * CLEAR CARRY.
0090 0543 FB70 BARR, R3, ADD0 BRANCH IF NOT READY
0091 0545 9038 BCFR, Z, OVFL BRANCH IF OVERFLOW
0092 0547 17 RETC, UN RETURN
0093 *
```

```
TWIN ASSEMBLER VER 1.0 PAGE 0003
LINE ADDR OBJECT E SOURCE
0095 0548 0C0700 LBL0 LODA, R0, OPR1 FETCH SIGN OF OPRAND1
0096 0548 9A62 BCFR, N, ADD BRANCH IF OPR1 NOT NEGATIVE
0097 054D 3F0500 BSTR, UN, C012 COMPARE OPR1 WITH OPR2,
0098 * (MAGNITUDES ONLY)
0099 0550 991E BCFR, GT, LBL3 BRANCH IF OPR1 < OR = TO OPR2
0100 0552 04F0 LODI, R0, 'H'F0' FETCH MINUS SIGN
0101 0554 0C0705 STRA, R0, OPR2 STORE IN MS-BYTE RESULT
0102 *
0103 * OPR1 - OPR2 --> OPR2
0104 0557 0704 LBL1 LODI, R3, LENG-1 LOAD INDEX REGISTER
0105 0559 0F6700 SU12 LODA, R0, OPR1, R3 FETCH BYTE OF OPRAND1
0106 055C 0F6705 SUBA, R0, OPR2, R3 SUBTRACT BYTE OF OPRAND2
0107 055F 94 DAR, R0 DECIMAL ADJUST RESULT
0108 0560 CF6705 STRA, R0, OPR2, R3 STORE RESULTING BYTE IN OPR2
0109 0563 FB74 BARR, R3, SU12 BRANCH IF NOT READY
0110 0565 17 RETC, UN RETURN
0111 *
0112 0566 3F0500 LBL2 BSTR, UN, C012 COMPARE OPR1 WITH OPR2,
0113 * (MAGNITUDES ONLY)
0114 0569 9A6C BCFR, LT, LBL1 BRANCH IF OPR1 > OR = OPR2
0115 056B 04F0 LODI, R0, 'H'F0' FETCH MINUS SIGN
0116 056D 0C0705 STRA, R0, OPR2 STORE IN MS-BYTE OF RESULT
0117 *
0118 * OPR2 - OPR1 --> OPR2
0119 0570 0704 LBL3 LODI, R3, LENG-1 LOAD INDEX REGISTER
0120 0572 0F6705 SU21 LODA, R0, OPR2, R3 FETCH BYTE OF OPRAND2
0121 0575 0F6700 SUBA, R0, OPR1, R3 SUBTRACT BYTE OF OPRAND1
0122 0578 94 DAR, R0 DECIMAL ADJUST RESULT
0123 0579 CF6705 STRA, R0, OPR2, R3 STORE RESULTING BYTE
0124 057C FB74 BARR, R3, SU21 BRANCH IF NOT READY
0125 057E 17 RETC, UN RETURN
0126 *
0127 057F 40 OVFL, HALT ARITHMETIC OVERFLOW
0128 *
0129 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000
```

Figure 7

Program Title

DECIMAL MULTIPLICATION FOR SIGNED INTEGERS (PACKED BCD)

FUNCTION

Multiplication of 2 decimal integers in sign-magnitude notation.

Multiplicand, multiplier, and product are of equal length as defined by LENG.

MULTIPLICAND X MULTIPLIER → MULTIPLIER

Parameters

Input:

Length of numbers (in bytes) is defined by LENG.

MPLC, MPLC+1, MPLC+2, etc., contain multiplicand.

MPLR, MPLR+1, MPLR+2, etc., contain multiplier.

Output:

MPLR, MPLR+1, MPLR+2, etc., contain product.

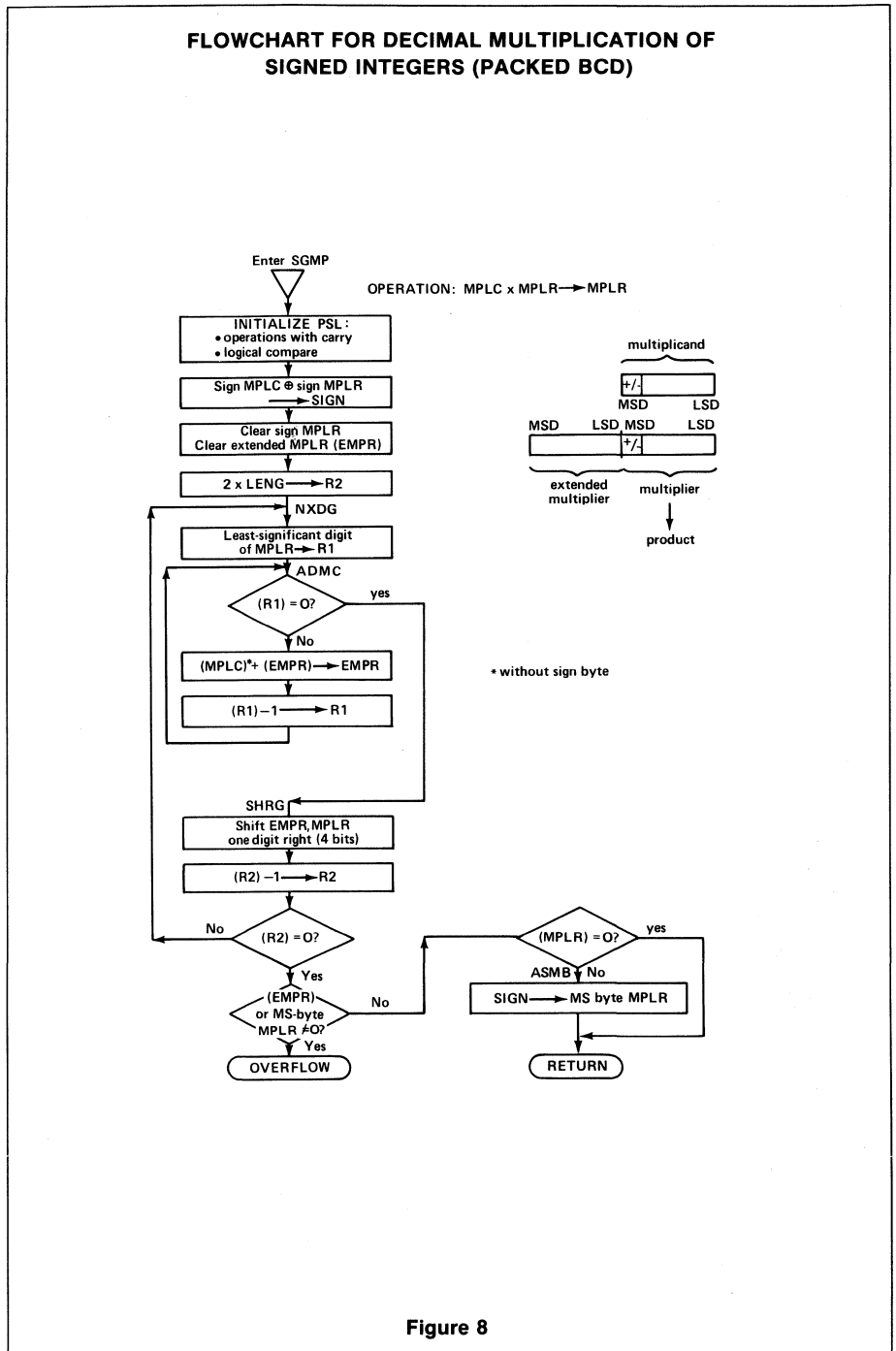
Multiplier is destroyed after multiplication.

Overflow is detected.

OPERATION

Prior to the multiplication algorithm (which is an unsigned operation), the sign of the product is determined. The multiplication gives a double-length result, of which only the least-significant half is retained as the product. If the most-significant half is unequal to zero, an overflow is detected. A "minus-zero" is excluded by means of a test for zero product.

Refer to Figures 8 and 9 for flowchart and program listing.



HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES): (3 X LENG) + 1

ROM REQUIRED (BYTES): 111

MAXIMUM SUBROUTINE NESTING LEVELS: None

ASSEMBLER/COMPILER USED: TWIN VER 1.0

DECIMAL MULTIPLICATION FOR SIGNED INTEGERS

```

TWIN ASSEMBLER VER 1.0                PAGE 0001

LINE ADDR OBJECT E SOURCE
0001      * P0760085
0002      *****
0003      * DECIMAL MULTIPLICATION FOR SIGNED-INTEGERS.
0004      * NUMBERS ARE IN PACKED BCD. SIGN-MAGNITUDE NOTATION *
0005      *****
0006      * OPERATION: MULTIPLICAND X MULTIPLIER -> MULTIPLIER
0007      * MULTIPLICAND IS IN: MPLC,MPLC+1,MPLC+2, ETC.
0008      * MULTIPLIER IS IN: MPLR,MPLR+1,MPLR+2, ETC.
0009      * PRODUCT IS IN: MPLR,MPLR+1,MPLR+2, ETC.
0010      * MULTIPLIER IS DESTROYED AFTER MULTIPLICATION.
0011      * MPLC,MPLR ARE MOST-SIGNIFICANT BYTES.
0012      * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG
0013      * ALLOWED RANGE: 1 < LENG < 65.
0014      * MS BYTE REPRESENTS SIGN: H'00' FOR +, H'F0' FOR -
0015      *
0016      * DEFINITIONS OF SYMBOLS:
0017      *
0018 0000  R0 EQU 0      PROCESSOR-REGISTERS
0019 0001  R1 EQU 1
0020 0002  R2 EQU 2
0021 0003  R3 EQU 3
0022 0008  WC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
0023 0002  COM EQU H'02' 1-LOGIC, 0-ARITH COMPARE
0024 0001  C EQU H'01' CARRY/BORROW
0025 0000  Z EQU 0      BRANCH CONDITION: ZERO
0026 0003  UN EQU 3     UNCONDITIONAL
0027      *
0028      * PARAMETERS *
0029      *
0030 0005  LENG EQU 5     LENGTH OF OPERANDS (BYTES)
0031      *
0032 0000  ORG H'700'
0033      *
0034 0700  MPLC RES LENG MULTIPLICAND
0035 0705  EMPLR RES LENG EXTENDED MULTIPLIER
0036 070A  MPLR RES LENG MULTIPLIER
0037      * NOTE: EMPLR AND MPLR MUST BE IN SUCCESSIVE
0038      * RAM LOCATIONS FOR DOUBLE-LENGTH SHIFT.
0039 070F  SIGN RES 1     TEMPORARY SIGN
0040      *
0041      * *****
0042 0710  ORG H'500'   * MULTIPLICATION PROGRAM *
0043      * *****
0044      *
0045 0500 770A  SGMP PPSL WC+COM OPERATIONS WITH CARRY, LOGICAL COMPARE
0046 0502 0C0700  LODA,R0 MPLC  FETCH SIGN MULTIPLICAND
0047 0505 2C070A  EORR,R0 MPLR  TAKE EX-OR WITH SIGN MULTIPLIER
0048 0508 0C070F  STRA,R0 SIGN  SAVE PRODUCT SIGN IN SIGN
0049 0500 20      EORZ R0      CLEAR R0
0050 0500 0706  LODI,R3 LENG+1  LOAD INDEX REGISTER
0051 0500 CF4705  CLEM STRA,R0 EMPLR,R3;- CLEAR EXTENDED MULTIPLIER AND SIGN OF MULTIPLIER
0052 0511 5B7B  BRRR,R3 CLEM  BRANCH IF NOT DONE
0053 0513 060A  LODI,R2 LENG+LENG  LOAD LOOP COUNTER WITH NUMBER OF DIGITS
0054 0515 00070E  NODG LODA,R1 MPLR+LENG-1  FETCH LS-BYTE MULTIPLIER
0055 0518 450F  ANDI,R1 H'0F'  CLEAR MS-DIGIT
0056 051A 1826  BCTR,Z SHRG  BRANCH IF LS-DIGIT IS ZERO
    
```

```

TWIN ASSEMBLER VER 1.0                PAGE 0002

LINE ADDR OBJECT E SOURCE
0058      * ADD MULTIPLICAND TO EXTENDED
0059      * MULTIPLIER WITHOUT SIGN
0060 051C 7501  ADMC CPSL C      CLEAR CARRY
0061 051E 0704  LODI,R3 LENG-1  LOAD INDEX REGISTER
0062 0520 0F6705  ADMM LODA,R0 EMPLR,R3  FETCH BYTE OF EXTENDED MULTIPLIER
0063 0523 0466  ADDI,R0 H'66'  ADD OFFSET FOR DECIMAL ADJUST
0064 0525 CF6705  STRA,R0 EMPLR,R3  RESTORE INTERMEDIATE SUM
0065 0528 FB76  BRRR,R3 ADMM  BRANCH IF ALL BYTES NOT READY
0066 052A 0704  LODI,R3 LENG-1  LOAD INDEX REGISTER
0067 052C 0F6705  ADMM LODA,R0 EMPLR,R3  FETCH BYTE OF INTERMEDIATE SUM
0068 052F 0F6700  ADMA,R0 MPLC,R3  ADD BYTE OF MULTIPLICAND
0069 0532 94      DRR,R0      DECIMAL ADJUST RESULT
0070 0533 CF6705  STRA,R0 EMPLR,R3  STORE RESULTING BYTE
0071 0536 FB74  BRRR,R3 ADMA  BRANCH IF NOT READY
0072 0538 0C0705  LODA,R0 EMPLR  FETCH MS-BYTE EXTENDED MULTIPLIER
0073 053B 0400  ADDI,R0 0      ADD CARRY
0074 053D 0C0705  STRA,R0 EMPLR  RESTORE MS-BYTE EXTENDED MULTIPLIER
0075 0540 F95A  BRRR,R1 ADMC  BRANCH IF NOT READY WITH DIGIT
0076      *
0077      * SHIFT EMPLR AND MPLR ONE DIGIT
0078      * POSITION RIGHT (4 BITS)
0079 0542 0504  SHRG LODI,R1 4  LOAD LOOP COUNTER
0080 0544 7501  SHRR CPSL C      CLEAR CARRY
0081 0546 07F6  LODI,R3 -LENG-LENG  LOAD INDEX REGISTER
0082 0548 0F660F  SHRI LODA,R0 EMPLR-256+LENG+LENG,R3  FETCH BYTE OF EXTENDED MULTIPLIER
0083 054B 50      RRR,R0      ROTATE RIGHT WITH CARRY
0084 054C CF660F  STRA,R0 EMPLR-256+LENG+LENG,R3  RESTORE BYTE
0085 054F D877  BIRR,R3 SHRI  BRANCH IF ALL NOT SHIFTED
0086 0551 F971  BRRR,R1 SHRR  BRANCH IF 4 BITS NOT SHIFTED
0087      *
0088 0553 FA40  BRRR,R2 NODG  BRANCH IF ALL DIGITS NOT READY
0089      *
0090      * TEST FOR OVERFLOW: OVERFLOW IF
0091      * (EMPLR) OR MS-BYTE MPLR ARE UNEQUAL TO ZERO
0092 0555 0706  LODI,R3 LENG+1  LOAD INDEX REGISTER
0093 0557 0F4705  TOVF LODA,R0 EMPLR,R3;-  FETCH BYTE OF EXTENDED MPLR
0094 055A 9813  BCFR,Z OVFL  BRANCH IF NOT ZERO
0095 055C 5B79  BRRR,R3 TOVF  BRANCH IF ALL BYTES NOT TESTED
0096 055E 0704  LODI,R3 LENG-1  TEST IF PRODUCT=0; LOAD INDEX
0097 0560 0F670A  TZER LODA,R0 MPLR,R3  FETCH BYTE OF PRODUCT
0098 0563 9803  BCFR,Z ASMB  BRANCH IF NOT ZERO
0099 0565 FB79  BRRR,R3 TZER  BRANCH IF ALL BYTES NOT TESTED
0100 0567 17      RETC,UN     PRODUCT=0; SIGN REMAINS ZERO.
0101      *
0102 0568 0C070F  ASMB LODA,R0 SIGN  FETCH PRODUCT SIGN
0103 056B 0C070A  STRA,R0 MPLR  STORE IN MS-BYTE MPLR
0104 056E 17      RETC,UN     RETURN
0105      *
0106 056F 40      OVFL HALT  ARITHMETIC OVERFLOW
0107      *
0108 0000      END 0

TOTAL ASSEMBLY ERRORS = 0000
    
```

Figure 9

Program Title

DECIMAL DIVISION FOR SIGNED INTEGERS (PACKED BCD)

Function

Division of 2 decimal integers in sign-magnitude notation.

Dividend, divisor, quotient, and remainder are of equal length as defined by LENG.

DIVIDEND: DIVISOR → DIVIDEND, REMAINDER

Parameters

Input:

Length of numbers (in bytes) is defined by LENG.

DVDN, DVDN+1, DVDN+2, etc., contain dividend.

DVSR, DVSR+1, DVSR+2, etc., contain divisor.

Output:

DVDN, DVDN+1, DVDN+2, etc., contain quotient.

RMDR, RMDR+1, RMDR+2, etc., contain remainder.

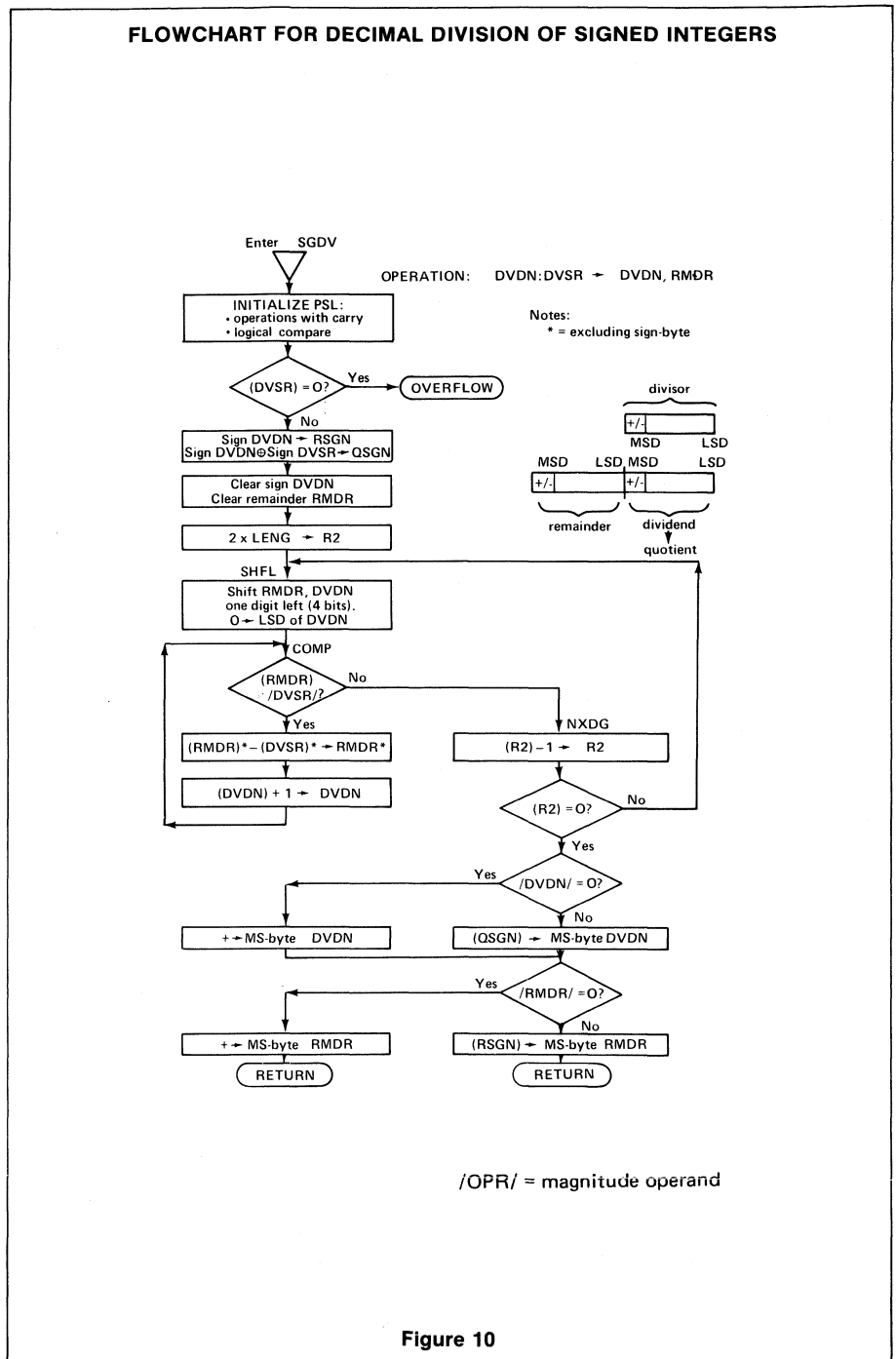
Dividend is destroyed after division.

Overflow is detected.

OPERATION:

Prior to the division, which in itself is an unsigned operation, the signs of the remainder and quotient are determined. Because the division can result in a zero quotient and/or remainder, the possibility of a "minus zero" is excluded by tests. If the divisor is zero, overflow is detected.

Refer to Figures 10 and 11 for flowchart and program listing.



HARDWARE AFFECTED									
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'		
PSU	F	II	SP					RAM REQUIRED (BYTES): <u> (3 x LENG) + 4 </u> ROM REQUIRED (BYTES): <u> 144 </u>	
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X		
								ASSEMBLER/COMPILER USED: <u> TWIN VER 1.0 </u>	

DECIMAL DIVISION FOR SIGNED INTEGERS

```
TWIN ASSEMBLER VER 1.0 PAGE 0001
LINE ADDR OBJECT E SOURCE
0001 * P0760004
0002 *****
0003 * DECIMAL DIVISION FOR SIGNED INTEGERS *
0004 * NUMBERS ARE IN PACKED BCD, SIGN-MAGNITUDE NOTATION *
0005 *****
0006 * OPERATION:
0007 * DIVIDEND : DIVISOR --> DIVIDEND, REMAINDER
0008 * DIVIDEND 15 IN: DVDN,DVDN+1,DVDN+2, ETC.
0009 * DIVISOR 15 IN: DVSR,DVSR+1,DVSR+2, ETC.
0010 * QUOTIENT 15 IN: DVQN,DVQN+1,DVQN+2, ETC.
0011 * REMAINDER 15 IN: RMDR,RMDR+1,RMDR+2, ETC.
0012 * DIVIDEND 15 DESTROYED AFTER DIVISION.
0013 * DVDN,DVSR AND RMDR ARE MOST-SIGNIFICANT BYTES.
0014 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG.
0015 * ALLOWED RANGE: 1 < LENG < 65.
0016 * MS-BYTE HOLDS SIGN INFORMATION: H'00' FOR +, H'F0' FOR -
0017 *
0018 * DEFINITIONS OF SYMBOLS:
0019 *
0020 0000 R0 EQU 0 PROCESSOR REGISTERS
0021 0001 R1 EQU 1
0022 0002 R2 EQU 2
0023 0003 R3 EQU 3
0024 0000 MC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
0025 0002 COM EQU H'02' 1=LOGIC, 0=ARITH COMPARE
0026 0001 C EQU H'01' CARRY/BORROW
0027 0000 Z EQU 0 BRANCH COND.: ZERO
0028 0001 P EQU 1 POSITIVE
0029 0002 N EQU 2 NEGATIVE
0030 0000 EQ EQU 0 EQUAL
0031 0002 LT EQU 2 LESS THAN
0032 0003 UN EQU 3 UNCONDITIONAL
0033 *
0034 * PARAMETERS *
0035 *
0036 0005 LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0037 *
0038 0000 * ORG H'700'
0039 *
0040 0700 RMDR RES LENG REMAINDER
0041 0705 DVDN RES LENG DIVIDEND
0042 * NOTE: RMDR AND DVDN MUST BE IN SUCCESSIVE
0043 * RAM LOCATIONS, BECAUSE OF DOUBLE-LENGTH SHIFT.
0044 070A DVSR RES LENG DIVISOR
0045 070F TEMP RES 2 TEMPORARY STORAGE FOR ADDRESS
0046 0711 QSGN RES 1 QUOTIENT SIGN
0047 0712 RSGN RES 1 REMAINDER SIGN
0048 *
0049 *
0050 0713 * ORG H'500'
0051 *
0052 *****
0053 * SUBROUTINE TO TEST OPERAND FOR ZERO *
0054 *****
0055 * OPERAND ADDRESS MUST BE IN R0,R1 (HIGH,LOW ADDR.)
0056 * ALL BYTES, EXCEPT MS-BYTE (=SIGN) ARE TESTED.
0057 * CONDITION CODE BECOMES 00 IF OPERAND WAS ZERO.
0058 *
0059 0500 C0870F TZER STRA,R0 TEMP SAVE OPERAND ADDRESS
0060 0503 C08710 STRA,R1 TEMP+1
0061 0506 0704 LODI,R3 LENG-1 LOAD INDEX REGISTER
0062 0508 0FE70F TZE0 LODA,R0 *TEMP,R3 FETCH BYTE OF OPERAND
0063 050B 15 RETC,P RETURN IF POSITIVE (CC=01)
0064 050C 16 RETC,N RETURN IF NEGATIVE (CC=10)
0065 050D FB79 BDRR,R3 TZE0 BRANCH IF ALL NOT TESTED
0066 050F 17 \ RETC,UN RETURN WITH CC=00
0067 *
0068 *
0069 * * DIVISION PROGRAM *
0070 * *****
0071 *
0072 0510 770A SGDV PPSL MC+COM OPERATIONS WITH CARRY,
0073 * LOGICAL COMPARISON
0074 0512 0407 LODI,R0 >DVSR HIGH-ADDRESS DIVISOR TO R0
0075 0514 050A LODI,R1 >DVSR LOW-ADDRESS DIVISOR TO R1
```

```
0076 0516 3F0500 BSTR,UN TZER TEST DIVISOR FOR ZERO
0077 0519 1C0595 BCTR,Z OVFL BRANCH IF ZERO
0078 051C 0C0705 LODA,R0 DVDN FETCH SIGN DIVIDEND
0079 051F CC0712 STRA,R0 RSGN SAVE IN REMAINDER SIGN
0080 0522 2C070A EORA,R0 DVSR TAKE EX-OR WITH DVSR SIGN
0081 0525 CC0711 STRA,R0 QSGN SAVE IN QUOTIENT SIGN
0082 0528 20 EORZ R0 CLEAR R0
0083 0529 0706 LODI,R3 LENG-1 LOAD INDEX REGISTER
0084 052B CF4700 CLRM STRA,R0 RMDR,R3- CLEAR REMAINDER AND SIGN DVDN
0085 052E 5B7B BRRR,R3 CLRM BRANCH IF NOT DONE
0086 *
0087 0530 060A * LODI,R2 LENG+LENG NUMBER OF DIGITS TO LOOP COUNTER
0088 *
0089 *
0090 * SHIFT RMDR/DVN 4 BITS LEFT
0091 0532 0504 SHFL LODI,R1 4 LOAD BIT COUNTER
0092 0534 7501 SHF0 CPSEL C CLEAR CARRY
0093 0536 070A LODI,R3 LENG+LENG LOAD INDEX REGISTER
0094 0538 0F4700 SHF1 LODA,R0 RMDR,R3- FETCH BYTE OF RMDR/DVN
0095 053B 00 RRL,R0 ROTATE LEFT WITH CARRY
0096 053C CF6700 STRA,R0 RMDR,R3 RESTORE SHIFTED BYTE
0097 053F 5B77 BRRR,R3 SHF1 BRANCH IF ALL NOT SHIFTED
0098 0541 F971 BRRR,R1 SHF0 BRANCH IF 4 BITS NOT SHIFTED
```

```
TWIN ASSEMBLER VER 1.0 PAGE 0003
LINE ADDR OBJECT E SOURCE
0100 *
0101 * COMPARE RMDR AND DVSR TO TEST
IF SUBTRACTION IS POSSIBLE.
0102 0543 0500 COMP LODI,R1 0 CLEAR R1; MS-BIT OF R1 BECOMES
1 FOR RMDR < DVSR.
0103 *
0104 0545 0704 LODI,R3 LENG-1 LOAD INDEX REGISTER
0105 0547 0F6700 COM0 LODA,R0 RMDR,R3 FETCH BYTE OF REMAINDER
0106 054A 0F670A COMA,R0 DVSR,R3 COMPARE WITH BYTE OF DIVISOR
0107 054D 1802 BCTR,E0 COM1 BRANCH IF EQUAL
0108 054F 13 SPSEL PSFL PSL TO R0
0109 0550 C1 STR2 R1 SAVE PSL IN R1
0110 0551 FB74 COM1 BDRR,R3 COM0 BRANCH IF ALL BYTES NOT TESTED
0111 0553 01 LOOZ R1 FETCH STATUS OF COMPARISON
0112 0554 1A1A BCTR,LT NXDG BRANCH IF RMDR < DVSR
0113 *
0114 *
0115 * SUBTRACT DIVISOR FROM
REMAINDER WITHOUT MS-BYTES;
0116 0556 7701 PPSL C CLEAR BORROW
0117 0558 0704 LODI,R3 LENG-1 LOAD INDEX REGISTER
0118 055A 0F6700 SURD LODA,R0 RMDR,R3 FETCH BYTE OF REMAINDER
0119 055D 0F670A SUBA,R0 DVSR,R3 SUBTRACT BYTE OF DIVISOR
0120 0560 34 DRR,R0 DECIMAL ADJUST RESULT
0121 0561 CF6700 STRA,R0 RMDR,R3 RESTORE IN REMAINDER
0122 0564 FB74 BDRR,R3 SURD BRANCH IF NOT READY
0123 *
0124 0566 0C0709 LODA,R0 DVDN+LENG-1 FETCH LS-BYTE QUOTIENT
0125 0569 0800 BIRR,R0 #+2 INCREASE R0
0126 056B CC0709 STRA,R0 DVDN+LENG-1 RESTORE INCREMENTED QUOTIENT
0127 056E 1B53 BCTR,UN COMP BRANCH FOR NEXT COMPARISON
0128 *
0129 0570 FA40 NXDG BDRR,R2 SHFL BRANCH IF DIVISION NOT READY
0130 0572 0E0711 LODA,R2 QSGN FETCH SIGN QUOTIENT
0131 0575 0407 LODI,R0 >DVQN HIGH-ADDRESS QUOTIENT TO R0
0132 0577 0505 LODI,R1 >DVQN LOW- ADDRESS QUOTIENT TO R1
0133 0579 3F0500 BSTR,UN TZER TEST IF QUOTIENT IS ZERO
0134 057C 9002 BCTR,Z STOS BRANCH IF NOT ZERO
0135 057E 0600 CLEAR R0
0136 0580 0E0705 STOS STRA,R2 DVDN STORE SIGN IN MS-BYTE DVDN
0137 0583 0E0712 LODA,R2 QSGN FETCH REMAINDER SIGN
0138 0586 0407 LODI,R0 RMDR HIGH-ADDRESS REMAINDER TO R0
0139 0588 0500 LODI,R1 >RMDR LOW- ADDRESS REMAINDER TO R1
0140 058A 3F0500 BSTR,UN TZER TEST IF REMAINDER IS ZERO
0141 058D 9002 BCTR,Z STRS BRANCH IF NOT ZERO
0142 058F 0600 LODI,R2 0 CLEAR R2
0143 0591 CE0700 STRS STRA,R2 RMDR STORE SIGN IN MS-BYTE RMDR
0144 0594 17 RETC,UN RETURN
0145 *
0146 0595 40 OVFL HALT OVERFLOW LOCATION
0147 *
0148 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000
```

Figure 11

ROUTINES FOR SIGNED FIXED-POINT ARITHMETIC

As illustrated in Figure 12, the numbers used in these arithmetic routines are in sign-magnitude notation with decimal point indication. The latter gives the number of decimals, and has a minimum of zero and a maximum limit of 15 or the number of digits, whichever is smaller.

The length of the numbers is defined by the number of bytes (including the sign byte) they require. This parameter can be modified by changing the definition of LENG in the source program. Note that for clarity, each routine is written in a "stand-alone" form. If more than one routine is required in a program, considerable savings in the program space required can be realized by breaking out common operations as sub-routines.

Program Title

ALIGNMENT SUBROUTINE FOR FIXED-POINT NUMBERS

Function

Aligns a fixed-point number to the decimal point indicated by the contents of register DPNT. Performs rounding as specified.

Parameters

Input:
RO contains the high address of the operand.

R1 contains the low address of the operand.

DPNT contains the required decimal point.

ROUN contains the rounding constant: (ROUN) = H'00' for no rounding; (ROUN) = H'05' for 5/4 rounding; and (ROUN) = H'09' for round-up.

Prior to entry, WC in PSL must be 1.

Length of operand (in bytes) is defined by LENG.

Output:

Aligned operand; rounded if specified.

Alignment overflow is detected.

Operation:

The results of a fixed-point operation must be aligned to the required number of decimals. By means of this aligning routine, the numbers are shifted left or right, if necessary, until the appropriate decimal point position is obtained. This position must have previously been stored in a register designated DPNT. During left alignment, overflow can occur if a non-zero digit drops out of the most-significant digit position.

During aligning it is also possible to perform rounding of the operand. This is done by adding a rounding digit to the most-significant digit of the decimals which are truncated by the right alignment. This rounding digit must have previously been stored in register ROUN and gives the possibilities listed above. Since rounding

can only be performed during right alignment, the required decimal point position must be less than 15 if rounding is desired. If the aligned result is minus zero, the sign is changed.

Refer to Figures 13 and 14 for flowchart and program listing.

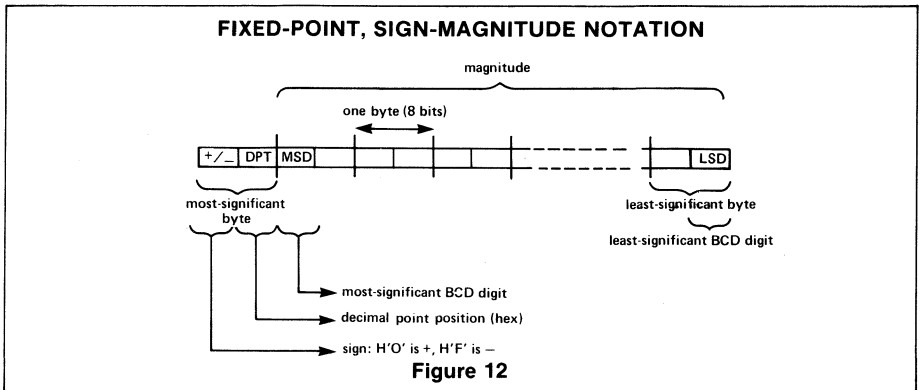


Figure 12

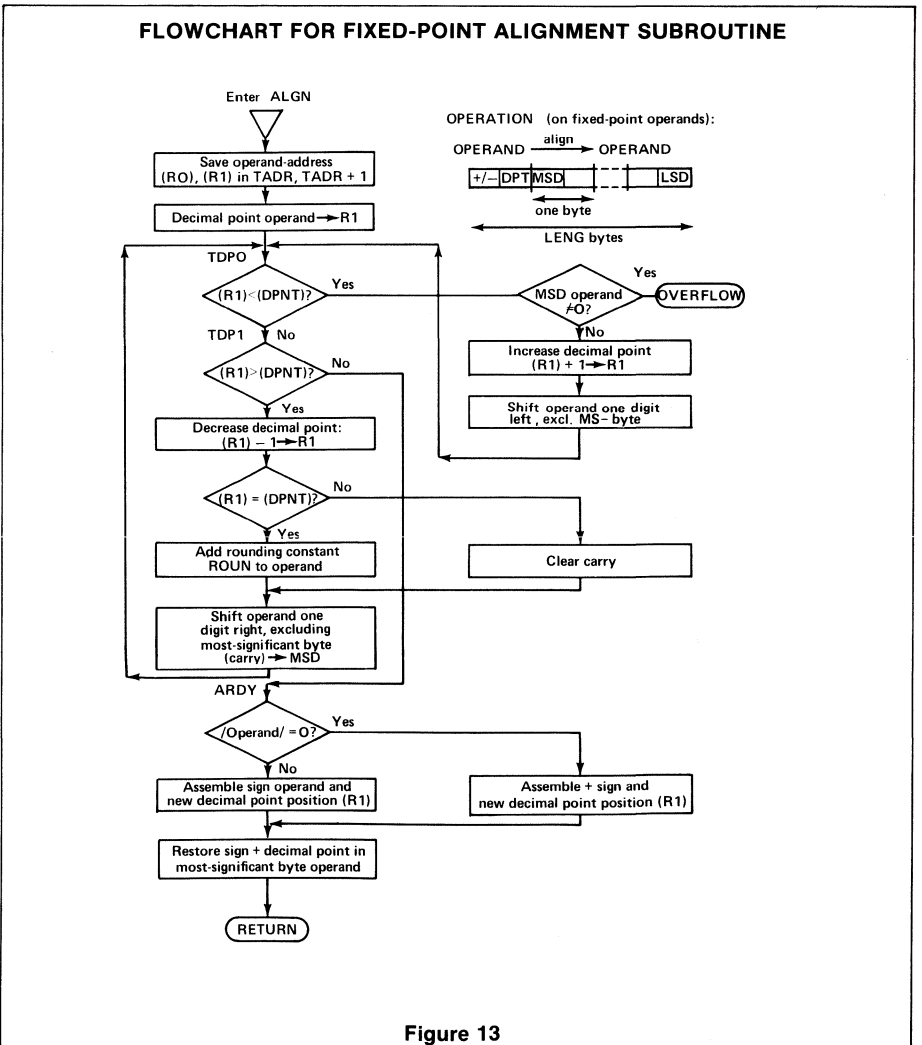


Figure 13

HARDWARE AFFECTED							
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X			X		X

RAM REQUIRED (BYTES):	4
ROM REQUIRED (BYTES):	120
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	TWIN VER 1.0

FIXED-POINT ALIGNMENT SUBROUTINE

```

TWIN ASSEMBLER VER 1.0                PAGE 0001
LINE ADDR OBJECT E SOURCE
0001          * PD70000
0002          *****
0003          * FIXED POINT ALIGNMENT SUBROUTINE *
0004          *****
0005          *
0006          * DEFINITIONS OF SYMBOLS:
0007          *
0008 0000      R0 EQU 0      PROCESSOR REGISTERS
0009 0001      R1 EQU 1
0010 0002      R2 EQU 2
0011 0003      R3 EQU 3
0012 0008      MC EQU H'00'  PSL: 1=WITH, 0=WITHOUT CARRY
0013 0001      C EQU H'01'  CARRY/BORROW
0014 0000      Z EQU 0      BRANCH COND.: ZERO
0015 0000      EQ EQU 0      EQUAL
0016 0001      GT EQU 1     GREATER THAN
0017 0002      LT EQU 2     LESS THAN
0018 0003      UN EQU 3     UNCONDITIONAL
0019          *
0020          * PARAMETERS *
0021          *
0022 0005      LENG EQU 5     LENGTH OF OPERAND (BYTES)
0023          *
0024 0000      ORG H'400'
0025          *
0026 0440      DPNT RES 1     REQUIRED DECIMAL POINT (0 THROUGH 15)
0027 0441      ROUN RES 1    ROUNING CONSTANT (0,5 OR 9)
0028 0442      TADR RES 2    TEMPORARY STORAGE FOR ADDRESS
0029          *
0030 0444      ORG H'450'    START OF SUBROUTINE
0031          *
0032          * OPERAND IS ALIGNED TO DECIMAL POINT POSITION AS
0033          * INDICATED BY REGISTER DPNT.
0034          * ROUNDING IS PERFORMED UNDER FOLLOWING CONDITIONS:
0035          * (ROUN) CONTAINS H'00' FOR NO ROUNDING
0036          * (ROUN) CONTAINS H'05' FOR 5/4 ROUNDING
0037          * (ROUN) CONTAINS H'09' FOR ROUND-UP
0038          * (DPNT) MUST BE < 15 IF ROUNDING IS REQUIRED.
0039          * ALIGNMENT-OVERFLOW IS DETECTED.
0040          * PRIOR TO ENTRY: MC IN PSL MUST BE 1.
0041          *      R0 CONTAINS HIGH-ADDR OF OPERAND
0042          *      R1 CONTAINS LOW- ADDR OF OPERAND
0043          *      DPNT CONTAINS DECIMAL POINT
0044          *      ROUN CONTAINS ROUNDING CONSTANT
0045          *
0046 0450 0C0442  ALGN STRA,R0 TADR  SAVE HI-ADDRESS OF OPERAND
0047 0453 0D0443  STRA,R1 TADR,H1  SAVE LO-ADDRESS OF OPERAND
0048 0456 0D0442  LODA,R1 *TADR  FETCH MS-BYTE OF OPERAND
0049 0459 450F  ANDI,R1 H'0F'  REMOVE SIGN, KEEP DECIMAL POINT
0050 045B 0604  TDP0 LODI,R2 4     LOAD LOOP COUNTER
0051 045D ED0440  COMP,R1 DPNT  COMPARE ACTUAL DECIMAL POINT WITH
0052          *      REQUIRED DECIMAL POINT.
0053 0460 91C    BCFR,LT TDP1  BRANCH IF EQUAL OR TOO BIG
0054 0462 20     EOR2 R0      CLEAR R0
0055 0463 0C0442  LODA,R0 *TADR,R0,+  FETCH MS-DIGITS OF OPERAND
0056 0466 44F0  ANDI,R0 H'F0'  CLEAR LS-DIGIT (TEST MSD = 0)
    
```

```

TWIN ASSEMBLER VER 1.0                PAGE 0002
LINE ADDR OBJECT E SOURCE
0057 0468 9C04C8  BCFR,Z OVF0  BRANCH IF ALIGNMENT OVERFLOW
0058 046B D900    BIRR,R1 #+2  INCREASE DECIMAL POINT
0059          *      SHIFT OPERAND ONE DIGIT LEFT,
0060          *      EXCEPT MS-BYTE (SIGN+DPNT)
0061 046D 7501    SHL0 CPSL C    CLEAR CARRY
0062 046F 0704    LODI,R3 LENG-1  LOAD INDEX REG
0063 0471 0FE442  SHL1 LODA,R0 *TADR,R3  FETCH BYTE OF OPERAND
0064 0474 D0     RRL,R0      ROTATE LEFT WITH CARRY
0065 0475 CFE442  STRA,R0 *TADR,R3  RESTORE
0066 0478 FB77    BORR,R3 SHL1  BRANCH IF ALL NOT SHIFTED
0067 047A FA71    BORR,R2 SHL0  BRANCH IF 4 BITS NOT SHIFTED
0068 047C 1B5D    BCTR,UN TDP0  BRANCH FOR NEXT TEST
0069          *
0070 047E 9933    TDF1 BCFR,GT ARDY  BRANCH IF DECIMAL POINT IS CORRECT
0071 0480 F900    BORR,R1 #+2  DECREASE DECIMAL POINT
0072 0482 ED0440  COMP,R1 DPNT  TEST IF LS-DIGIT IS ROUNDING DIGIT
0073 0485 9018    BCFR,EQ SHR0  BRANCH IF NOT
0074          *      ADD (ROUN) TO ROUNDING-DIGIT:
0075 0487 7501    CPSL C      CLEAR CARRY
0076 0489 0704    LODI,R3 LENG-1  LOAD INDEX REGISTER
0077 048B 0FE442  RND0 LODA,R0 *TADR,R3  FETCH BYTE OF OPERAND
0078 048E 9466    ADDI,R0 H'66'  ADD OFFSET FOR BCD ADD
0079 0490 E704    COMI,R3 LENG-1  COMPLEMENT
0080 0492 9803    BCFR,EQ RND1  BRANCH IF NOT LS-BYTE
0081 0494 0C0441  ADDA,R0 ROUN  ADD ROUNDING CONSTANT
0082 0497 94     RND1 DAR,R0  DECIMAL ADJUST RESULT
0083 0498 CFE442  STRA,R0 *TADR,R3  RESTORE RESULT
0084 049B FB6E    BORR,R3 RND0  BRANCH IF ALL BYTES NOT READY
0085 049D 1B02    BCTR,UN SHR1  BRANCH TO RIGHT-SHIFT OPERAND
0086          *      SHIFT OPERAND ONE DIGIT RIGHT,
0087          *      EXCEPT MS-BYTE (SIGN+DPNT)
0088 049F 7501    SHR0 CPSL C    CLEAR CARRY
0089 04A1 0700    SHR1 LODI,R3 0    CLEAR INDEX
0090 04A3 0FA442  SHR2 LODA,R0 *TADR,R3,+  FETCH BYTE OF OPERAND
0091 04A6 50     RRR,R0      ROTATE RIGHT WITH CARRY
0092 04A7 CFE442  STRA,R0 *TADR,R3  RESTORE BYTE
0093 04A9 E704    COMI,R3 LENG-1  COMPLEMENT
0094 04AC 9075    BCFR,EQ SHR2  BRANCH IF ALL NOT SHIFTED
0095 04AE FA6F    BORR,R2 SHR0  BRANCH IF 4 BITS NOT SHIFTED
0096 04B0 1F045B  BCTR,UN TDP0  BRANCH FOR NEXT TEST
0097          *
0098 04B3 0E0442  ARDY LODA,R2 *TADR  FETCH MS-BYTE OF OPERAND
0099 04B6 46F0    ANDI,R2 H'F0'  REMOVE DECIMAL POINT, KEEP SIGN
0100 04B8 0704    LODI,R3 LENG-1  LOAD INDEX REGISTER FOR ZERO TEST
0101 04BA 0FE442  TZER LODA,R0 *TADR,R3  FETCH BYTE OF ALIGNED OPERAND
0102 04BD 9083    BCFR,Z NZER  BRANCH IF NON-ZERO
0103 04BF FB79    BORR,R3 TZER  BRANCH IF ALL BYTES NOT READY
0104 04C1 C2     STR2 R2     ZERO RESULT, CLEAR SIGN
0105 04C2 02     NZER LOD2 R2  FETCH SIGN
0106 04C3 61     IOR2 R1     ASSEMBLE SIGN AND DECIMAL POINT
0107 04C4 0C0442  STRA,R0 *TADR  STORE IN MS-BYTE OF OPERAND
0108 04C7 17     RETC,UN     RETURN
0109          *
0110 04C8 40     OVF0 HALT   ALIGNMENT OVERFLOW
0111          *
0112 0000          END 0
    
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 14

Program Title

FIXED-POINT ADDITION/SUBTRACTION OF SIGNED, PACKED BCD NUMBERS

Function

Addition/subtraction of 2 decimal fixed-point numbers.

Operands and result are of equal length as defined by LENG.

OPERAND1 +/- OPERAND2 → OPERAND2

Parameters

Input:

Length of numbers (in bytes) defined by LENG.

OPR1, OPR1+1, OPR1+2, etc. contain augend or subtrahend.

OPR2, OPR2+1, OPR2+2, etc., contain addend or minuend.

In the alignment subroutine, the decimal-point position is in DPNT and the rounding constant is in ROUN.

Output:

OPR2, OPR2+1, OPR2+2, etc., contain sum or difference.

Result and operand1 are aligned (and rounded if specified).

Overflow is detected.

Special Requirements

Software: Fixed-point alignment subroutine ALGN.

Operation

Subtraction is performed by changing the sign of the second operand before entering the (signed) addition routine. Prior to the addition/subtraction of the magnitudes of the operands, both operands are aligned (and rounded if programmed), the sign of the result is determined and, in the event the operands have opposite signs, the subtrahend and minuend are designated.

Refer to Figures 15 and 16 for flowchart and program listing.

FLOWCHART FOR ADDITION/SUBTRACTION OF FIXED-POINT NUMBERS

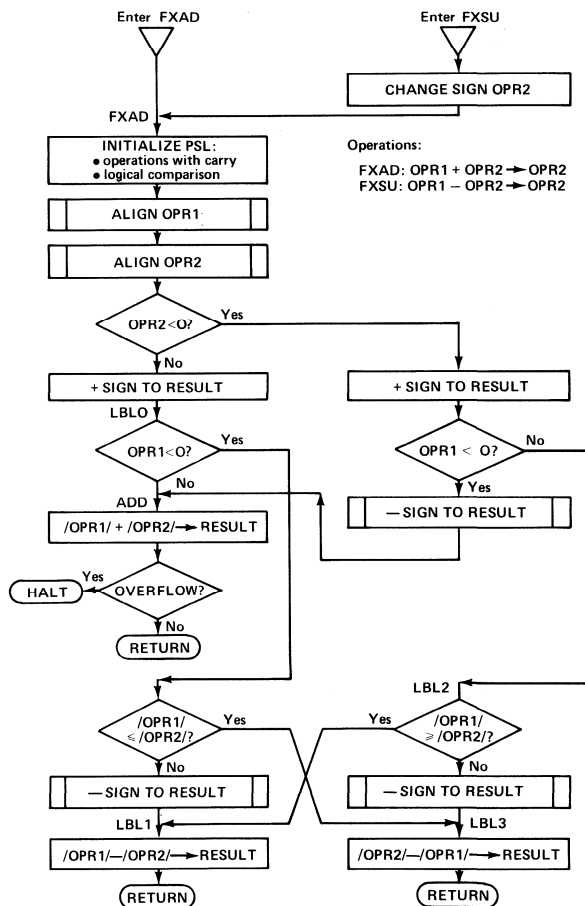


Figure 15

HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X
RAM REQUIRED (BYTES): 2 x LENG							
ROM REQUIRED (BYTES): 151							
MAXIMUM SUBROUTINE NESTING LEVELS: 1							
ASSEMBLER/COMPILER USED: TWIN VER 1.0							

FIXED-POINT DECIMAL ADDITION/SUBTRACTION
FOR SIGNED, PACKED BCD NUMBERS

TWIN ASSEMBLER VER 1.0 PAGE 0001

```

LINE ADDR OBJECT E SOURCE
0001 * PD760002
0002 *****
0003 * FIXED-POINT DECIMAL ADDITION/SUBTRACTION *
0004 * FOR SIGNED, PACKED BCD NUMBERS. *
0005 *****
0006 * OPERATION: OPERAND1 +/- OPERAND2 --> OPERAND2
0007 * OPERAND1 IS IN: OPR1, OPR1+1, OPR1+2, ETC.
0008 * OPERAND2 IS IN: OPR2, OPR2+1, OPR2+2, ETC.
0009 * SUM/DIFFERENCE IS IN: OPR2, OPR2+1, OPR2+2, ETC.
0010 * OPERAND2 IS DESTROYED AFTER ADD/SUBTRACT.
0011 * OPR1, OPR2 ARE MOST-SIGNIFICANT BYTES.
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG.
0013 * ALLOWED RANGE: 1 < LENG < 255.
0014 * NUMBERS ARE IN SIGN-MAGNITUDE NOTATION.
0015 * MS-BYTE HOLDS SIGN AND DECIMAL POINT INFORMATION:
0016 * SIGN IS IN MS 4 BITS: H'0' IS +, H'F' IS -
0017 * DECIMAL POINT IS IN LS 4 BITS: BINARY CODED.
0018 * RANGE (0 THRU 15) EQUALS NUMBER OF DECIMALS.
0019 *
0020 * DEFINITIONS OF SYMBOLS:
0021 *
0022 0000 R0 EQU 0 PROCESSOR REGISTERS
0023 0001 R1 EQU 1
0024 0002 R2 EQU 2
0025 0003 R3 EQU 3
0026 0008 MC EQU H'08' PSL: 1=WITH, 0=WITHOUT CARRY
0027 0002 COM EQU H'02' 1=LOGIC, 0=ARITH COMPARE
0028 0001 C EQU H'01' CARRY/BORROW
0029 0000 Z EQU 0 BRANCH COND.: ZERO
0030 0002 N EQU 2 NEGATIVE
0031 0000 EQ EQU 0 EQUAL
0032 0001 GT EQU 1 GREATER THAN
0033 0002 LT EQU 2 LESS THAN
0034 0003 UN EQU 3 UNCONDITIONAL
0035 *
0036 * PARAMETERS *
0037 *
0038 0450 ALGN EQU H'450' ADDRESS OF ALIGNMENT SUBROUTINE
0039 0005 LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0040 *
0041 0000 ORG H'700'
0042 *
0043 0700 OPR1 RES LENG OPERAND1
0044 0705 OPR2 RES LENG OPERAND2/RESULT
0045 *
    
```

TWIN ASSEMBLER VER 1.0 PAGE 0002

```

LINE ADDR OBJECT E SOURCE
0047 0700 ORG H'500'
0048 *
0049 *****
0050 * SUBROUTINE TO COMPARE OPERAND1 WITH OPERAND2 (UPDATE CC) *
0051 *****
0052 0500 0500 C012 L001, R1 0 CLEAR R1; MS-BITS ARE USED
0053 * TO SAVE CC INFORMATION
0054 0502 0704 L001, R3 LENG-1 LOAD INDEX REGISTER
0055 0504 0F6700 COM0 L00A, R0 OPR1, R3 FETCH BYTE OF OPERAND1
0056 0507 0F6705 COMA, R0 OPR2, R3 COMPARE WITH BYTE OF OPERAND2
0057 050A 1802 BCTR, EQ COM1, BRANCH IF EQUAL
0058 050C 13 SP5L PSL TO R0
0059 0500 C1 STRZ R1 SAVE PSL IN R1
0060 050E FB74 COM1 B0RR, R3 COM0 BRANCH IF ALL BYTES NOT TESTED
0061 0510 01 L002 R1 UPDATE CC WITH STATUS COMPARE
0062 0511 17 RETC, UN RETURN
0063 *
0064 *****
0065 * SUBROUTINE TO SET SIGN OF RESULT TO NEGATIVE *
0066 *****
0067 0512 0C0705 SSGN L00A, R0 OPR2 FETCH SIGN OF RESULT
0068 0515 64F8 IOR1, R0 H'F0' SET NEGATIVE SIGN
0069 0517 0C0705 STRA, R0 OPR2 RESTORE
0070 051A 17 RETC, UN RETURN
0071 *
0072 *
0073 * * FIXED-POINT SUBTRACTION *
0074 *
0075 *
    
```

```

0076 051B 0C0705 FXSU L00A, R0 OPR2 FETCH SIGN OF OPERAND2
0077 051E 24F8 EOR1, R0 H'F0' CHANGE SIGN
0078 0520 0C0705 STRA, R0 OPR2 RESTORE SIGN OF OPERAND2
0079 *
0080 *
0081 * *****
0082 * * FIXED-POINT ADDITION *
0083 * *****
0084 0523 770A FXAD PPSL MC+COM OPERATIONS WITH CARRY, LOGICAL COMPARE
0085 0525 0407 L001, R0 <OPR1 HIGH-ADDRESS OPR1 TO R0
0086 0527 0500 L001, R1 >OPR1 LOW- ADDRESS OPR1 TO R1
0087 0529 3F0450 BSTA, UN ALGN ALIGN OPERAND1
0088 052C 0407 L001, R0 <OPR2 HIGH-ADDRESS OPR2 TO R0
0089 052E 0505 L001, R1 >OPR2 LOW- ADDRESS OPR2 TO R1
0090 0530 3F0450 BSTA, UN ALGN ALIGN OPERAND2
0091 0533 0C0705 L00A, R0 OPR2 FETCH SIGN OPERAND2
0092 0536 C1 STRZ R1 SAVE IN R1
0093 0537 440F AND1, R0 H'0F' REMOVE SIGN
0094 0539 0C0705 STRA, R0 OPR2 SET SIGN OF RESULT TO +
0095 053C 01 L002 R1 FETCH SIGN OPERAND2
0096 053D 9A21 BCFR, N LBL0 BRANCH IF OPR2 NOT NEGATIVE
0097 053F 0C0700 L00A, R0 OPR1 FETCH SIGN OPERAND1
0098 0542 9A1A BCFR, N LBL2 BRANCH IF OPR1 NOT NEGATIVE
0099 0544 3F0512 BSTA, UN SSGN SET NEGATIVE SIGN RESULT
    
```

TWIN ASSEMBLER VER 1.0 PAGE 0003

```

LINE ADDR OBJECT E SOURCE
0101 *
0102 0547 7501 ADD OPSL C (OPR1) + (OPR2) --> OPR2
0103 0549 0704 L001, R3 LENG-1 CLEAR CARRY
0104 054B 0500 L001, R1 0 CLEAR INTERBYTE CARRY
0105 054D 0F6700 R000 L00A, R0 OPR1, R3 FETCH BYTE OF OPERAND1
0106 0550 0466 ADD1, R0 H'66' ADD OFFSET FOR BCD ADD
0107 0552 51 RRR, R1 INTERBYTE CARRY TO CARRY
0108 0553 0F6705 R00A, R0 OPR2, R3 ADD BYTE OF OPERAND2
0109 0556 94 DRR, R0 DECIMAL ADJUST RESULT
0110 0557 0F6705 STRA, R0 OPR2, R3 STORE RESULTING BYTE
0111 055A 01 RRL, R1 CARRY (=INTERBYTE CARRY) TO R3, CLEAR CARRY
0112 055B FB70 B0RR, R3 R000 BRANCH IF NOT READY
0113 055D 5038 BCFR, Z OVFL BRANCH IF INTERBYTE CARRY = 1
0114 055F 17 RETC, UN RETURN
0115 *
0116 0560 0C0700 LBL0 L00A, R0 OPR1 FETCH SIGN OF OPERAND1
0117 0563 9A62 BCFR, N ADD BRANCH IF OPR1 NOT NEGATIVE
0118 0565 3F0500 BSTA, UN C012 COMPARE OPR1 WITH OPR2
0119 * (MAGNITUDES ONLY).
0120 0568 991C BCFR, GT LBL3 BRANCH IF OPR1 < OPR2
0121 056A 3F0512 BSTA, UN SSGN SET NEGATIVE SIGN OF RESULT
0122 *
0123 * (OPR1) - (OPR2) --> OPR2:
0124 056D 0704 LBL1 L001, R3 LENG-1 LOAD INDEX REGISTER
0125 056F 7701 PPSL C CLEAR BORROW
0126 0571 0F6700 SUI2 L00A, R0 OPR1, R3 FETCH BYTE OF OPERAND1
0127 0574 0F6705 SUBA, R0 OPR2, R3 SUBTRACT BYTE OF OPERAND2
0128 0577 94 DRR, R0 DECIMAL ADJUST RESULT
0129 0578 0F6705 STRA, R0 OPR2, R3 STORE RESULTING BYTE IN OPR2
0130 057B FB74 B0RR, R3 SUI2 BRANCH IF NOT READY
0131 057D 17 RETC, UN RETURN
0132 *
0133 057E 3F0500 LBL2 BSTA, UN C012 COMPARE OPR1 WITH OPR2
0134 * (MAGNITUDES ONLY)
0135 0581 9A6A BCFR, LT LBL1 BRANCH IF OPR1 > OPR2
0136 0583 3F0512 BSTA, UN SSGN SET NEGATIVE SIGN OF RESULT
0137 *
0138 * (OPR2) - (OPR1) --> OPR2:
0139 0586 0704 LBL3 L001, R3 LENG-1 LOAD INDEX REGISTER
0140 0588 7701 PPSL C CLEAR BORROW
0141 058A 0F6705 SUI2 L00A, R0 OPR2, R3 FETCH BYTE OF OPERAND2
0142 058D 0F6700 SUBA, R0 OPR1, R3 SUBTRACT BYTE OF OPERAND1
0143 0590 94 DRR, R0 DECIMAL ADJUST RESULT
0144 0591 0F6705 STRA, R0 OPR2, R3 STORE RESULTING BYTE
0145 0594 FB74 B0RR, R3 SUI2 BRANCH IF NOT READY
0146 0596 17 RETC, UN RETURN
0147 *
0148 0597 40 OVFL HALT ARITHMETIC OVERFLOW
0149 *
0150 0000 END 0
    
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 16

Program Title

FIXED-POINT DECIMAL MULTIPLICATION FOR SIGNED, PACKED BCD NUMBERS

Function

Multiplication of 2 decimal fixed-point numbers.

Multiplicand, multiplier, and product are of equal length as defined by LENG.

MULTIPLICAND x MULTIPLIER → MULTIPLIER

Parameters

Input:

Length of numbers (in bytes) is defined by LENG.

MPLC, MPLC+1, MPLC+2, etc., contain multiplicand.

MPLR, MPLR+1, MPLR+2, etc., contain multiplier.

Output:

MPLR, MPLR+1, MPLR+2, etc., contain product.

Multiplier is destroyed after multiplication.

Overflow is detected.

Special Requirements

Software: Fixed-point alignment subroutine ALGN

Operation

Prior to the multiplication algorithm (which is an unsigned operation), the product sign is determined. The product is formed in a double-length register and is right aligned until the decimal point is 15 or less; this is required due to the fixed-point format. Then the product length is reduced to the single-length, fixed-point format; if this is not possible, overflow is detected. A "minus zero" product result is excluded by means of a test during aligning.

Refer to Figures 17 and 18 for flowchart and program listing.

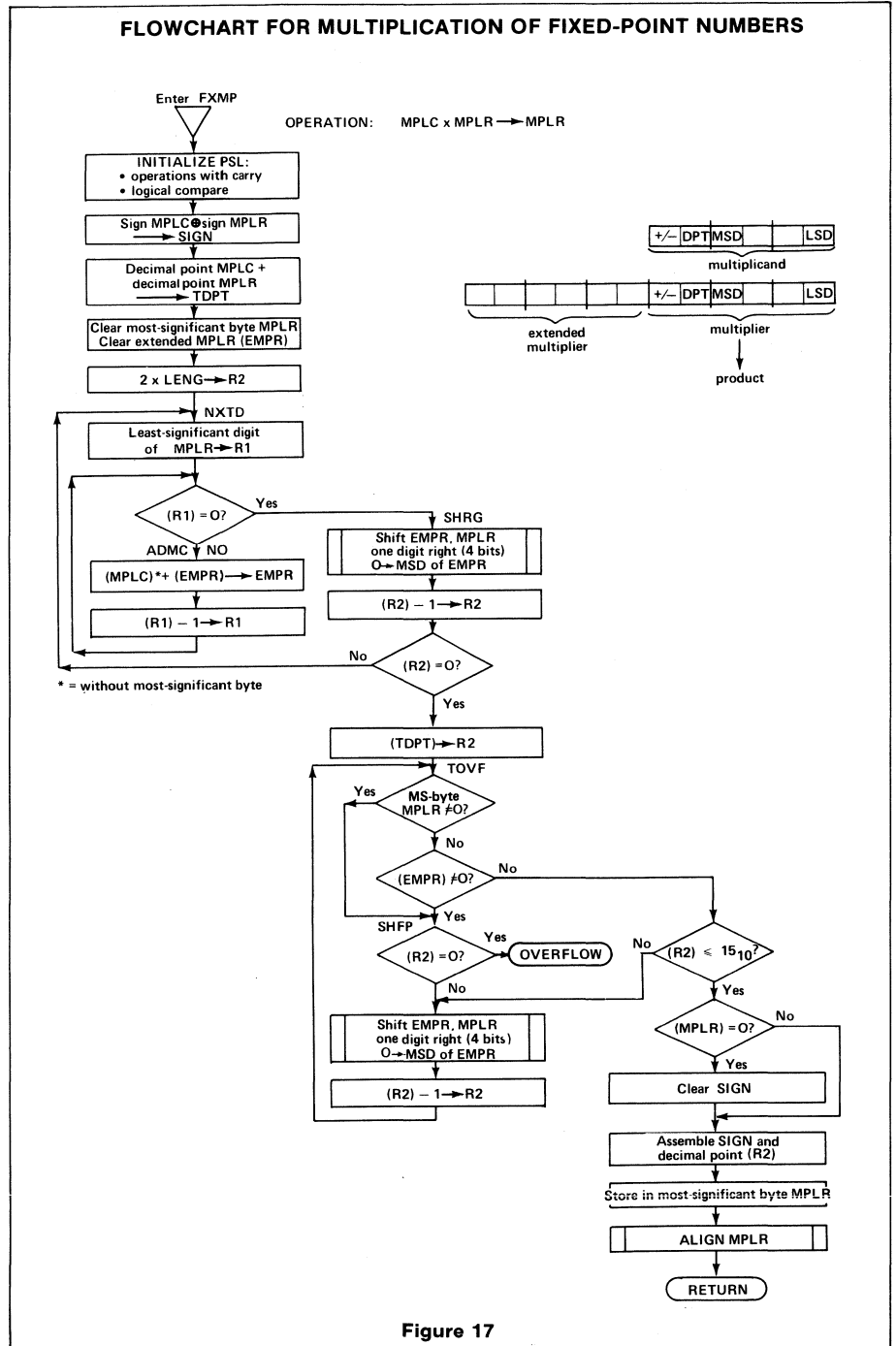


Figure 17

HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES):	(3 X LENG) + 4
ROM REQUIRED (BYTES):	144
MAXIMUM SUBROUTINE NESTING LEVELS:	1
ASSEMBLER/COMPILER USED:	TWIN VER 1.0

FIXED-POINT DECIMAL MULTIPLICATION FOR SIGNED, PACKED BCD NUMBERS

TWIN ASSEMBLER VER 1.0 PAGE 0001

```

LINE ADDR OBJECT E SOURCE
0001 * PD7600B3
0002 *****
0003 * FIXED POINT DECIMAL MULTIPLICATION FOR *
0004 * SIGNED, PACKED-BCD NUMBERS *
0005 *****
0006 * OPERATION: MULTIPLICAND X MULTIPLIER -> MULTIPLIER
0007 * MULTIPLICAND IS IN: MPLC,MPLC+1,MPLC+2, ETC
0008 * MULTIPLIER IS IN: MPLR,MPLR+1,MPLR+2, ETC
0009 * PRODUCT IS IN: MPLR,MPLR+1,MPLR+2, ETC
0010 * MULTIPLIER IS DESTROYED AFTER MULTIPLICATION
0011 * MPLC,MPLR ARE MOST-SIGNIFICANT BYTES
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG
0013 * ALLOWED RANGE: 1 < LENG < 65
0014 * REQUIRED NUMBER OF DECIMALS IN PRODUCT MUST BE
0015 * STORED IN LOCATION: DPNT (MPLR = 15).
0016 * NUMBERS ARE IN SIGN-MAGNITUDE NOTATION
0017 * MS-BYTE HOLDS SIGN AND DECIMAL POINT INFORMATION:
0018 * SIGN IS IN MS 4 BITS: 'H'0' IS +, 'H'F' IS -
0019 * DECIMAL POINT IS IN LS 4 BITS: BINARY CODED.
0020 * RANGE (0 THRU 15) EQUALS NUMBER OF DECIMALS.
0021 *
0022 * DEFINITIONS OF SYMBOLS:
0023 *
0024 0000 R0 EQU 0 PROCESSOR REGISTERS
0025 0001 R1 EQU 1
0026 0002 R2 EQU 2
0027 0003 R3 EQU 3
0028 0008 MC EQU 'H'00' PSL: 1=WITH, 0=WITHOUT CARRY
0029 0002 COM EQU 'H'02' 1=LOGIC, 0=ARITH COMPARE
0030 0001 C EQU 'H'01' CARRY/BORROW
0031 0000 Z EQU 0 BRANCH COND.: ZERO
0032 0002 LT EQU 2 LESS THAN
0033 0003 UN EQU 3 UNCONDITIONAL
0034 *
0035 * PARAMETERS *
0036 *
0037 0450 ALGN EQU 'H'450' ADDRESS OF ALIGNMENT SUBROUTINE
0038 0005 LENG EQU 5 LENGTH OF PARAMETERS (BYTES)
0039 *
0040 0000 ORG 'H'700'
0041 *
0042 0700 MPLC RES LENG MULTIPLICAND
0043 0705 EMPR RES LENG EXTENDED MULTIPLIER
0044 070A MPLR RES LENG MULTIPLIER
0045 * NOTE: EMPR AND MPLR MUST BE IN SUCCESSIVE
0046 * RAM LOCATIONS FOR DOUBLE-LENGTH SHIFT.
0047 070F SIGN RES 1 TEMPORARY SIGN
0048 0710 TEMP RES 2 TEMPORARY STORAGE FOR ADDRESS
0049 0712 TDPT RES 1 TEMPORARY STORAGE FOR DECIMAL POINT
0050 *
0051 0713 ORG 'H'500'
    
```

TWIN ASSEMBLER VER 1.0 PAGE 0002

```

LINE ADDR OBJECT E SOURCE
0053 *****
0054 * SUBROUTINE TO SHIFT EMPR AND MPLR ONE DIGIT RIGHT *
0055 *****
0056 * PRIOR TO ENTRY: MC IN PSL MUST BE 1.
0057 *
0058 0500 0504 SHEN LODI,R1,4 LOAD LOOP COUNTER
0059 0502 07F6 SHE0 LODI,R3,-LENG-LENG LOAD INDEX REGISTER
0060 0504 7501 CPSL C CLEAR CARRY
0061 0506 0F660F SHE1 LODA,R0 EMPR-256+LENG+LENG,R3 FETCH BYTE
0062 0509 50 RRR,R0 ROTATE RIGHT
0063 050A CF660F STRA,R0 EMPR-256+LENG+LENG,R3 RESTORE BYTE
0064 0500 D677 BIRR,R3 SHE1 BRANCH IF ALL NOT SHIFTED
0065 050F F971 BARR,R1 SHE0 BRANCH IF 4 BITS NOT SHIFTED
0066 0511 17 RETC,UN RETURN
0067 *
0068 *****
0069 * FIXED POINT MULTIPLICATION *
0070 *****
0071 *
0072 0512 770A FZMP PPSL MC,COM OPERATIONS WITH CARRY, LOGICAL COMPARE
0073 0514 006700 LODA,R1 MPLC FETCH MS-BYTE MULTIPLICAND
0074 0517 01 LOOZ R1 SAVE IN R0
    
```

```

0075 0510 0E070A LODA,R2 MPLR FETCH MS-BYTE MULTIPLIER
0076 0510 22 EORZ R2 TAKE EX-OR OF SIGNS
0077 051C 44F0 ANDI,R0 'H'F0' REMOVE NON-SIGN DIGIT
0078 051E C0070F STRA,R0 SIGN SAVE SIGN
0079 0521 01 LOOZ R1 MS-BYTE OF MPLC TO R0
0080 0522 440F ANDI,R0 'H'0F' REMOVE SIGN MPLC, KEEP DECIMAL POINT
0081 0524 460F ANDI,R2 'H'0F' REMOVE SIGN MPLR, KEEP DECIMAL POINT
0082 0526 7501 CPSL C CLEAR CARRY
0083 0528 82 ADDZ R2 ADD DECIMAL POINT POSITIONS
0084 0529 C00712 STRA,R0 TDPT SAVE NEW DECIMAL POINT POSITION
0085 *
0086 052C 20 EORZ R0 CLEAR R0
0087 052D 0706 LODI,R3 LENG+1 LOAD INDEX REGISTER
0088 052F CF4705 CLEM STRA,R0 EMPR,R3,- CLEAR MS-BYTE MPLR, ALL EMPR
0089 0532 5B7B BRNR,R3 CLEM BRANCH IF NOT DONE
0090 *
0091 0534 060A LODI,R2 LENG+LENG NUMBER OF DIGITS TO LOOP COUNTER
0092 0536 00070E NXTD LODA,R1 MPLR+LENG-1 FETCH LS-BYTE MULTIPLIER
0093 0539 450F ANDI,R1 'H'0F' TAKE ONLY LS-DIGIT
0094 053B 1826 BCTR,Z SHRG BRANCH IF ZERO
0095 *
0096 * ADD MPLC (WITHOUT MS-BYTE) TO EMPR
0097 053D 7501 ADMC CPSL C CLEAR CARRY
0098 053F 0704 LODI,R3 LENG-1 LOAD INDEX REGISTER
0099 0541 0F6705 ADMA LODA,R0 EMPR,R3 FETCH BYTE OF EXTENDED MULTIPLIER
0100 0544 8466 ADDI,R0 'H'66' ADD OFFSET
0101 0546 CF6705 STRA,R0 EMPR,R3 RESTORE INTERMEDIATE SUM
0102 0549 FB76 BARR,R3 ADMA BRANCH IF ALL BYTES NOT ADDED
0103 054B 0704 LODI,R3 LENG-1 LOAD INDEX REGISTER
0104 054D 0F6705 ADMA LODA,R0 EMPR,R3 FETCH BYTE OF INTERMEDIATE SUM
0105 0550 0F6700 ADMA LODA,R0 MPLC,R3 ADD BYTE OF MULTIPLICAND
0106 0552 94 DAR,R0 DECIMAL ADJUST RESULT
0107 0554 CF6705 STRA,R0 EMPR,R3 RESTORE RESULTING BYTE
    
```

TWIN ASSEMBLER VER 1.0 PAGE 0003

```

LINE ADDR OBJECT E SOURCE
0108 0557 FB74 BARR,R3 ADM1 BRANCH IF NOT READY
0109 0559 0C0705 LODA,R0 EMPR FETCH MS-BYTE EXTENDED MULTIPLIER
0110 055C 8400 ADDI,R0 0 ADD CARRY
0111 055E C00705 STRA,R0 EMPR RESTORE
0112 0561 F95A BARR,R1 ADMC DECREMENT DIGIT, BRANCH IF NOT 0
0113 0563 3F0500 SHRG BSTR,UN SHEM SHIFT EMPR AND MPLR RIGHT ONE DIGIT POSITION
0114 0566 FA4E BARR,R2 NXTD BRANCH IF MULTIPLICATION NOT READY
0115 *
0116 0568 0E0712 LODA,R2 TDPT DECIMAL POINT TO R2
0117 056B 0706 TOWF LODI,R3 LENG+1 TEST OVERFLOW; LOAD INDEX REGISTER
0118 056D 0F4705 TOWB LODA,R0 EMPR,R3,- FETCH BYTE OF EMPR OR MS-BYTE
0119 * OF MPLR TO TEST FOR ZERO.
0120 0570 9814 BCFR,Z SHFP BRANCH IF NOT ZERO
0121 0572 5B79 BRNR,R3 TOWB BRANCH IF ALL NOT TESTED
0122 *
0123 0574 E610 COMI,R2 16 TEST IF DECIMAL POINT IS < 16.
0124 0576 9411 BCFR,LT SHFB BRANCH IF TOO BIG
0125 0578 6E070F IORR,R2 SIGN ASSEMBLE SIGN AND DECIMAL POINT
0126 057B CE070A RSMB STRA,R2 MPLR STORE IN MS-BYTE MPLR.
0127 057E 0407 LODI,R0 CMPLR HIGH-ORDER ADDRESS MPLR TO R0
0128 0580 050A LODI,R1 CMPLR LOW-ORDER ADDRESS MPLR TO R1
0129 0582 3F0450 BSTR,UN ALGN ALIGN PRODUCT; SET + SIGN IF
0130 * PRODUCT IS ZERO.
0131 0585 17 RETC,UN
0132 *
0133 0586 02 SHFP LOOZ R2 UPDATE CC FOR NUMBER OF DECIMALS
0134 0587 1807 BCTR,Z OVFL BCTR IF ZERO; OVERFLOW
0135 0589 FA00 SHFB BARR,R2 #+2 DECREASE DECIMAL POINT
0136 058B 3F0500 BSTR,UN SHEM SHIFT EMPR + MPLR RIGHT
0137 058E 185B BCTR,UN TOWF BRANCH FOR OVERFLOW TEST
0138 *
0139 0590 40 OVFL HALT ARITHMETIC OVERFLOW
0140 *
0141 0000 END 0
    
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 18

Program Title

FIXED-POINT DECIMAL DIVISION FOR SIGNED, PACKED BCD NUMBERS

Function

Division of 2 decimal numbers (fixed point). Dividend, divisor, and quotient are of equal length as defined by LENG.

DIVIDEND :DIVISOR → DIVIDEND.

Parameters

Input:

Length of numbers (in bytes) is defined by LENG.

DVDN, DVDN+1, DVDN+2, etc., contain dividend.

DVSR, DVSR+1, DVSR+2, etc., contain divisor.

Output:

DVDN, DVDN+1, DVDN+2, etc., contain quotient.

Dividend is destroyed after division.

Overflow is detected.

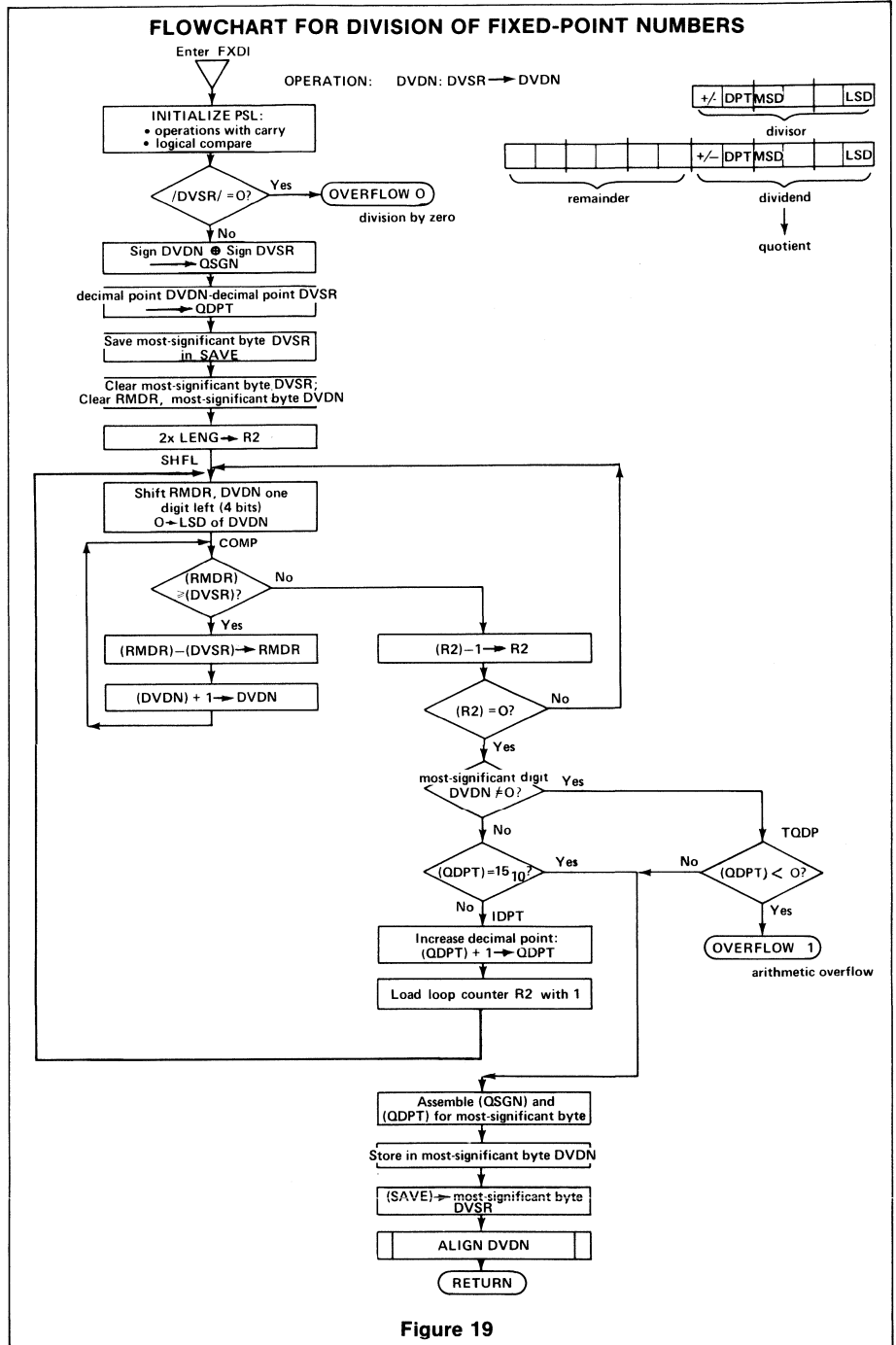
Special Requirements

Software: Fixed-point alignment subroutine ALGN.

Operation

Prior to the division algorithm (which is an unsigned operation), the sign of the quotient is determined. To obtain maximum precision, the division procedure is continued until either a non-zero most-significant digit is detected or the maximum allowed decimal point position is reached. Then the resulting quotient is aligned with a minus zero result suppressed. Overflow is detected if the divisor is zero.

Refer to Figures 19 and 20 for flowchart and program listing.



HARDWARE AFFECTED							
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PGL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES):	(3 x LENG) +5
ROM REQUIRED (BYTES):	166
MAXIMUM SUBROUTINE NESTING LEVELS:	1
ASSEMBLER/COMPILER USED:	TWIN VER 1.0

FIXED-POINT DECIMAL DIVISION FOR SIGNED, PACKED BCD NUMBERS

TWIN ASSEMBLER VER 1.0 PAGE 0001

```

LINE ADDR OBJECT E SOURCE
0001 * PD70001
0002 *****
0003 * FIXED-POINT DECIMAL DIVISION *
0004 * FOR SIGNED, PACKED-BCD NUMBERS *
0005 *****
0006 * OPERATION DIVIDEND DIVISOR -> DIVIDEND:
0007 * DIVIDEND IS IN DYNL,DYDH+1,DYDH+2, ETC.
0008 * DIVISOR IS IN DVSR,DVSR+1,DVSR+2, ETC.
0009 * QUOTIENT IS IN DVQN,DVQH+1,DVQH+2, ETC.
0010 * DIVIDEND IS DESTROYED AFTER DIVISION.
0011 * DYNL AND DVSR ARE MOST-SIGNIFICANT BYTES.
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY LENG.
0013 * ALLOWED RANGE: 1 < LENG < 65.
0014 * NUMBERS ARE IN SIGN-MAGNITUDE NOTATION.
0015 * MS-BYTE HOLDS SIGN AND DECIMAL POINT INFORMATION.
0016 * SIGN IS IN MS 4 BITS: H'0' IS +, H'F' IS -
0017 * DECIMAL POINT IS IN LS 4 BITS: BINARY CODED.
0018 * RANGE (0 THRU 15) EQUALS NUMBER OF DECIMALS.
0019 *
0020 * DEFINITIONS OF SYMBOLS:
0021 *
0022 0000 R0 EQU 0 PROCESSOR REGISTERS
0023 0001 R1 EQU 1
0024 0002 R2 EQU 2
0025 0003 R3 EQU 3
0026 0000 MC EQU H'08' PSL: 1=WITH, 0=WITHOUT CARRY
0027 0002 COM EQU H'02' 1=LOGIC, 0=ARITH COMPARE
0028 0001 C EQU H'01' CARRY/BORROW
0029 0000 Z EQU 0 BRANCH COND.: ZERO
0030 0001 P EQU 1 POSITIVE
0031 0002 N EQU 2 NEGATIVE
0032 0000 EQ EQU 0 EQUAL
0033 0002 LT EQU 2 LESS THAN
0034 0003 UN EQU 3 UNCONDITIONAL
0035 *
0036 * PARAMETERS *
0037 *
0038 0450 ALGN EQU H'450' ADDRESS OF ALIGNMENT SUBROUTINE
0039 0005 LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0040 *
0041 0000 * ORG H'700'
0042 *
0043 0700 RMDR RES LENG REMAINDER
0044 0705 DVND RES LENG DIVIDEND
0045 * NOTE: RMDR AND DVND MUST BE IN SUCCESSIVE
0046 * RRM LOCATIONS, BECAUSE OF DOUBLE-LENGTH SHIFT.
0047 070A DVSR RES LENG DIVISOR
0048 070F TEMP RES 2 TEMPORARY STORAGE FOR ADDRESS
0049 0711 QSGN RES 1 QUOTIENT SIGN
0050 0712 QDPT RES 1 QUOTIENT DECIMAL POINT
0051 0713 SAVE RES 1 TEMPORARY STORAGE
0052 *

TWIN ASSEMBLER VER 1.0 PAGE 0002
LINE ADDR OBJECT E SOURCE
0054 0714 * ORG H'500'
0055 *
0056 0500 770B FXDI PPSL MC+COM+C OPERATIONS WITH CARRY,
0057 * LOGICAL COMPARISON, CLEAR BORROW
0058 0502 0704 L001,R3 LENG-1 LOAD INDEX REGISTER FOR ZERO TEST
0059 0504 0F670A TZER L00A,R0 DVSR,R3 FETCH BYTE OF DIVISOR
0060 0507 5005 BCFR,Z NZER BRANCH IF NON-ZERO
0061 0509 FB79 B0RR,R3 TZER BRANCH IF ALL BYTES NOT R0V
0062 050B 1005A6 BCTA,Z 0VFB BRANCH IF ZERO
0063 050E 0C0705 NZER L00A,R0 DVND FETCH MS-BYTE DIVIDEND
0064 0511 C1 STRZ R1 SAVE IN R1
0065 0512 0E070A L00A,R2 DVSR FETCH MS-BYTE DIVISOR
0066 0515 CE0713 STRA,R2 SAVE SAVE MS-BYTE DIVISOR
0067 0518 22 EORZ R2 EX-OR SIGN DVND AND DVSR
0068 0519 44F0 ANDI,R0 H'F0' REMOVE DECIMAL POINT DIGIT
0069 051B CC0711 STRA,R0 QSGN SAVE QUOTIENT SIGN
0070 051E 01 L0DZ R1 FETCH MS-BYTE DIVIDEND
0071 051F 440F ANDI,R0 H'0F' REMOVE SIGN
0072 0521 460F ANDI,R2 H'0F' REMOVE SIGN MS-BYTE DIVISOR
0073 0523 R2 SUBZ R2 SUBTRACT DECIMAL POINTS: DVND - DVSR
0074 0524 CC0712 STRA,R0 QDPT SAVE DECIMAL POINT QUOTIENT
0075 *
0076 0527 20 EORZ R0 CLEAR R0
    
```

```

0077 0528 CC070A STRA,R0 DVSR CLEAR MS-BYTE DIVISOR
0078 052B 0706 L0D1,R3 LENG+1 LOAD INDEX REGISTER
0079 052D CF4700 CLRM STRA,R0 RMDR,R3- CLEAR REMAINDER AND SIGN DVND
0080 0530 507B B0RR,R3 CLRM BRANCH IF NOT DONE
0081 *
0082 0532 060A L0D1,R2 LENG+LENG NUMBER OF DIGITS TO LOOP COUNTER
0083 *
0084 *
0085 * SHIFT RMDR/DVND 4 BITS LEFT
0086 0534 0504 SHFL L001,R1 4 LOAD BIT COUNTER
0087 0536 7501 SHF0 CPSL C CLEAR CARRY
0088 0538 070A L0D1,R3 LENG+LENG LOAD INDEX REGISTER
0089 053A 0F4700 SHF1 L00A,R0 RMDR,R3- FETCH BYTE OF RMDR/DVND
0090 053D 08 RRL,R0 ROTATE LEFT WITH CARRY
0091 053E CF6700 STRA,R0 RMDR,R3 RESTORE SHIFTED BYTE
0092 0541 5077 B0RR,R3 SHF1 BRANCH IF ALL NOT SHIFTED
0093 0543 F971 B0RR,R1 SHF0 BRANCH IF 4 BITS NOT SHIFTED
0094 *
0095 * COMPARE RMDR AND DVSR TO TEST
0096 * IF SUBTRACTION IS POSSIBLE
0097 0545 0500 COMP L001,R1 0 CLEAR R1: MS-BIT OF R1 BECOMES
0098 * 1 FOR RMDR < DVSR.
    
```

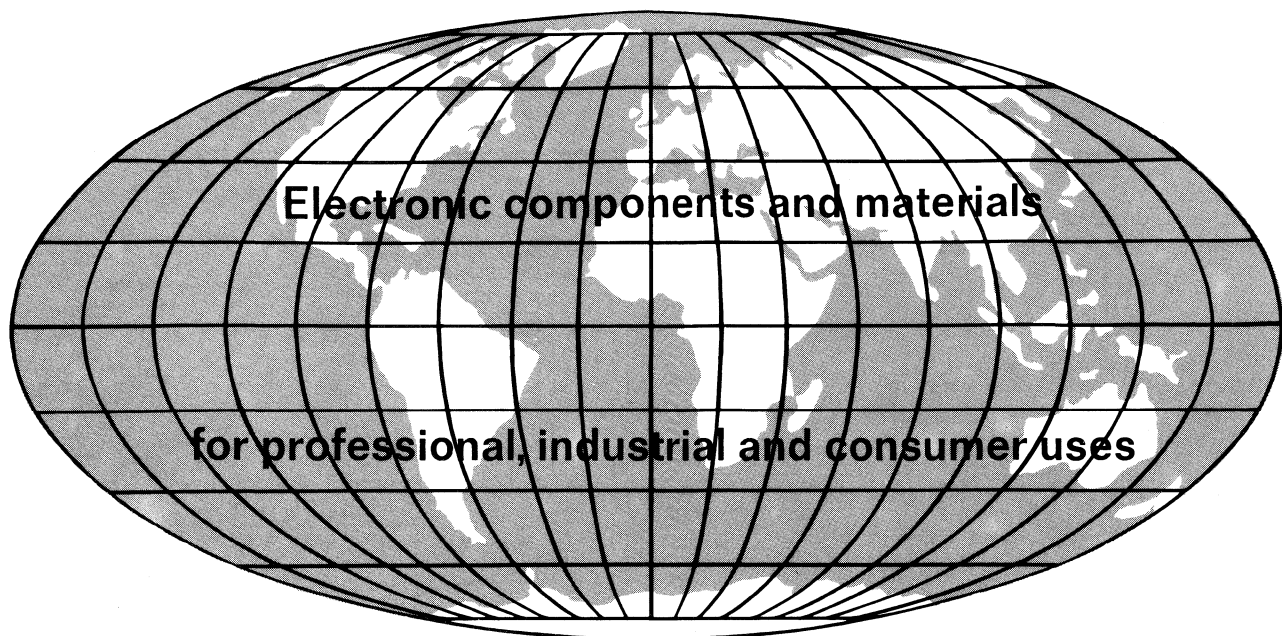
TWIN ASSEMBLER VER 1.0 PAGE 0003

```

LINE ADDR OBJECT E SOURCE
0100 0547 0705 L0D1,R3 LENG LOAD INDEX REGISTER
0101 0549 0F4700 COM0 L00A,R0 RMDR,R3- FETCH BYTE OF REMAINDER
0102 054C EF670A COMA,R0 DVSR,R3 COMPARE WITH BYTE OF DIVISOR
0103 054F 1802 BCTR,EQ COM1 BRANCH IF EQUAL
0104 0551 13 SP5L PSL TO R0
0105 0552 C1 STRZ R1 SAVE PSL IN R1
0106 0553 5074 COM1 B0RR,R3 COM0 BRANCH IF ALL BYTES NOT TESTED
0107 0555 04 L0DZ R1 FETCH STATUS OF COMPARISON
0108 0556 1A1A BCTR,LT NXDG BRANCH IF RMDR < DVSR
0109 *
0110 * SUBTRACT DIVISOR FROM REMAINDER
0111 0558 7701 PPSL C CLEAR BORROW
0112 055A 0705 L0D1,R3 LENG LOAD INDEX REGISTER
0113 055C 0F4700 SURD L00A,R0 RMDR,R3- FETCH BYTE OF REMAINDER
0114 055F AF670A SUBA,R0 DVSR,R3 SUBTRACT BYTE OF DIVISOR
0115 0562 94 DAR,R0 DECIMAL ADJUST RESULT
0116 0563 CF6700 STRA,R0 RMDR,R3 RESTORE IN REMAINDER
0117 0566 5074 B0RR,R3 SURD BRANCH IF NOT READY
0118 *
0119 0568 0C0709 L0D0,R0 DVND+LENG-1 FETCH LS-BYTE QUOTIENT
0120 056B D000 BIRR,R0 #+2 INCREASE R0
0121 056D CC0709 STRA,R0 DVND+LENG-1 RESTORE INCREMENTED QUOTIENT
0122 0570 1B53 BCTR,UN COMP BRANCH FOR NEXT COMPARISON
0123 *
0124 0572 FA00 NXDG B0RR,R2 SHFL BRANCH IF DIVISION NOT READY
0125 0574 20 EORZ R0 CLEAR INDEX REGISTER
0126 0575 0C2705 L00A,R0 DVND,R0,+ FETCH MS-DIGITS QUOTIENT
0127 0578 44F0 ANDI,R0 H'F0' TAKE MSD ONLY
0128 057A 9011 BCFR,Z 0D0F BRANCH IF MSD NOT ZERO
0129 057C 0E0712 L0D0,R2 QDPT FETCH DECIMAL POINT QUOTIENT
0130 057F E00F COM1,R2 15
0131 0581 900F BCFR,EQ AS0U BRANCH IF DECIMAL POINT=MAX.
0132 0583 D000 IDPT BIRR,R2 #+2 INCREASE DECIMAL POINT QUOTIENT
0133 0585 CE0712 STRA,R2 QDPT RESTORE
0134 0588 0601 L0D1,R2 1 LOAD LOOP COUNTER
0135 058A 1F0534 BCTR,UN SHFL BRANCH FOR NEXT DIVIDE LOOP
0136 *
0137 058D 0E0712 TD0F L0D0,R2 QDPT FETCH DECIMAL POINT QUOTIENT
0138 0590 1A15 DCTR,N 0VFB BRANCH IF NEGATIVE
0139 0592 CE0711 AS0U I0RR,R2 QSGN ASSEMBLE SIGN+DECIMAL POINT QUOTIENT
0140 0595 CE0705 STRA,R2 DVND STORE SIGN IN MS-BYTE DVND
0141 0598 0C0713 L00A,R0 SAVE FETCH SIGN+DECIMAL POINT DIVISOR
0142 059B CC070A STRA,R0 DVSR RESTORE MS-BYTE DIVISOR
0143 059E 0407 L0D1,R0 <DVND HIGH-ADDRESS QUOTIENT TO R0
0144 05A0 0505 L0D1,R1 >DVND LOW-ADDRESS QUOTIENT TO R1
0145 05A2 3F0450 BSTR,UN ALGN ALIGN QUOTIENT; SET + SIGN IF
0146 * QUOTIENT IS ZERO.
0147 05A5 17 RETC,UN RETURN
0148 *
0149 05A6 40 OVFB HALT OVERFLOW: DIVISION BY ZERO
0150 05A7 40 OVFB HALT ARITHMETIC OVERFLOW
0151 *
0152 0000 END 0
    
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 20

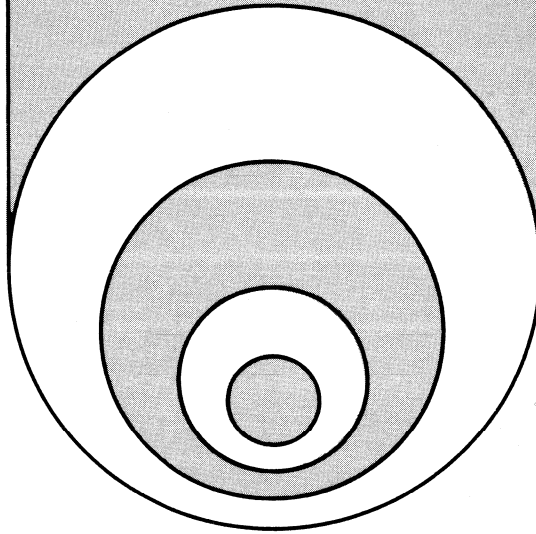


from the world-wide Philips Group of Companies

- Argentina:** FAPESA I.y.C., Av. Crovara 2550, Tablada, Prov. de BUENOS AIRES, Tel. 652-7438/7478.
- Australia:** PHILIPS INDUSTRIES HOLDINGS LTD., Elcoma Division, 67 Mars Road, LANE COVE, 2066, N.S.W., Tel. 42 1261.
- Austria:** ÖSTERREICHISCHE PHILIPS BAUELEMENTE Industrie G.m.b.H., Triester Str. 64, A-1101 WIEN, Tel. 62 91 11.
- Belgium:** M.B.L.E., 80, rue des Deux Gares, B-1070 BRUXELLES, Tel 523 00 00.
- Brazil:** IBRAPE, Caixa Postal 7383, Av. Paulista 2073-S/Loja, SAO PAULO, SP, Tel. 287-7144.
- Canada:** PHILIPS ELECTRONICS LTD., Electron Devices Div., 601 Milner Ave., SCARBOROUGH, Ontario, M1B 1M8, Tel. 292-5161.
- Chile:** PHILIPS CHILENA S.A., Av. Santa Maria 0760, SANTIAGO, Tel. 39-40 01.
- Colombia:** SADAPE S.A., P.O. Box 9805, Calle 13, No. 51 + 39, BOGOTA D.E. 1., Tel. 600 600.
- Denmark:** MINIWATT A/S, Emdrupvej 115A, DK-2400 KØBENHAVN NV., Tel. (01) 69 16 22.
- Finland:** OY PHILIPS AB, Elcoma Division, Kaivokatu 8, SF-00100 HELSINKI 10, Tel. 1 72 71.
- France:** R.T.C. LA RADIOTECHNIQUE-COMPELEC, 130 Avenue Ledru Rollin, F-75540 PARIS 11, Tel. 355-44-99.
- Germany:** VALVO, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, D-2 HAMBURG 1, Tel. (040) 3296-1.
- Greece:** PHILIPS S.A. HELLENIQUE, Elcoma Division, 52, Av. Syngrou, ATHENS, Tel. 915 311.
- Hong Kong:** PHILIPS HONG KONG LTD., Comp. Dept., Philips Ind. Bldg., Kung Yip St., K.C.T.L. 289, KWAI CHUNG, N.T. Tel. 12-24 51 21.
- India:** PHILIPS INDIA LTD., Elcoma Div., Band Box House, 254-D, Dr. Annie Besant Rd., Prabhadevi, BOMBAY-25-DD, Tel. 457 311-5.
- Indonesia:** P.T. PHILIPS-RALIN ELECTRONICS, Elcoma Division, 'Timah' Building, Jl. Jen. Gatot Subroto, JAKARTA, Tel. 44 163.
- Ireland:** PHILIPS ELECTRICAL (IRELAND) LTD., Newstead, Clonskeagh, DUBLIN 14, Tel. 69 33 55.
- Italy:** PHILIPS S.P.A., Sezione Elcoma, Piazza IV Novembre 3, I-20124 MILANO, Tel. 2-6994.
- Japan:** NIHON PHILIPS CORP., Shuwa Shinagawa Bldg., 26-33 Takanawa 3-chome, Minato-ku, TOKYO (108), Tel. 448-5611.
(IC Products) SIGNETICS JAPAN, LTD., TOKYO, Tel. (03) 230-1521.
- Korea:** PHILIPS ELECTRONICS (KOREA) LTD., Philips House, 260-199 Itaewon-dong, Yongsan-ku, C.P.O. Box 3680, SEOUL, Tel. 44-4202.
- Mexico:** ELECTRONICA S.A. de C.V., Varsovia No. 36, MEXICO 6, D.F., Tel. 5-33-11-80.
- Netherlands:** PHILIPS NEDERLAND B.V., Afd. Elonco, Boschdijk 525, NL-4510 EINDHOVEN, Tel. (040) 79 33 33.
- New Zealand:** Philips Electrical Ind. Ltd., Elcoma Division, 2 Wagener Place, St. Lukes, AUCKLAND, Tel. 867 119.
- Norway:** ELECTRONICA A/S., Vitaminveien 11, P.O. Box 29, Grefsen, OSLO 4, Tel. (02) 15 05 90.
- Peru:** CADESA, Jr. Ilo, No. 216, Apartado 10132, LIMA, Tel. 27 73 17.
- Philippines:** ELDAC, Philips Industrial Dev. Inc., 2246 Pasong Tamo, MAKATI-RIZAL, Tel. 86-89-51 to 59.
- Portugal:** PHILIPS PORTUGESA S.A.R.L., Av. Eng. Duharte Pacheco 6, LISBOA 1, Tel. 68 31 21.
- Singapore:** PHILIPS SINGAPORE PTE LTD., Elcoma Div., POB 340, Toa Payoh CPO, Lorong 1, Toa Payoh, SINGAPORE 12, Tel. 53 88 11.
- South Africa:** EDAC (Pty.) Ltd., South Park Lane, New Doornfontein, JOHANNESBURG 2001, Tel. 24/6701.
- Spain:** COPRESA S.A., Balmes 22, BARCELONA 7, Tel. 301 63 12.
- Sweden:** A.B. ELCOMA, Lidingövägen 50, S-10 250 STOCKHOLM 27, Tel. 08/67 97 80.
- Switzerland:** PHILIPS A.G., Elcoma Dept., Edenstrasse 20, CH-8027 ZÜRICH, Tel. 01/44 22 11.
- Taiwan:** PHILIPS TAIWAN LTD., 3rd Fl., San Min Building, 57-1, Chung Shan N. Rd, Section 2, P.O. Box 22978, TAIPEI, Tel. 5513101-5.
- Turkey:** TÜRK PHILIPS TICARET A.S., EMET Department, Inonu Cad. No. 78-80, ISTANBUL, Tel. 43 59 10.
- United Kingdom:** MULLARD LTD., Mullard House, Torrington Place, LONDON WC1E 7HD, Tel. 01-580 6633.
- United States:** (Active devices & Materials) AMPEREX SALES CORP., 230, Duffy Avenue, HICKSVILLE, N.Y. 11802, Tel. (516) 931-6200.
(Passive devices) MEPCO/ELECTRA INC., Columbia Rd., MORRISTOWN, N.J. 07960, Tel. (201) 539-2000.
(IC Products) SIGNETICS CORPORATION, 811 East Arques Avenue, SUNNYVALE, California 94086, Tel. (408) 739-7700.
- Uruguay:** LUZIELECTRON S.A., Rondeau 1567, piso 5, MONTEVIDEO, Tel. 9 43 21.
- Venezuela:** IND. VENEZOLANAS PHILIPS S.A., Elcoma Dept., A. Ppal de los Ruices, Edif. Centro Colgate, Apdo 1167, CARACAS, Tel. 36 05 11.

signetics

MICROPROCESSOR



2650 EVALUATION PRINTED
CIRCUIT BOARD LEVEL
SYSTEM (PC1001).....SP50

APPLICATIONS MEMO

GENERAL

The PC1001 is an evaluation and design tool for the 2650 microprocessor. Each PC1001 board has a 2650 microprocessor, 1k bytes of RAM, 1k bytes of PROM loaded with PIPBUG*, a crystal clock, and sufficient additional logic to allow the user to exercise all aspects of the 2650 microprocessor. There is a serial I/O port on the board that can be used to drive a current loop driven terminal or an RS232 type terminal. The PC1001 provides the system engineer with a very flexible design tool from which he can easily develop a pre-production prototype of his product designed around the 2650 microprocessor.

FEATURES

The PC1001 has many features that make it a valuable design aid. The most noteworthy features are:

- The Signetics 2650 N-MOS, 8-bit microprocessor
- 1k – bytes of RAM memory
- 1k – bytes of PROM memory
- A 1MHz crystal oscillator
- A serial I/O channel
- Two Non-Extended 8-bit parallel input ports

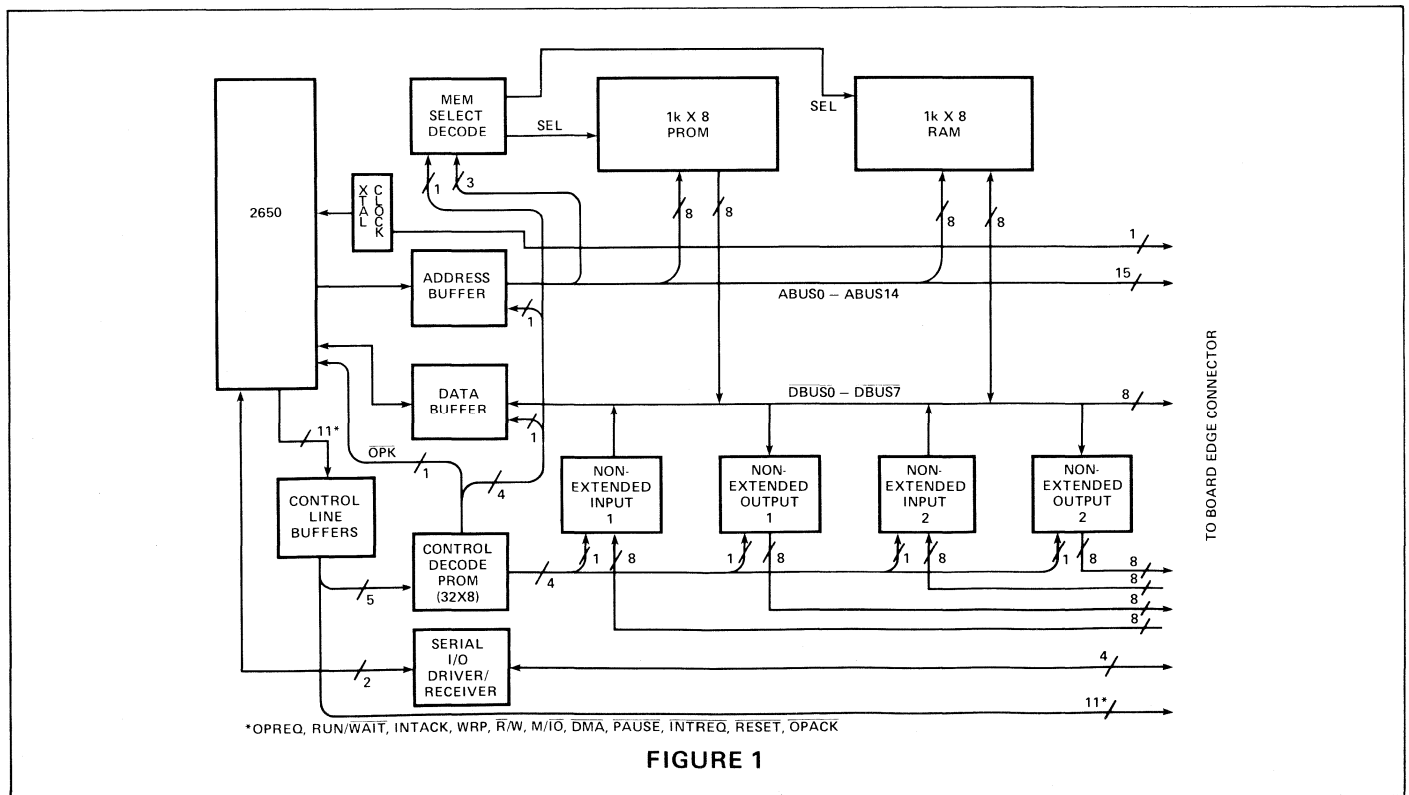
- Two Non-Extended 8-bit parallel output ports
- Buffered address, data, and control lines for implementing additional 8-bit parallel I/O ports or expanded memory
- Direct Memory Access (DMA) capability, including the memory on the PC1001 board
- Display indicators on the board for the RUN/ $\overline{\text{WAIT}}$, OPREQ, $\text{M}/\overline{\text{IO}}$, $\text{R}/\overline{\text{W}}$ control lines, and the Non-Extended output ports
- Vectored interrupts
- A program debug module (called PIPBUG) written for use with the 2650

*PIPBUG – a program debug module

DESCRIPTION

The PC1001 is configured as a very flexible, general purpose microprocessor board to allow the system designer to easily expand memory, implement input/output functions and execute programs written for the 2650. A functional description of the PC1001 is given in this section. A functional block diagram of the PC1001 is shown in Figure 1.

PC 1001 BLOCK DIAGRAM



CPU

The 2650 is the heart of the PC1001, executing instructions from memory and controlling the I/O functions. The address, data, and control lines of the 2650 are buffered and available at the edge connector of the PC1001. The on-board bus drivers allow the user to build a microprocessor system around the PC1001 without additional buffering. The tri-state function of the 2650 address and data busses is transferred to the buffer gates which drive the lines used by the system designer. The address and data bus buffers are in the tri-state mode whenever the OPREQ line from the 2650 is a logic ZERO.

MEMORY

The 1024 bytes of read only memory are implemented with 82S129 256X4 bipolar PROM's. The PROM's are accessed by addressing the first 1024 bytes of the address space (locations $0_{16} - 3FF_{16}$). The PROM's are mounted in sockets on the PC1001 board and are loaded with the PIPBUG debug program. The sockets on the PC1001 board allow the user to put different 82S129 PROM's in the first 1k bytes of the memory address space when developing a prototype system.

The 1024 bytes of random access memory are implemented with 2606 256X4 MOS RAM's. The RAM's are accessed by addressing the second 1024 bytes of the address space (locations $400_{16} - 7FF_{16}$).

PARALLEL I/O

The buffered address, data, and control lines available to the user of the PC1001 allow any of the 2650 parallel I/O modes to be implemented, or to expand memory beyond the 2k bytes already on the board. The extended I/O instructions provide device select capability for 256 I/O functions by decoding the least significant 8-bits of the address bus (ABUS 0 - ABUS 7). The buffered data bus is a bi-directional tri-state bus so that input devices may use the data bus by driving it with tri-state drivers.

If the Non-Extended I/O instructions are used, two latched output ports and two gated input ports are already provided on the PC1001, and no control line decoding is necessary.

When the 2650 executes memory reference instructions or Non-Extended I/O instructions, the control decode PROM generates the operation acknowledge signal (\overline{OPACK}) in response to operation request (OPREQ). When the 2650 executes Extended I/O instructions, the selected I/O device must generate \overline{OPACK} . By requiring the I/O device to return the \overline{OPACK} signal, the PC1001 gives the user the flexibility of connecting peripheral functions that may require more than one microsecond to respond to an I/O request. If the Extended I/O functions are all faster than one microsecond they will not slow down the 2650, and \overline{OPACK} may be tied to logic ZERO.

SERIAL I/O

The 2650 is equipped with a SENSE input and a FLAG output. These two functions provide a serial I/O data path directly into the 2650. Part of the PIPBUG PROM program

is dedicated to implementing an asynchronous serial communications port for the PC1001. The program checks the SENSE line for a start bit from the serial device to achieve synchronization. Once a start bit is detected, the 2650 shifts the next eight character bits into register R0. The PC1001 is designed for full duplex serial I/O, and will echo the transmitted character back to the serial device using the FLAG output. The timing loops that determine when to sample a character bit are written for a ten character per second serial data rate (110 baud), but the 2650 is capable of handling much higher serial data rates.

The serial I/O device used with the PC1001 may be a 20 milliamp current loop device, or it may be RS232 compatible (voltage driven). A current loop driver and receiver, and an RS232 driver and receiver are on the PC1001 board. The type of driver and receiver is selected with a wire jumper. If the RS232 driver and receiver is used, external ± 15 volt power supplies are required. If the current loop driver and receiver is used, the PC1001 requires only a single +5 volt power supply.

The PIPBUG debug program includes a read paper tape control function. The program sets a bit in the output register of Non-Extended I/O port C (WRTC instruction) to advance the tape reader one character at a time. This function can be used by modifying a standard teletype to include a tape reader control relay and driving it with the TTY TAPE READER OUT SIGNAL.

It should be pointed out that the tape reader control bit and bit 7 of the Non-Extended I/O port (OPC7) are the same and caution should be exercised to avoid a conflict between the two functions.

CLOCK

The clock circuit on the PC1001 is a hybrid circuit crystal oscillator that runs at a frequency of 1.000 MHz. Instruction loops are used to determine bit times and the crystal controlled clock minimizes errors due to changes in the system clock.

The clock input to the 2650 that is driven by the crystal controlled clock (pin 38) is available at the edge connector of PC1001. If the user chooses to drive the PC1001 with an external clock he must first remove the crystal clock circuit. The clock input to the 2650 is fully TTL compatible and requires no special drive circuitry.

DISPLAYS

Minature LED indicator displays are driven by the three basic control lines (OPREQ, M/\overline{IO} , and R/\overline{W}), and the Non-Extended output latches. A logic ONE state on the control lines, or in the output latches, "lights" the corresponding LED. The minature LED's are mounted on the PC1001 board and are shown in Figure 2.

PRINTED CIRCUIT BOARD LAYOUT

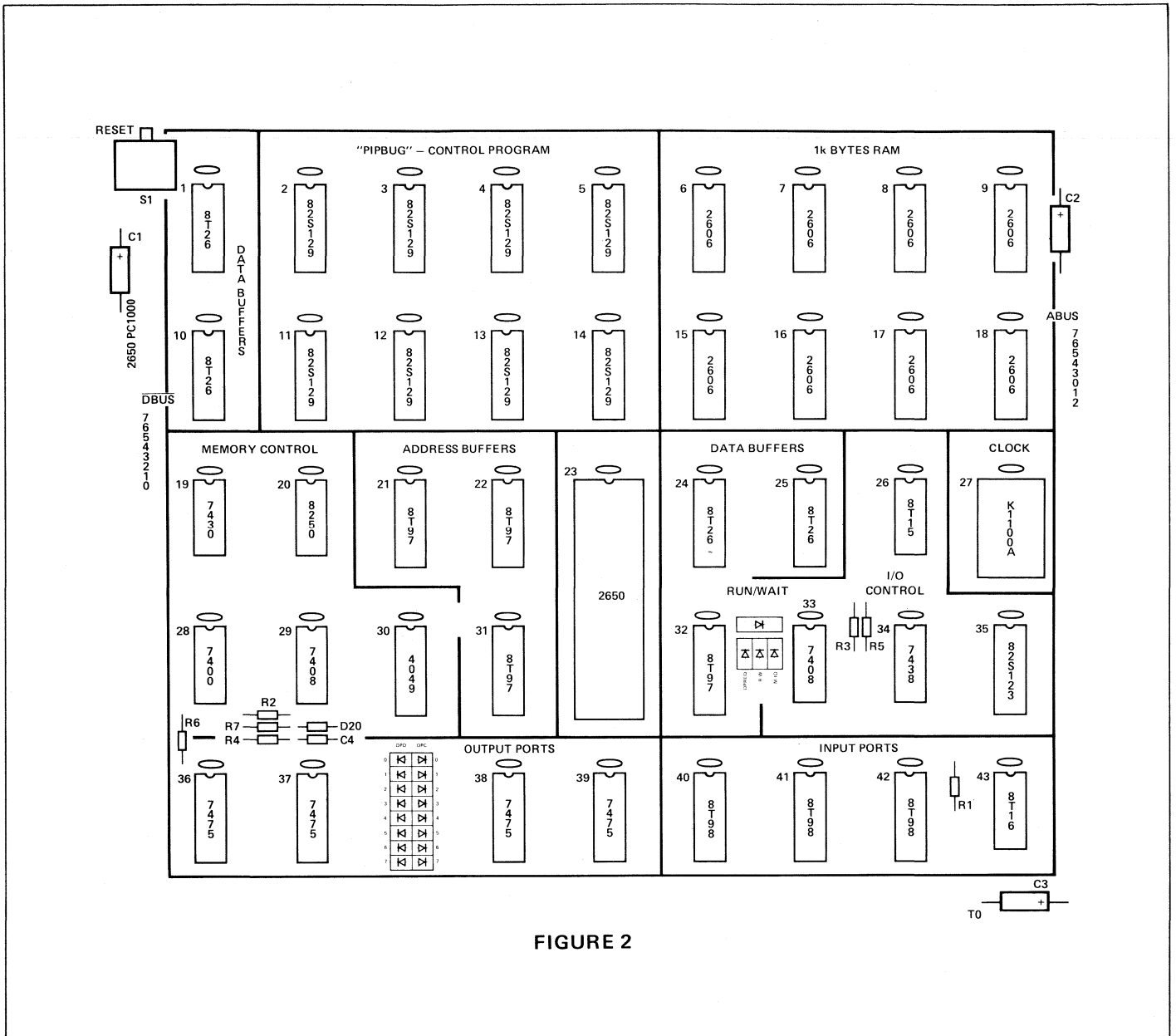


FIGURE 2

DMA

Direct access to memory by an external device (DMA) is easily accomplished with the PC1001. An input to the board is provided for direct memory access and the signal name of that input is DMA (PC1001 pin 14). When DMA is pulled "low" the 2650 finishes executing the current instruction and enters the wait state. To avoid interrupting a memory or I/O transfer in progress the DMA line should not be pulled "low" while OPREQ is "high". When the RUN/WAIT lines goes "low" the external device may drive the address, data, and control lines (except OPREQ, and RUN/WAIT) to accomplish the necessary DMA transfer.

An external operation request line (OPEX) is provided for DMA transfers to the memory on the PC1001 board. Since OPREQ is only driven by the 2650, and is used in the memory select decoders, the user must drive OPEX to access the memory on the PC1001.

Because the DMA function is implemented with the pause feature of the 2650, and since the 2650 is a static device, the length of time that the DMA device may be active for any one transfer is limited only by the other processing responsibilities of the 2650.

INTERRUPTS

The 2650 has a true vectored interrupt system. The user must first drive the interrupt line (INTREQ) on the PC1001, then wait to be acknowledged (INTACK), and finally drive the data bus with a 7-bit signed displacement relative to page zero, location zero. The displacement vector may also indicate indirect addressing, allowing the interrupt service sub-routine to be located anywhere in the 32k-byte address space.

INTERRUPTS (Continued)

The $\overline{\text{INTREQ}}$ line may be driven by several interrupting devices in a "wired OR" configuration. When a priority exists between the various interrupting devices, and to prevent confusion from multiple simultaneous interrupts, the user must arrange the interrupt hardware to resolve priority and simultaneity conflicts.

The PC1001 board comes with PIPBUG stored in the first 1k-bytes of .ROM and therefore the user cannot store an interrupt service subroutine or an indirect address in this part of the memory address space. But the interrupt displacement vector may be a negative number referring to the last 64 locations in page zero (1FBF_{16} to 1FFF_{16}). If an indirect address or interrupt service subroutine is placed in one of the last 64 locations of page zero, the user must also provide external memory at the locations used (the PC1001 has only 2k-bytes of memory on the board).

There is another way to accomplish a "link" to an interrupt service subroutine through the ROM on the PC1001. It is possible that PIPBUG instructions themselves could provide an indirect address to the second 1k-bytes of RAM on the PC1001 board. An example of a very useable indirect address to an interrupt service routine may be found at locations 8_{16} and 9_{16} of PIPBUG. If these locations are used as an indirect address, the program would branch to location 477_{16} where it would expect to find a subroutine to service the active interrupt.

A timing diagram for interrupt processing is shown in Figure 3, as well as the format for the displacement vector.

INTERRUPT TIMING

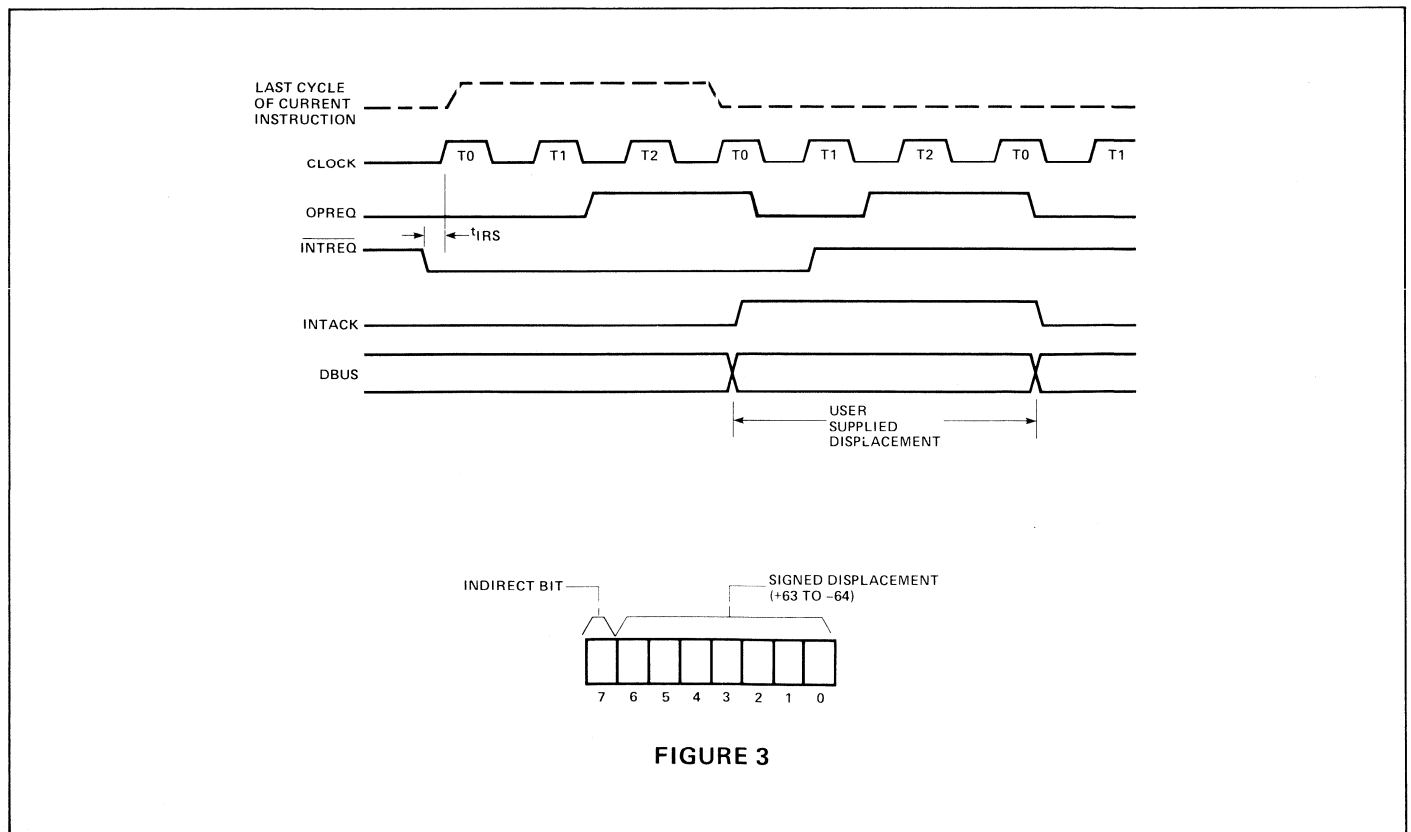


FIGURE 3

LOGIC

The logic on the PC1001 board is uncomplicated and very general purpose. It includes:

1. 2650 CPU and memory
2. Address bus, and data bus drivers and receivers
3. Control line drivers and receivers
4. Control line decode
5. Memory select decode
6. Serial I/O transmitter and receiver
7. Non-Extended parallel I/O latches and receivers

The PC1001 logic drawing will be referred to during this description and is shown in Figure 4. The integrated circuit numbers used in Figure 4 may be cross-correlated to those used on Figure 2 for locating an integrated circuit on the PC1001 board.

CPU and MEMORY

CPU – The address bus, and the data bus from the 2650 are buffered for easy system expansion. With the exception of the address tri-state control line ($\overline{\text{ADREN}}$) and the data bus tri-state control line ($\overline{\text{DBUSEN}}$), all of the control lines from the 2650 are also buffered. The $\overline{\text{ADREN}}$ and $\overline{\text{DBUSEN}}$ lines are tied "low" on the PC1001 board, and the tri-state function of the address, data, and control lines is fulfilled by the buffers.

The clock input is driven directly from the K1100A clock circuit (IC #27). The clock output is available off-board on PC1001 pin 23 (the signal name is CLOCK). The K1100 clock circuit has a frequency stability of $\pm 0.1\%$ and will drive 10 standard TTL (7400 series) unit loads. The 2650

PC1001 LOGIC DIAGRAM

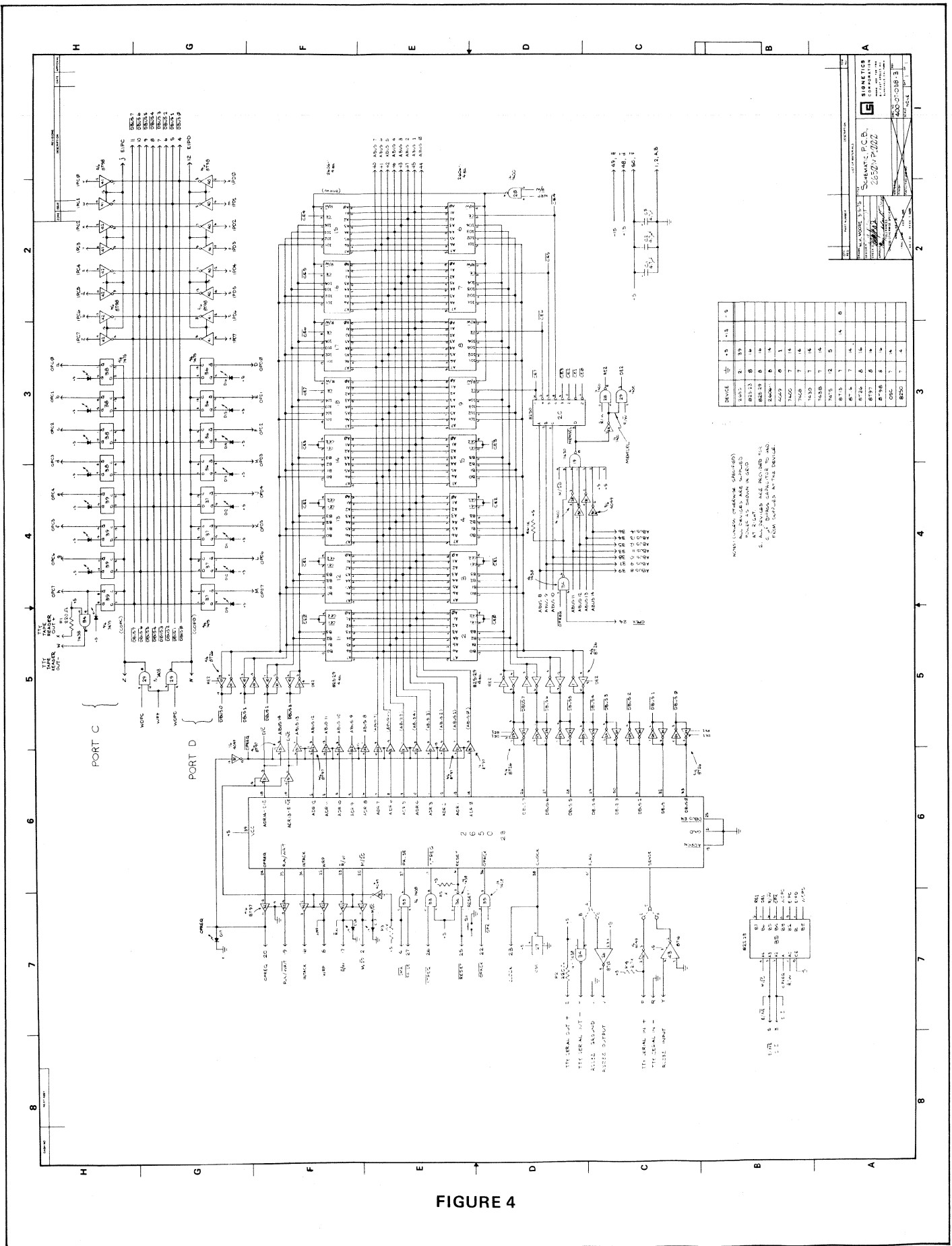


FIGURE 4

CPU and MEMORY (Continued)

is the only load the clock must drive on the PC1001, using only 10 μ amps of its drive capability.

Memory — The memories are of two types: 82S129 256X4 PROMs, and 2606 256X4 RAMs. All 16 memory IC's (IC's 2-9, and 11-18) are addressed by the least significant 8-bits of the buffered address bus. The memories drive and receive the data bus through 8T26 tri-state transceivers to prevent an expanded system from presenting too great a capacitive load for the MOS memories.

The PROM memories (IC's 2-5, and IC's 11-14) are plugged into sockets and come programmed with PIPBUG. Any user's program may be stored in these PROM locations if PIPBUG is not required.

When the PC1001 is used to develop programs, and PIPBUG is resident in the first 1k-bytes of memory space, all of the 1k-bytes of RAM memory is available for use except the first 64 bytes (400₁₆ to 43F₁₆). The 64 locations are used by PIPBUG for temporary storage.

ADDRESS AND DATA BUS DRIVERS AND RECEIVERS

The data bus (DBUS0 - DBUS7) is buffered with 8T26 quad tri-state transceivers (IC's 24 and 25). These transceivers are inverting, and therefore the data bus transferred off of the PC1001 board is negative true ($\overline{\text{DBUS0}} - \overline{\text{DBUS7}}$). The tri-state transceivers are controlled by RE1 (receiver control) and DE1 (driver control) from the control decode PROM (IC 35). The receiver control RE1 is a negative true signal (active "low") and has the following logic equation:

$$\text{RE1} = \text{OPREQ} \bullet \overline{\text{R/W}}$$

The driver control DE1 is a positive true signal and has the following logic equation:

$$\text{DE1} = \text{OPREQ} \bullet \overline{\text{R/W}}$$

The logic equations reflect the fact that the 2650 drives the external data bus ($\overline{\text{DBUS0}} - \overline{\text{DBUS7}}$) during all write operations (memory or I/O), and receives the external data bus during all read operations. But, when OPREQ is not a "high" the external data bus transceivers are in the tri-state mode.

The memory on the PC1001 board is buffered from the user's data bus ($\overline{\text{DBUS0}} - \overline{\text{DBUS7}}$) with 8T26 quad tri-state transceivers. These transceivers are inverting so that information stored in memory is not complimented relative to the 2650. These transceivers are controlled by RE2 (receiver control) and DE2 (driver control) from the memory select decode logic. The logic for these control lines is shown below IC 20 (IC's 28 and 29) in Figure 4 and they have the following logic equations:

$$\begin{aligned} \text{RE2} &= \text{MEMSEL} \bullet \overline{\text{R/W}} \\ \text{DE1} &= \text{MEMSEL} \bullet \overline{\text{R/W}} \end{aligned}$$

The RE2 control line is a negative true signal and is active when the memory on the PC1001 is selected to be written into. The DE1 control line is positive true and active when the memory on the PC1001 is selected to be read from.

The address bus is buffered with 8T97 tri-state buffers (IC's 21, 22, and 31). These buffers are in the tri-state mode whenever OPREQ is inactive.

CONTROL LINE DRIVERS AND RECEIVERS

The two control lines OPREQ and $\overline{\text{RUN/WAIT}}$ are buffered with 8T97 tri-state buffers (IC 32), but are never placed in the tri-state mode.

The control lines INTACK, WRP, $\overline{\text{R/W}}$, and $\overline{\text{M/I/O}}$ are also driven by 8T97 tri-state buffers (IC 32), and are switched to the tri-state mode when the $\overline{\text{DMA}}$ line is pulled "low". The pause input to the 2650 may be activated by driving the $\overline{\text{DMA}}$ line (PC1001 pin 14) or the $\overline{\text{PAUSE}}$ line (PC1001 pin 27) "low".

The interrupt request line and the reset line to the 2650 are buffered by TTL AND gates (IC 33). The reset switch on the PC1001 (upper left corner of Figure 2) is "wire ORed" with the $\overline{\text{RESET}}$ line to the PC1001 board (PC1001 pin 25).

The operation acknowledge line to the 2650 ($\overline{\text{OPACK}}$) is buffered with a TTL AND gate (IC 33), and has as its inputs an external acknowledge ($\overline{\text{OPACK}}$, PC1001 pin 22) and an internal acknowledge (OPK). The internal acknowledge is generated for all memory access cycles and Non-Extended I/O cycles initiated by the 2650. For Extended I/O cycles the external device must generate the external operation acknowledge ($\overline{\text{OPACK}}$).

CONTROL LINE DECODE

A control line decoder is implemented with a 32X8 PROM (82S123) to generate secondary control lines used by the logic supporting the 2650. The primary control lines from the 2650 ($\overline{\text{R/W}}$, OPREQ, $\overline{\text{M/I/O}}$ E/ $\overline{\text{NE}}$, and $\overline{\text{D/C}}$) are used to address the PROM, and each address represents one combination of the primary control lines. Stored at each memory location are eight bits, each one of which represents the logical state of a secondary control line. There are five address inputs to the PROM, and the 32 (2^5) possible addresses exhaust all of the logical combination of the primary control lines. The secondary control lines, their logic equations, and their functions are given in Table 1. Table 2 shows the contents of each of the 32 locations of the PROM. The control line decode PROM is shown in Figure 4 (IC 35).

MEMORY SELECT DECODE

The memory select decode logic is shown in Figure 4 (IC's 19, 20, 28, 29, 30 and 34). The 2k-bytes of memory are implemented with 256X4 bit memory chips. The memory chips are arranged into eight 256-byte sections.

The ninth, tenth, and eleventh bits of the address bus (ABUS8-ABUS10) are decoded to select one of the eight 256-byte sections of memory. The one-of-eight decoder (IC 20) is enabled by MEMSEL, which has the following logic equations:

$$\overline{\text{MEMSEL}} = (\text{OPREQ} + \text{OPEX}) \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{ABUS11}} \bullet \overline{\text{ABUS12}} \bullet \overline{\text{ABUS13}} \bullet \overline{\text{ABUS14}}$$

The MEMSEL line is also used to enable the 8T26 quad tri-state transceivers that buffer the memory on the PC1001 from the external data bus ($\overline{\text{DBUS0}}\text{-}\overline{\text{DBUS7}}$).

SERIAL I/O TRANSMITTER AND RECEIVER

A serial I/O port is implemented on the PC1001 with the flag and sense line of the 2650. The PIPBUG program handles the serial I/O using software timing loops to sample

the SENSE input and build eight bit ASCII characters. The PC1001 is capable of interfacing to a current loop type terminal, or an RS232 compatible terminal.

The current loop driver uses an open collector NAND gate (IC 34) as the switching element. The 20 milliamp source is a 220Ω resistor connected to +5 volts on the PC1001 (PC1001 pin S), and the open collector NAND gate either provides a return path for the 20 milliamps (NAND output "on") or it does not (NAND output "off"). The current loop receiver is a CMOS hex inverter (IC 30) with the input pulled to +5 volts through a $2.7\text{k}\Omega$ resistor (PC1001 pin P). The teletype transmitter is a contact closure and connects the input of the CMOS inverter to the receiver return line (PC1001 pin R), which is tied to ground on the PC1001 board.

The RS232 driver is an 8T15 EIA Line Driver (IC 26), and the RS232 receiver is an 8T16 EIA Line Receiver (IC 43). The 8T15 is the only chip on the PC1001 that does not operate on the +5 volt power supply, and ± 15 volt power supplies are specified for this driver.

CONTROL LINE DECODE PROM DESCRIPTION

SIGNAL NAME	OUTPUT	PIN #	LOGIC EQUATION	FUNCTION
WOPD	B0	1	$\text{WOPD} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\text{R}/\overline{\text{W}}}$	LOADS NON-EXTENDED OUTPUT LATCH, PORT D
EIPD*	B1	2	$\overline{\text{EIPD}} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\text{R}/\overline{\text{W}}}$	ENABLES NON-EXTENDED INPUT GATES, PORT D
EIPC*	B2	3	$\overline{\text{EIPC}} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\text{R}/\overline{\text{W}}}$	ENABLES NON-EXTENDED INPUT GATES, PORT C
WOPC	B3	4	$\text{WOPC} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\text{R}/\overline{\text{W}}}$	LOADS NON-EXTENDED OUTPUT LATCH, PORT C
OPK*	B4	5	$\overline{\text{OPK}} = \text{OPREQ} [(\overline{\text{M}/\overline{\text{IO}}}) + (\overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}})]$	RETURNS $\overline{\text{OPACK}}$ FOR ALL OPREQ EXCEPT EXTENDED I/O
$\overline{\text{R}/\overline{\text{W}}}$	B5	6	$\overline{\text{R}/\overline{\text{W}}} = \overline{\overline{\text{R}/\overline{\text{W}}}}$	INVERTS $\overline{\text{R}/\overline{\text{W}}}$
DE1	B6	7	$\text{DE1} = \text{OPREQ} \bullet \overline{\text{R}/\overline{\text{W}}}$	DRIVES EXTERNAL DATA BUS ($\overline{\text{DBUS0}}\text{-}\overline{\text{DBUS7}}$)
RE1*	B7	9	$\overline{\text{RE1}} = \text{OPREQ} \bullet \overline{\overline{\text{R}/\overline{\text{W}}}}$	ENABLES RECEIVERS OF EXTERNAL DATA BUS ($\overline{\text{DBUS0}}\text{-}\overline{\text{DBUS7}}$)

*NEGATIVE TRUE SIGNALS

TABLE 1

CONTROL LINE DECODE PROM

ADDRESS	INPUT					OUTPUT								OUTPUT ₁₆
	A4	A3	A2	A1	A0	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	1	1	0	1	1	0	B6
1	0	0	0	0	1	1	0	0	1	0	1	1	0	96
2	0	0	0	1	0	0	0	1	0	0	0	1	0	22
3	0	0	0	1	1	1	1	0	0	1	1	1	0	CE
4	0	0	1	0	0	1	0	1	1	0	1	1	0	B6
5	0	0	1	0	1	1	0	0	1	0	1	1	0	96
6	0	0	1	1	0	0	0	1	0	0	1	0	0	24
7	0	0	1	1	1	1	1	0	0	0	1	1	1	C7
8	0	1	0	0	0	1	0	1	1	0	1	1	0	B6
9	0	1	0	0	1	1	0	0	1	0	1	1	0	96
10	0	1	0	1	0	0	0	1	1	0	1	1	0	36
11	0	1	0	1	1	1	1	0	1	0	1	1	0	D6
12	0	1	1	0	0	1	0	1	1	0	1	1	0	B6
13	0	1	1	0	1	1	0	0	1	0	1	1	0	96
14	0	1	1	1	0	0	0	1	1	0	1	1	0	36
15	0	1	1	1	1	1	1	0	1	0	1	1	0	D6
16	1	0	0	0	0	1	0	1	1	0	1	1	0	B6
17	1	0	0	0	1	1	0	0	1	0	1	1	0	96
18	1	0	0	1	0	0	0	1	0	0	1	1	0	26
19	1	0	0	1	1	1	1	0	0	0	1	1	0	C6
20	1	0	1	0	1	1	0	0	1	0	1	1	0	B6
21	1	0	1	0	1	1	0	0	1	0	1	1	0	96
22	1	0	1	1	0	0	0	1	0	0	1	1	0	26
23	1	0	1	1	1	1	1	0	0	0	1	1	0	C6
24	1	1	0	0	0	1	0	1	1	0	1	1	0	B6
25	1	1	0	0	1	1	0	0	1	0	1	1	0	96
26	1	1	0	1	0	0	0	1	0	0	1	1	0	26
27	1	1	0	1	1	1	1	0	0	0	1	1	0	C6
28	1	1	1	0	0	1	0	1	1	0	1	1	0	B6
29	1	1	1	0	1	1	0	0	1	0	1	1	0	96
30	1	1	1	1	0	0	0	1	0	0	1	1	0	26
31	1	1	1	1	1	1	1	0	0	0	1	1	0	C6
	M/ \overline{IO}	E/ \overline{NE}	D/ \overline{C}	O P R E Q	$\overline{R/W}$	REI	DEI	R/ \overline{W}	O P K	W O P C	E I P C	E I P D	W O P D	

TABLE 2

The current loop driver/receiver pair or the RS232 driver/receiver pair is selected by a hardwire jumper on the PC1001 board. The connection of these jumpers is described in Table 3, and shown in Figure 4 (2650 pin 40/FLAG, 2650 pin 1/SENSE).

SERIAL I/O DRIVER/RECEIVER MODE

2650 FUNCTION	JUMPER	DESCRIPTION
FLAG	A-B	CURRENT LOOP DRIVER
FLAG	A-C	RS232 DRIVER
SENSE	E-D	CURRENT LOOP RECEIVER
SENSE	F-D	RS232 RECEIVER

TABLE 3

PARALLEL I/O LATCHES AND RECEIVERS

The logic used to implement the two parallel I/O ports on the PC1001 is identical. The output ports are 7475 quad bistable latches (IC's 36, 37, 38, and 39), and are loaded when a Non-Extended write I/O instruction is executed (WRTC, WRTD). The input ports use 8T98 tri-state high speed hex inverters (IC's 40, 41, and 42), and are gated on the external data bus ($\overline{DBUS0}$ - $\overline{DBUS7}$) when a Non-Extended read I/O instruction is executed (REDC, REDD).

The control signals used to activate the tri-state gates (EIPD, and EIPC) are generated by the control line decode PROM (IC 35).

The control signals used to load the output latches are designated COPC and COPD, and have the following logic equations:

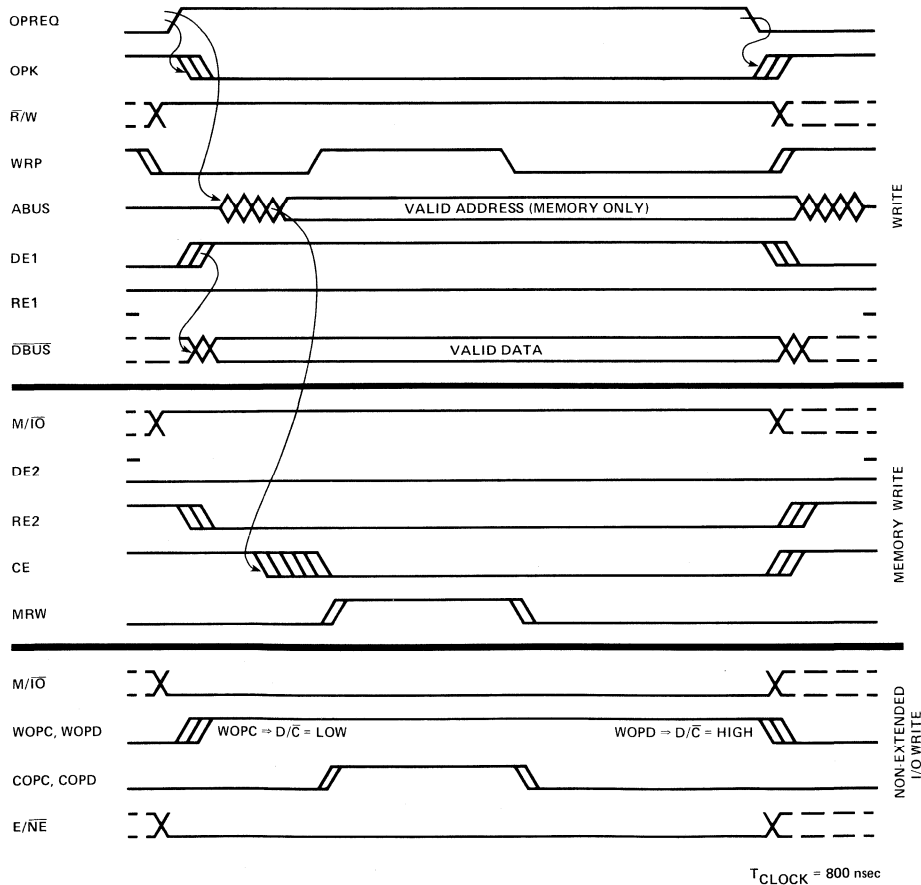
$$\begin{aligned} \text{COPD} &= \text{WRP} \bullet \text{WOPD} \\ \text{COPC} &= \text{WRP} \bullet \text{WOPC} \end{aligned}$$

The WRP signal is the "write pulse" from the 2650, while the WOPD and WOPC signals are generated by the control line decode PROM (IC 35).

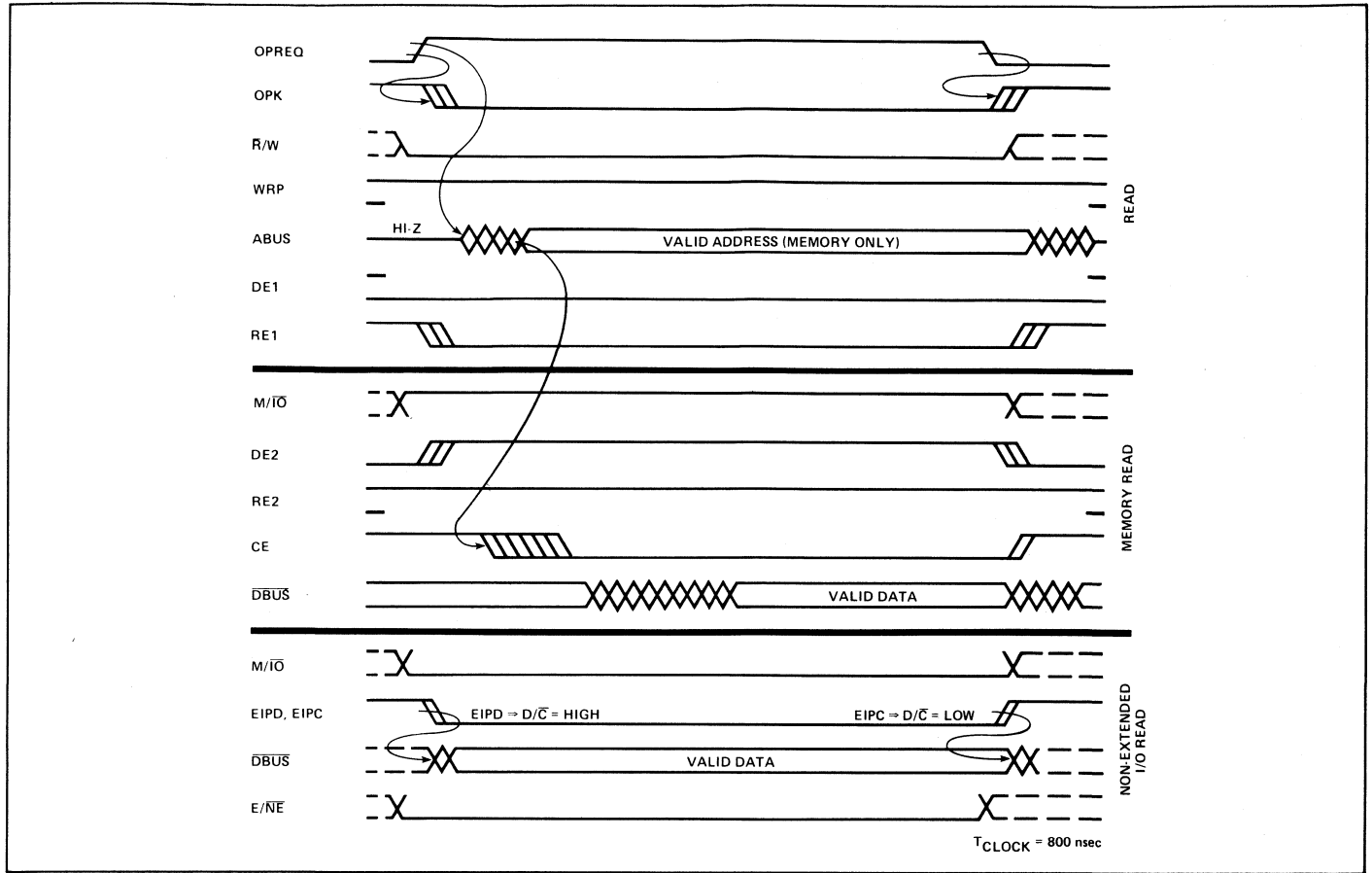
The output latches drive LED's on the PC1001 board. A logic ONE from the 2650 lights the corresponding LED. The output latches are loaded from the external data bus (DBUS0 - DBUS7), and to obtain the required inversion at the latch output (OPD0 - OPD7, and OPC0 - OPC7) the \bar{Q} pin is used.

APPENDIX

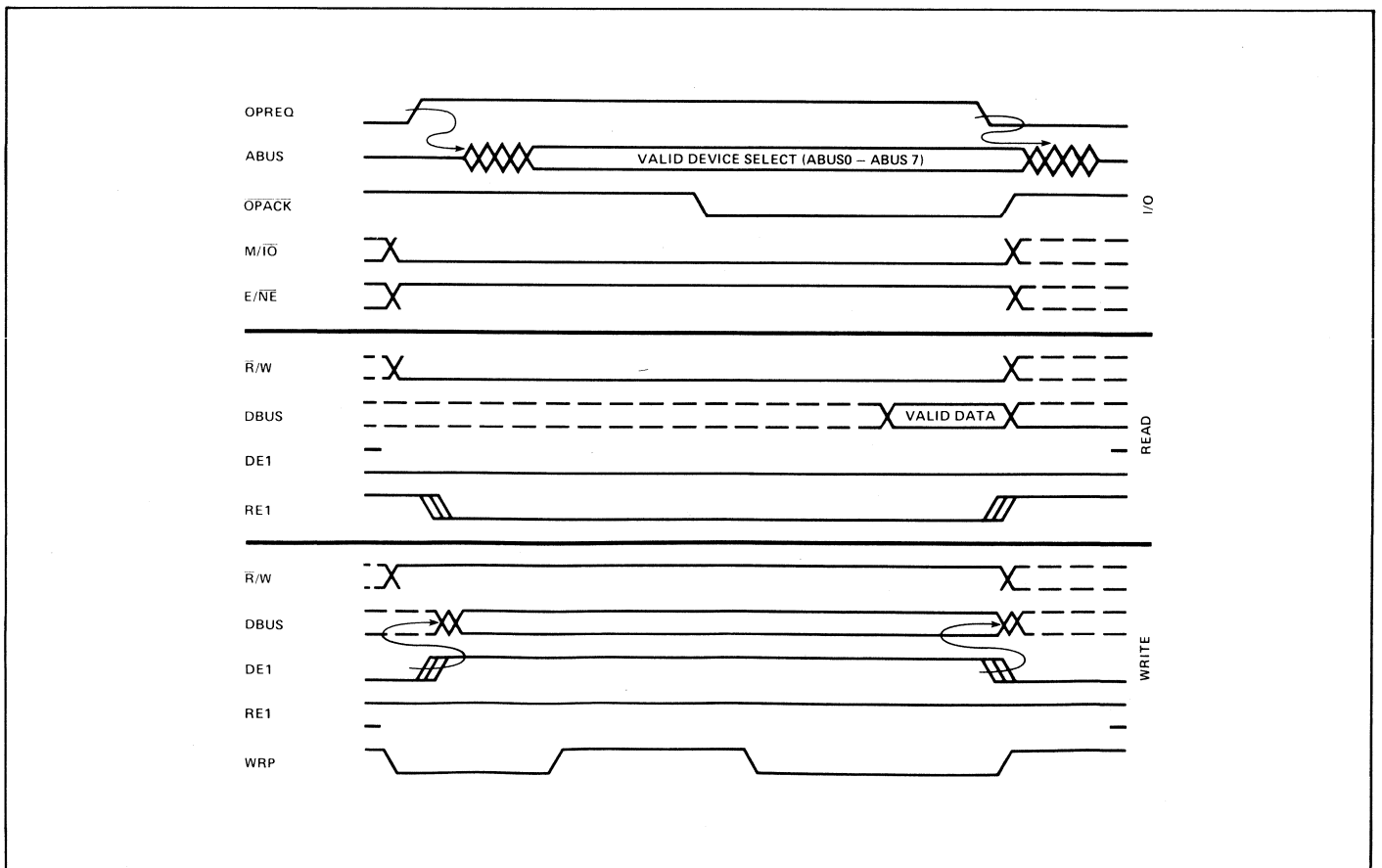
WRITE TIMING



READ TIMING



EXTENDED I/O TIMING



POWER REQUIREMENTS

+5 VOLT POWER SUPPLY:

LINE REGULATION – 0.1%
 LOAD REGULATION – 0.1%
 RIPPLE – 10 millivolts (MAX)
 RESPONSE – 30 μ sec (MAX)
 CURRENT – 2 amps

 \pm 15 VOLT POWER SUPPLIES:

LINE REGULATION – 0.1%
 LOAD REGULATION – 0.1%
 RIPPLE – 10 millivolts (MAX)
 RESPONSE – 30 μ sec (MAX)
 CURRENT – 50 milliamps

PARTS LIST

IC #	PART #	TYPE	QTY
28	7400	QUAD 2-INPUT NAND	1
29, 33	7408	QUAD 2-INPUT AND	2
19	7430	8-INPUT NAND	1
34	7438	QUAD 2-INPUT NAND OPEN COLLECTOR	1
36, 37, 38, 39	7475	QUAD BISTABLE LATCH	4
26	8T15	E1A DRIVER (RS232)	1
43	8T16	E1A RECEIVER (RS232)	1
1, 10 24, 25	8T26	QUAD BUS DRIVER/RECEIVER	4
40, 41, 42	8T98	HEX HIGH SPEED INVERTER	3
20	8250	1 OF 8 DECODER	1
6, 7, 8, 9 15, 16, 17, 18	2606	256X4 NMOS RAM	8
30	4049	HEX INVERTER (CMOS)	1
35	82S123	32X8 BIPOLAR PROM	1
2	82S129	PIPBUG PROM CK267	1
11	82S129	PIPBUG PROM CK268	1
3	82S129	PIPBUG PROM CK269	1
12	82S129	PIPBUG PROM CK270	1
4	82S129	PIPBUG PROM CK271	1
13	82S129	PIPBUG PROM CK272	1
5	82S129	PIPBUG PROM CK273	1
14	82S129	PIPBUG PROM CK274	1
23	2650	MICROPROCESSOR	1
27	MOTOROLA K1100A	XTAL OSCILLATOR	1
D20	IN914	DIODE	1
D1-D19	DIALCO 555-3007	LED INDICATOR	19
S1	GREYHILL 39-201	MINIATURE, PUSH BUTTON SWITCH	1
(IC 23)*	VERMON H23-20302	40-PIN DIP SOCKET	1
(IC 2, 3 4, 5, 11, 12, 13, 14)	AMPHENOL 821-25011-164	16-PIN DIP SOCKET	8

*#s in parenthesis indicate the IC's that are plugged into the listed socket.

PARTS LIST (Continued)

IC #	PART #	TYPE	QTY
(IC 27)	AMPHENOL 821-25011-144	14-PIN DIP SOCKET	1
C1, C2 C3	230-1250-004-230	4.7 μ FARAD CAP	3
—	EMCON 5021ES50RD104M	0.1 μ FARAD CAP	45
C4		0.047 μ FARAD CAP	1
R4	230-0910-332-230	51Kr, 1/4 WATT RES	1
R3		10Kr, 1/4 WATT RES	1
R7		7.4Kr, 1/4 WATT REST	1
R5, R6	230-0910-297-230	1Kr, 1/4 WATT RES	1
R1, R2	230-0910-282-230	220r, 1/4 WATT RES	2
	AMPHENOL 225-804-50	100 PIN P.C. EDGE CONNECTOR	1

RS232C STANDARD CONNECTOR

The RS232 Electronic Industries Association (EIA) standard for "interface between terminals and communications equipment using serial binary data interchange" describes a commonly used signal definition and connector pin assignment. The table below lists the pin numbers and signal names most frequently used by data terminals.

PIN #	DESCRIPTION
1	PROTECTIVE GROUND
2	TRANSMITTED DATA
3	RECEIVED DATA
5	CLEAR TO SEND
6	DATA SET READY
7	SIGNAL GROUND
8	RECEIVED LINE SIGNAL DETECTOR
20	DATA TERMINAL READY

Transmitted Data (pin 2) is received by the PC1001, therefore pin 2 of the RS232 connector is routed to the SENSE input of the 2650. Received Data (pin 3) is transmitted from the PC1001, therefore pin 3 of the RS232 is routed to the FLAG output of the 2650.

The signals on pins 5, 6, 8, and 20 are used between data terminals and communications MODEMs. Since the PC1001 does not provide these "handshake" lines they can be simulated by shorting them all together. In this configuration the Data Terminal Ready line drives the other 3 lines to the proper state for enabling the communication channel.

This is not required for all data terminals (not teletypes), but is required for some.

Further information on RS232C specifications can be obtained from the EIA RS-232-C Standard available from the Electronic Industries Association in Washington D.C.

The type of connector commonly used for RS232 compatible data terminals is a 25-pin TRW Cinch type connector of the DB25 series.

TELETYPE CONNECTION

Connection to a teletype may be made at the terminal strip inside of the teletype. The pin numbers and signal names are listed in the table below.

PIN #	DESCRIPTION
6	RECEIVER - (TTY SERIAL IN -)
7	RECEIVER + (TTY SERIAL IN +)
3	TRANSMITTER - (TTY SERIAL OUT -)
4	TRANSMITTER + (TTY SERIAL OUT +)

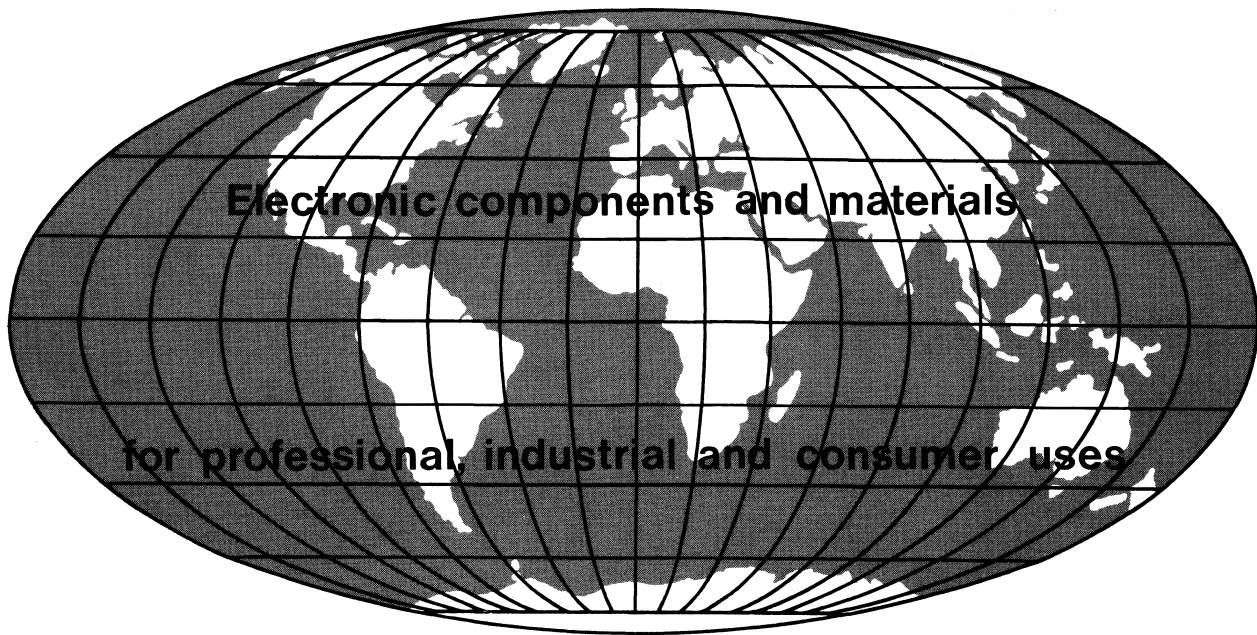
The teletype is a 20 milliamp current loop type of receiver and a contact closure type of transmitter. The PIPBUG debug program on the PC1001 board communicates with the teletype in a full duplex mode, echoing characters as they are received.

EDGE CONNECTOR SIGNAL LIST

PIN #	FUNCTION	PIN #	FUNCTION
1	GND	A	GND
2	GND	B	GND
3	NC*	C	NC
4	$\overline{\text{DBUS0}}$	D	OPD 0
5	$\overline{\text{DBUS1}}$	E	OPD 1
6	$\overline{\text{DBUS2}}$	F	OPD 2
7	$\overline{\text{DBUS3}}$	H	OPD 3
8	$\overline{\text{DBUS4}}$	J	OPD 4
9	$\overline{\text{DBUS5}}$	K	OPD 5
10	$\overline{\text{DBUS6}}$	L	OPD 6
11	$\overline{\text{DBUS7}}$	M	OPD 7
12	EIPD	N	COPD
13	$\overline{\text{D/C}}$	P	TTY SERIAL IN +
14	$\overline{\text{DMA}}$	R	TTY SERIAL IN -
15	E/ $\overline{\text{NE}}$	S	TTY SERIAL OUT +
16	INTACK	T	TTY SERIAL OUT -
17	$\overline{\text{R/W}}$	U	RS232 GROUND
18	WRP	V	RS232 OUTPUT
19	$\overline{\text{RUN/WAIT}}$	W	TTY TAPE READER OUT -
20	OPREQ	X	TTY TAPE READER OUT +
21	$\overline{\text{M/I0}}$	Y	RS232 INPUT
22	$\overline{\text{OPACK}}$	Z	COPC
23	CLOCK	a	OPC 0
24	$\overline{\text{OPEX}}$	b	OPC 1
25	$\overline{\text{RESET}}$	c	OPC 2
26	$\overline{\text{INTREQ}}$	d	OPC 3
27	$\overline{\text{PAUSE}}$	e	OPC 4
28	NC*	f	OPC 5
29	NC*	g	OPC 6
30	NC*	h	OPC 7
31	NC*	j	EIPC
32	NC*	k	IPD 0
33	ABUS 11	m	IPD 1
34	ABUS 13	n	IPD 2
35	ABUS 12	p	IPD 3
36	ABUS 14	r	IPD 4
37	ABUS 9	s	IPD 5
38	ABUS 10	t	IPD 6
39	ABUS 8	u	IPD 7
40	ABUS 7	v	IPC 0
41	ABUS 6	w	IPC 1
42	ABUS 5	x	IPC 2
43	ABUS 3	y	IPC 3
44	ABUS 0	z	IPC 4
45	ABUS 1	$\overline{\text{a}}$	IPC 5
46	ABUS 4	$\overline{\text{b}}$	IPC 6
47	ABUS 2	$\overline{\text{c}}$	IPC 7
48	+15V	$\overline{\text{d}}$	+15V
49	-15V	$\overline{\text{e}}$	-15V
50	+5V	$\overline{\text{f}}$	+5V

*NC = NO CONNECTION

TABLE 4



from the world-wide Philips Group of Companies

EUROPEAN SALES OFFICES

- Austria:** Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.
Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.
Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.
Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.
France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.
Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.
Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.
Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.
Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.
Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.
Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.
Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.
Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.
Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.
Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.
Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoğlu, İSTANBUL.
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

© N.V. Philips' Gloeilampenfabrieken

This information is furnished for guidance, and with no guarantees as to its accuracy or completeness; its publication conveys no licence under any patent or other right, nor does the publisher assume liability for any consequence of its use; specifications and availability of goods mentioned in it are subject to change without notice; it is not to be reproduced in any way, in whole or in part, without the written consent of the publisher.



Electronic
components
and materials

PHILIPS

THE ABC ADAPTABLE
BOARD COMPUTER SP55

AN APPLICATION MEMO

signetics

INTRODUCTION

System development cards are designed to simplify the user's task when doing microprocessor evaluation prototyping. To achieve this goal, the card should be designed with enough flexibility to make it adaptable to individual requirements. It should contain a certain amount of RAM for storage of programs under development. A ROM or PROM or a combination of both should be provided for permanent storage of programs such as debug software. The card should be designed for easy interfacing to current loop or RS-232 terminal devices. Buffers should be provided for address, data and control lines to facilitate expansion of memory and I/O logic. The use of one or more general-purpose ports will aid in I/O interfacing. In summary, the overall design philosophy should be to add nothing to the card that is not a basic necessity. This philosophy maximizes cost effectiveness and minimizes unused design features.

This applications memo describes the various components and applications of the ABC (Adaptable Board Computer) 1500 system development card. Topics covered include:

- ABC1500 memory organization
- Memory and I/O port decoding
- I/O interface
- Bus and control line buffers
- Clocking requirements
- Minimum ABC system configuration
- Addition of 1K of RAM memory
- Step mode operation
- Synchronous and asynchronous operation
- I/O port interface design examples
- Interrupt option
- Kit considerations
- Component identification list
- ABC1500 edge connector signal list

THE ABC1500

The objective of the ABC1500 card is to enable the user to develop 2650-based systems in a configuration that fits his particular needs. The card is designed around the Signetics 2650 8-bit microprocessor. It contains 1K bytes of ROM with the PIPBUG* debug program, 512 bytes of RAM, 2 general-purpose parallel I/O ports, 1 serial I/O port, and buffers for the address, data, and control lines.

Wire jumpers are included for selecting from among several available memory and I/O port configurations, terminal interface schemes, and operating modes. Additional circuitry can be added to the card in the

wire-wrap area provided. Expansion of the card is made possible by feeding all buffered address, data and control lines into a 100-pin edge connector.

An assembly drawing and a logic diagram of the ABC1500 are shown in Figures 1 and 2, respectively.

If the current-loop interface is used, only a single 5 volt supply is necessary to power the entire card. When communicating with RS-232 type terminal devices, a ±12 volt power supply is also required.

The ABC card is sold either as a completely assembled and tested card (2650PC1500) or in kit form (2650KT9500).

ABC1500 MEMORY ORGANIZATION

To simplify memory decoding, a trade-off was made between usable memory space and the complexity of the decoder. By allocating the entire 8K of page zero to the on-card memory and limiting memory expansion to 24K (adequate for the majority of prototyping applications), considerable simplification was realized. Only 2, 16-pin ICs are required to perform both memory selection and I/O port decoding.

Three address lines (A9, A11, and A12) are not used to address the on-card memory. The result is that the on-card ROM and RAM appear to occupy the entire 8K page in an interweaving pattern as shown in Table 1. This prohibits external memory from using any of the page zero memory space, and the

first allowable location for add-on memory is 2000₁₆, or the beginning of page 1. All of pages 1, 2 and 3 are available for memory expansion.

ROM Configuration

The 1K bytes of ROM are implemented with one 2608 NMOS static ROM (IC7) that contains PIPBUG, a firmware aid used to enter and debug user programs. Since the ROM occupies the first 1K bytes of address space, the 2650 will enter PIPBUG when the card is reset. If the debug program is not required, the ROM can be removed from its socket and replaced with a user ROM or with two 82S115 (512 x 8) PROMs, for which board space is provided (IC5 and IC6). However, the ROM and PROMs cannot be used together since they occupy the same address space.

RAM Configuration

The 512 bytes of RAM are implemented with four 2112-2 (256 x 4) NMOS static RAMs (ICs 1, 2, 3, 4). They are located in the memory address space from 400₁₆ to 5FF₁₆, but also appear to occupy other address spaces in page 0 as shown in Table I. Since the second block of memory occupies the "top" of page 0, the on-card RAM may be used to store indirect addresses or subroutines which can be accessed by the ZBRR and ZBSR instructions with negative offsets.

Since PIPBUG resides in the first 1K of memory, an interrupting device cannot use

ADDRESS LINES						DECIMAL ADDRESS	ORGANIZATION	HEX ADDRESS
A14	A13	A12	A11	A10	A9			
						8k	1FFF	1FFF
0	0	X	X	1	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						7k	1BFF	1BFF
0	0	X	X	0	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						6k	17FF	17FF
0	0	X	X	1	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						5k	13FF	13FF
0	0	X	X	0	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						4k	0FFF	0FFF
0	0	X	X	1	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						3k	0BFF	0BFF
0	0	X	X	0	X		SECOND BLOCK RAM	
							FIRST BLOCK RAM	
						2k	07FF	07FF
0	0	X	X	1	X		SECOND BLOCK RAM	06FF
							FIRST BLOCK RAM	05FF
							SECOND BLOCK RAM	04FF
							FIRST BLOCK RAM	03FF
0	0	X	X	0	X		PIPBUG ROM	
						0		0000

NOTES: 1. * = Don't care for ROM and RAM; ** = Don't care for RAM. 2. Each block of RAM = 256 bytes.

Table 1 MEMORY MAP

*PIPBUG, a program debug module, is described in detail in Signetics MOS Microprocessor Applications Memo SS50.

ABC1500 LOGIC DIAGRAM

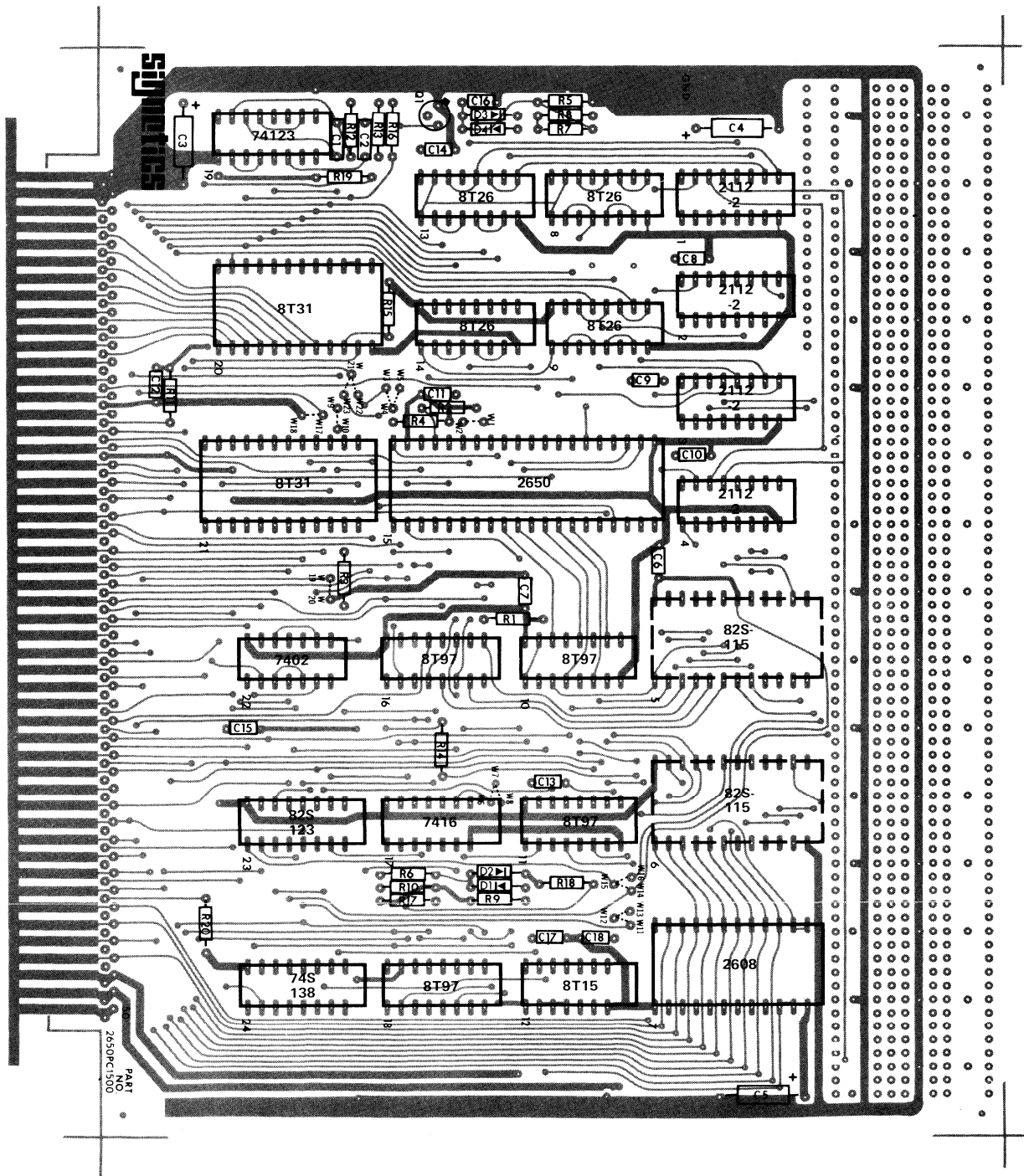


Figure 1

ABC1500 ASSEMBLY DRAWING

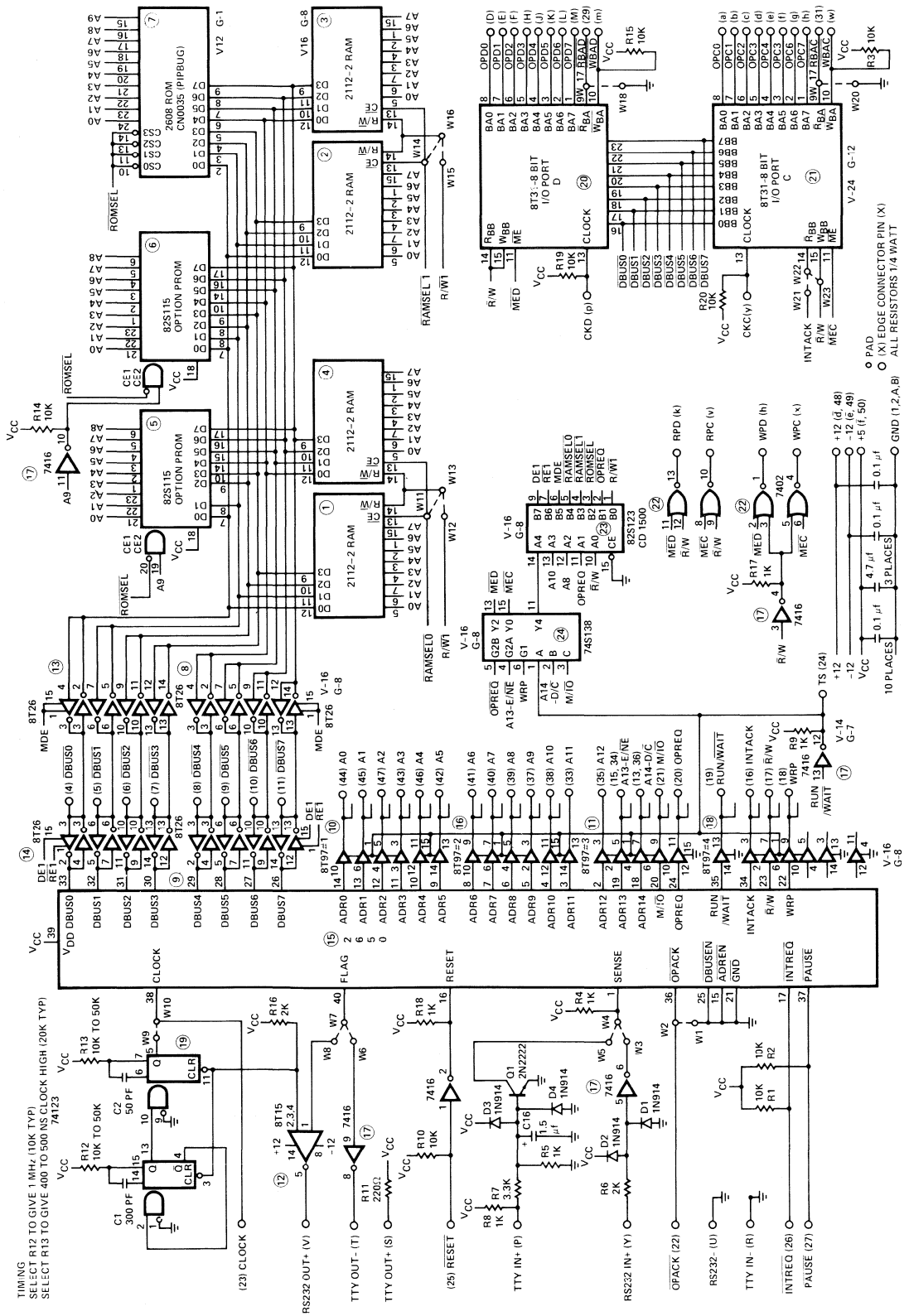


Figure 2

the first 63 memory locations for indirect address or interrupt routine locations. However, since RAM exists at the top of the 8K page, the interrupting device can provide vector addresses in the negative direction from address '0'. A negative vector from address location '0' wraps around to the top of the page.

Optional Memory Configurations

Optional operation using 82S129 PROMs or 82S229 ROMs in place of the 2112-2 RAMs is possible. This modification requires a jumper change for each 256-byte block of memory to exchange the R/W line on the RAM for the pin equivalent chip enable line on the PROM/ROM. The first block of memory (400₁₆ - 4FF₁₆) requires that jumper W₁₂-W₁₃ be replaced with jumper W₁₁-W₁₃. The second block of memory (500₁₆ - 5FF₁₆) requires that jumper W₁₅-W₁₆ be replaced with jumper W₁₄-W₁₆.

Table II is a representative sample of the memory configurations that are possible with the ABC card. Other combinations of RAM/PROM/ROM are possible. When working with PIPBUG, the first block of memory must be RAM, since PIPBUG uses the first 63 bytes of RAM for temporary storage.

DMA Transfers

The ability to transfer data into memory with a DMA transfer has been sacrificed in the interest of card simplicity. DMA transfers with external add-on memories, however, can be performed by stopping the 2650 via the 'PAUSE' line. If the PAUSE input is brought to ground, the 2650 will finish the current instruction, and then the RUN-/WAIT output of the 2650 will go low, causing all external memory address, data, and control lines to be tri-stated, except for OPREQ.

The user can then externally drive all of the memory control lines except OPREQ. This line is not tri-stated, since it is used to disable the decoding PROM (IC23) when the 2650 is in the WAIT state.

ABC MEMORY AND I/O PORT DECODING

Two 16-pin ICs are used to perform memory and I/O port decoding. The first is a 74S138, 3-to-8 line decoder with enable inputs. It performs decoding for port C selection, port D selection, and on-card memory decoding. Table III shows the logical relationship between the 6 input signals and the 3 output signals.

The second decoder is an 82S123, 32 x 8 PROM used as a logic element. Its outputs are programmed functions of the inputs. The DE1 and RE1 lines control the 8T26 driver/receivers between the 2650 and the external data bus. Signal MDE controls the 8T26s between the internal memory bus and the external data bus. The RAMSEL0, RAMSEL1, and ROMSEL outputs are chip selects for the RAM and ROM memories. Signal R/W1 performs read/write control of the on-board RAM memory. Table IV is the truth table for the PROM, while Table V represents the logical relationship between the 5 input signals and the 8 output signals.

I/O INTERFACE

I/O interface for the ABC1500 card consists of 1 serial port and 2 parallel 8-bit ports. The serial port provides a communication path for current loop (20mA) and RS-232 interfaces. The two 8-bit ports may be used for general-purpose I/O interfacing over the C and D buses.

Serial Port

Communication with the terminal device is performed serially using the 'SENSE' and 'FLAG' lines on the 2650. Both current loop and RS-232 transmission modes are possible, each being selected with a pair of wire jumpers, as shown in Table VI. A ±12 volt

MEMORY CONFIGURATION (BYTES)	MEMORY TYPES	ADDRESS RANGE* (HEX)	COMMENTS
1K ROM (PIPBUG) 512 RAM	1-2608 4-2112-2	0-3FF 400-5FF	Standard configuration
1K PROM 512 RAM	2-82S115 4-2112-2	0-3FF 400-5FF	Remove PIPBUG ROM from socket and insert PROMs in holes provided (IC5, 6).
1K ROM (PIPBUG) 256 RAM 256 PROM	1-2608 2-2112-2 2-82S129	0-3FF 400-4FF 500-5FF	PIPBUG requires 63 bytes of RAM storage. Remove jumper W15-W16. Add jumper W14-W16.
1K ROM 512 PROM	1-2608 4-82S129	0-3FF 400-5FF	Remove jumpers W15-16 and W12-W13. Add jumpers W14-W16 and W11-W13.
1K PROM 512 ROM	2-82S115 4-82S229	0-3FF 400-5FF	Remove PIPBUG ROM from socket and insert PROMs in holes provided (IC 5, 6). Remove jumpers W15-W16 and W12-W13. Add jumpers W14-W16 and W11-W13.

*Because of don't care address bits, these memory blocks will appear several times in the first 8K page (see Table I).

Table 2 SAMPLE ABC1500 MEMORY CONFIGURATIONS

OUTPUT	PIN	EQUATION	FUNCTION
MEC	15	$\overline{MEC} = OPREQ \cdot E/\overline{NE} \cdot WRP \cdot \overline{TS} \cdot \overline{D/C} \cdot \overline{M/IO}$	Select non-extended Port C
MED	13	$\overline{MED} = OPREQ \cdot E/\overline{NE} \cdot WRP \cdot \overline{TS} \cdot \overline{D/C} \cdot \overline{M/IO}$	Select non-extended Port D
MEMSEL	11	$\overline{MEMSEL} = OPREQ \cdot \overline{A13} \cdot WRP \cdot \overline{TS} \cdot \overline{A14} \cdot \overline{M/IO}$	Select on card memory

Table 3 ONE-OF-EIGHT DECODER OUTPUT LOGIC EQUATIONS

power supply is required for the RS-232 mode. The 'SENSE' input is driven by a discrete "current loop" receiver (Q1) or an RS-232 compatible inverter input (IC17, pin 5). The 'FLAG' output is connected to the RS-232 compatible 8T15 driver (IC12, pin 1) or the "current loop" driver (IC17, pin 9). Tables VII and VIII show the interconnections between the ABC1500 and current loop or RS-232 compatible terminals.

Parallel Ports

Two non-extended I/O channels are implemented on the ABC1500 with 8T31 8-bit, bidirectional latched ports (IC21-Port C, IC20-Port D). The 2650 transfers data to and from each port using single-byte, non-extended I/O instructions. Three control and 2 status lines for each port permit the establishment of a handshaking routine to

efficiently control data transfers between the user's device and these I/O ports. Since each port inverts the data from one side to the other and the data bus drivers/receivers are also of the inverting type, data on the C and D buses will have the same polarity as the data internal to the 2650.

Port Control Lines

Table IX lists the 3 port control lines and the operation performed by each. These lines are routed to the edge connector for external interface. If no external logic is connected, each port will be in the READ mode (C and D buses reflecting latched data in each port), since the \overline{WBAC} and \overline{WBAD} lines are pulled up to +5 volts with 10K resistors on the card. A low condition on either line will write data from the C or D bus into the appropriate port.

NOTE: Care must be exercised when writing to the ports. An external device will override the 2650 when "WRITE" conflicts occur.

As shown in Table IX, the \overline{RBAC} and \overline{RBAD} lines serve only to put the buses into the TRI-STATE mode. If the third state is not necessary, lines \overline{RBAC} and \overline{RBAD} can be tied to ground to allow read/write control with just 1 line on each port. The ABC1500 card includes provisions for ground connection, with jumper W19-W20 for \overline{RBAC} and jumper W17-W18 for \overline{RBAD} . The assembled card is shipped with these jumpers in place.

Each port has a clock line that controls writing to that port. Both clocks (CKC for port C and CKD for port D) are pulled up to +5 volts on the card with 10K resistors. Therefore, no external connection is required if the ports are to be always enabled. A low on a clock line will inhibit that port from receiving any data from either the 2650 or the external device.

Port Status Lines

During 2650 activity with the ports, the ABC1500 provides 4 output strobes indicating the nature of the operation. These "user-convenience" strobes are described in Table X. Each strobe is generated from a 7402, 2-input NOR gate. Examples of how these strobes may be used in practical applications are included later in this applications memo.

INPUTS					OUTPUTS								
ADDR.	\overline{MEM}	A10	A8	\overline{OPREQ}	$\overline{R/W}$	$\overline{RE1}$	DE1	MDE	\overline{RAM}	\overline{RAM}	\overline{ROM}	\overline{OPREQ}	$\overline{R/W1}$
	A4	A3	A2	A1	A0	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1
1	0	0	0	0	1	1	0	0	1	1	1	1	0
2	0	0	0	1	0	0	0	1	1	1	0	0	1
3	0	0	0	1	1	1	1	0	1	1	1	0	0
4	0	0	1	0	0	1	0	0	1	1	1	1	1
5	0	0	1	0	1	1	0	0	1	1	1	1	0
6	0	0	1	1	0	0	0	1	1	1	0	0	1
7	0	0	1	1	1	1	1	0	1	1	1	0	0
8	0	1	0	0	0	1	0	0	1	1	1	1	1
9	0	1	0	0	1	1	0	0	1	1	1	1	0
10	0	1	0	1	0	0	0	1	0	1	1	0	1
11	0	1	0	1	1	1	1	0	0	1	1	0	0
12	0	1	1	0	0	1	0	0	1	1	1	1	1
13	0	1	1	0	1	1	0	0	1	1	1	1	0
14	0	1	1	1	0	0	0	1	1	0	1	0	1
15	0	1	1	1	1	1	1	0	1	0	1	0	0
16	1	0	0	0	0	1	0	0	1	1	1	1	1
17	1	0	0	0	1	1	0	0	1	1	1	1	1
18	1	0	0	1	0	0	0	0	1	1	1	0	1
19	1	0	0	1	1	1	1	0	1	1	1	0	1
20	1	0	1	0	0	1	0	0	1	1	1	1	1
21	1	0	1	0	1	1	0	0	1	1	1	1	1
22	1	0	1	1	0	0	0	0	1	1	1	0	1
23	1	0	1	1	1	1	1	0	1	1	1	0	1
24	1	1	0	0	0	1	0	0	1	1	1	1	1
25	1	1	0	0	1	1	0	0	1	1	1	1	1
26	1	1	0	1	0	0	0	0	1	1	1	0	1
27	1	1	0	1	1	1	1	0	1	1	1	0	1
28	1	1	1	0	0	1	0	0	1	1	1	1	1
29	1	1	1	0	1	1	0	0	1	1	1	1	1
30	1	1	1	1	0	0	0	0	1	1	1	0	1
31	1	1	1	1	1	1	1	0	1	1	1	0	1

Table 4 CONTROL PROM TRUTH TABLE (82S129)

OUTPUT	PIN	EQUATION	FUNCTION
DE1	9	$DE1 = OPREQ \cdot \overline{R/W}$	Enables 2650 data bus drivers
$\overline{RE1}$	7	$\overline{RE1} = OPREQ \cdot \overline{R/W}$	Enables 2650 data bus receivers
MDE	6	$MDE = OPREQ \cdot MEMSEL \cdot \overline{R/W}$	Select ABC1500 RAM, first block
$\overline{RAMSEL0}$	5	$\overline{RAMSEL0} = OPREQ \cdot MEMSEL \cdot A10 \cdot \overline{A8}$	Causes ABC1500 memory to drive data bus
$\overline{RAMSEL1}$	4	$\overline{RAMSEL1} = OPREQ \cdot MEMSEL \cdot A10 \cdot A8$	Select ABC1500 RAM, second block
\overline{ROMSEL}	3	$\overline{ROMSEL} = OPREQ \cdot MEMSEL \cdot A10$	Select ABC1500 ROM
\overline{OPREQ}	2	$\overline{OPREQ} = OPREQ$	Invert OPREQ
R/W1	1	$R/W1 = MEMSEL \cdot \overline{R/W}$	Read/write control for on-card RAM

Table 5 PROGRAMMED OUTPUT LOGIC EQUATIONS FOR THE 82S123 PROM

MODE OF TRANSMISSION	JUMPERS	POWER SUPPLY
20 Milliamp Current Loop	W6 - W7 W4 - W5	No Additional Supply Required
RS-232 Compatible	W7 - W8 W3 - W4	±12 Volt Supply

Table 6 SERIAL TRANSMISSION MODES

RS-232 STANDARD PIN NUMBER	ABC1500 PIN NUMBER	DESCRIPTION
1	N.C.	Protective Ground
2	Y	Transmitted Data (RS-232 IN+)
3	V	Received Data (RS-232 OUT+)
5	N.C.	Clear to send
6	N.C.	Data Ready
7	U	Signal Ground (RS-232 -)
8	N.C.	Received Line Signal Detector
20	N.C.	Data Terminal Ready

Table 7 RS-232C STANDARD CONNECTION

NOTE: N.C. = No Connection

The signals on pins 5, 6, 8, and 20 are used between data terminals and communication modems. Since the ABC1500 does not provide these "handshaking" lines, they should be tied together on the connector. In this way, the "Data Terminal Ready" line drives the other three lines to the proper state. Not all terminals, however, require this connection.

TELETYPE PIN NUMBER	ABC1500 PIN NUMBER	DESCRIPTION
6	T	Receiver - (TTY Serial OUT-)
7	S	Receiver + (TTY Serial OUT+)
3	R	Transmitter - (TTY Serial IN-)
4	P	Transmitter + (TTY Serial IN+)

Table 8 CURRENT LOOP CONNECTION

NOTE:

The teletype is normally a 20mA current loop type of receiver and a contact closure type of transmitter. The PIPBUG Debug program communicates with the teletype in a full-duplex mode, echoing characters as they are received.

PORT C CONTROL LINE TRUTH TABLE			
WBAC	RBAC	CKC	DESCRIPTION
1	0	X	External device reading port C
0	0	1	External device writing to port C
1	1	X	Tri-state C bus
X	X	0	Inhibit writing to port C from either external device or 2650

PORT D CONTROL LINE TRUTH TABLE			
WBAD	RBAD	CKD	DESCRIPTION
1	0	X	External device reading port D
0	0	1	External device writing to port D
1	1	X	Tri-state D bus
X	X	0	Inhibit writing to port D from either external device or 2650

Table 9 PORT CONTROL LINES

ABC GENERATED STATUS STROBE	DESCRIPTION
WPC	Positive true pulse, high for the duration of WRP, indicating that the 2650 is placing data into port C.
WPD	Positive true pulse, high for the duration of WRP, indicating that the 2650 is placing data into port D.
RPC	Positive true pulse, high for the duration of OPREQ, indicating that the 2650 is reading port C.
RPD	Positive true pulse, high for the duration of OPREQ, indicating that the 2650 is reading port D.

Table 10 ABC GENERATED CONVENIENCE STROBES

BUS AND CONTROL LINE BUFFERS

The 2650 data bus is buffered with two 8T26 tri-state inverting driver/receivers with a 40mA current sink capability (IC9, 14). Logic on the card consumes approximately 1mA, leaving a net external drive capability of 39mA. When the 2650 is not transferring data over the bus, these buffers are in the tri-state mode. The output of the buffers (DBUS0-DBUS7) is routed to the edge connector (pins 4-11).

The on-board memory bus is buffered from the external data bus with two 8T26s (IC8, 13). These buffers are never in the tri-state mode. When not actively transferring data, these devices are reading the external data bus to the internal memory bus. Double buffering between the memory and 2650 allows data polarity to be preserved.

The 2650 address bus and control lines are buffered with four 8T97 hex tri-state buffers (IC10, 11, 16, 18). The state of the TS control line (IC17, pin 12) determines whether the drivers are in the tri-state mode or actively driving the external lines. When the 2650 is in the RUN mode, the TS line is low, enabling the 8T97s to be in the active state. When the 2650 is in the WAIT mode, the TS

line is high, and the address bus drivers are in the high-impedance state. External control of the TS line can change the address bus drivers to the active mode when the 2650 is in the WAIT state, but cannot force the drivers to the high-impedance state when the 2650 is "running."

The external drive capability of the address bus is essentially the same as the drive capability of the 8T97 (48mA sink capability). The control lines will be loaded slightly by logic on the card. The maximum load is on the A13•E/NE and A14•D/C lines. Card logic will consume 2mA of the 48mA capability.

ABC1500 CLOCKING REQUIREMENTS

The clock on the ABC1500 card is implemented with a 74123 (IC19) dual monostable multivibrator. One half of the 74123 is connected in an astable mode to determine the frequency of the clock. The other half is connected as a one-shot to set the pulse width.

When running under PIPBUG, the 2650 requires a 1MHz clock for serial communication with the TTY. The PIPBUG program performs all formatting for the 110 baud

interface and uses the clock as a timing base. The stability requirements of the clock are not critical, and the one-shot configuration is adequate. For KT9500 assembly, it may be necessary to select frequency resistor R12 to insure the 1MHz operation and the sampling of each bit at its approximate midpoint. R12 is typically 7.5K. In most cases pulse width resistor R13 will be fixed at 20K to obtain a clock high time between 400 and 500ns.

An external clock may be used in place of the one-shot configuration. When using an external clock, jumper W9-W10 must be removed, and the external clock can then be applied to pin 23 on the edge connector.

A MINIMUM ABC1500 SYSTEM CONFIGURATION

In Figure 3, the ABC1500 card is interfaced with 3 other components to configure a basic prototyping system.

The reset switch is used to reset the 2650's Instruction Address Register (IAR) to zero and to enter the PIPBUG program. This negative true input is inverted by IC17 to obtain the positive true polarity required by the 2650.

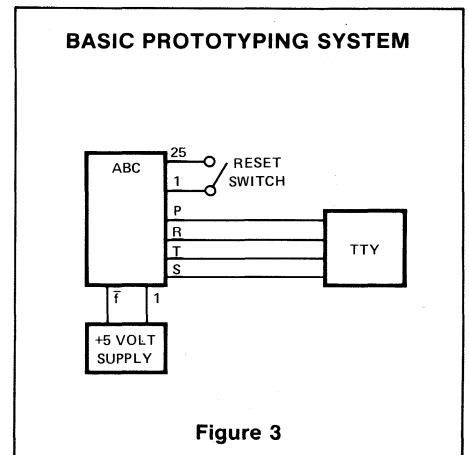


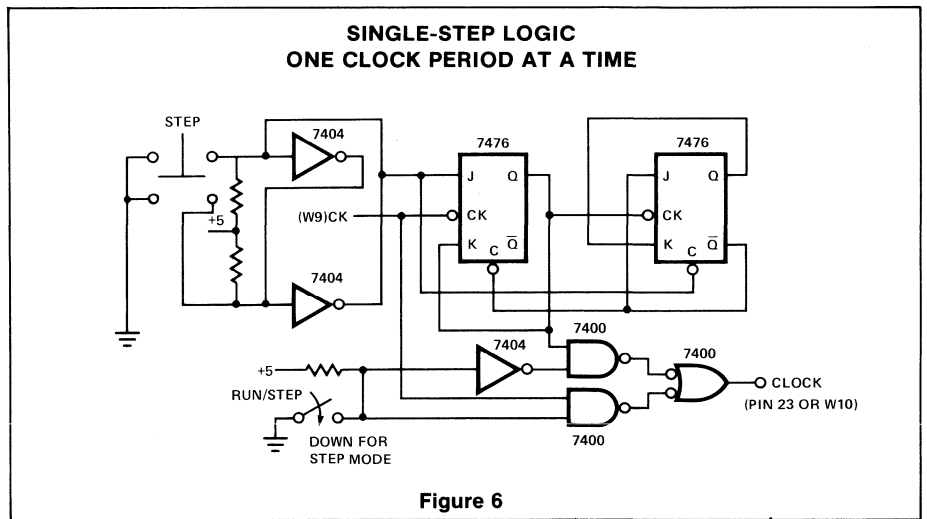
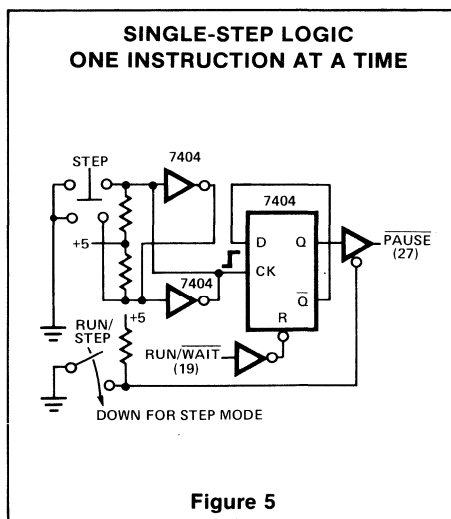
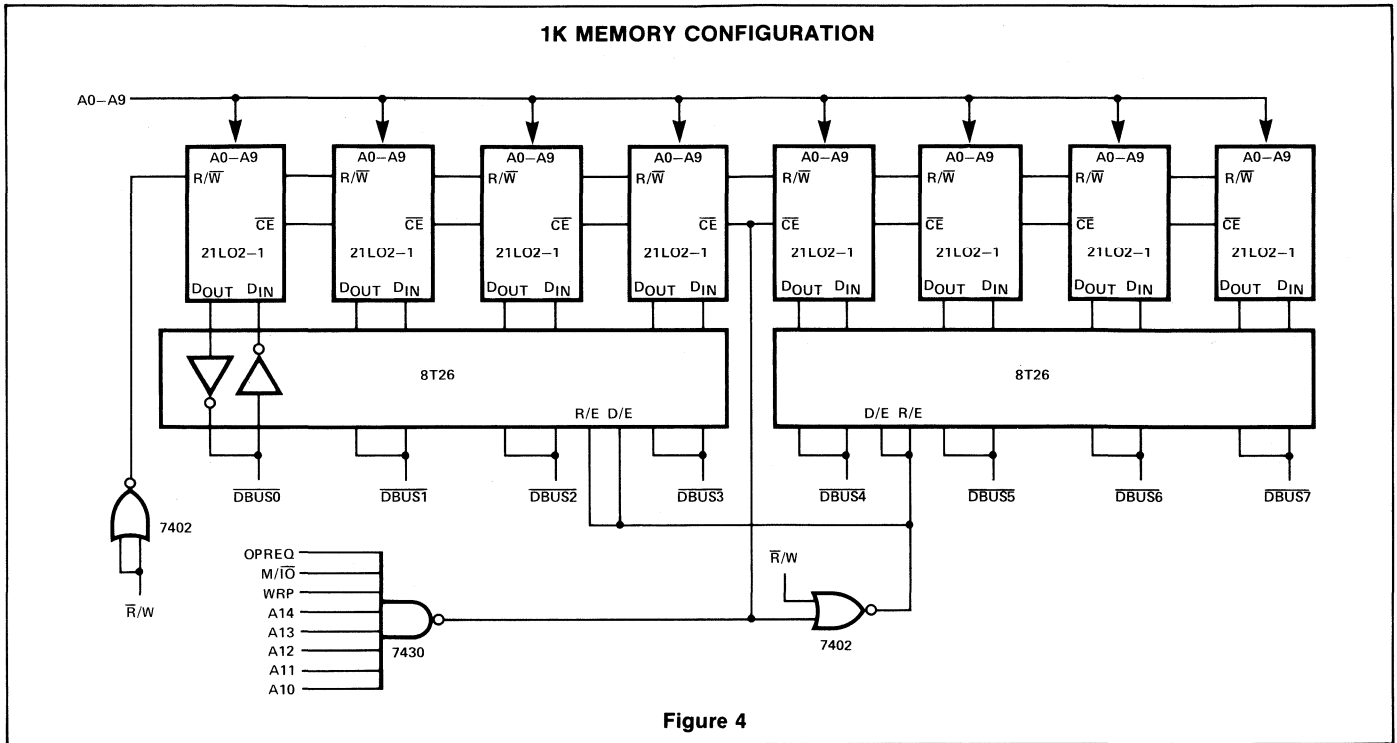
Figure 3

ADDITION OF 1K OF RAM MEMORY TO THE ABC1500 CARD

It is possible to expand the memory of the ABC1500 card by using the wire-wrap area. In the example shown in Figure 4, 12 ICs are used to add 1K of RAM memory.

The memory occupies the last 1K section of page 3 and uses a single N7430 gate to decode the appropriate signals. These signals can be obtained from wire wrap pins inserted into the appropriate holes on the card.

The 8T26 buffers are used to multiplex the single input and output from each memory (1K x 1) onto the external data bus. When



not communicating with the memory, the buffers are reading the external data bus.

STEP MODE OPERATION

The ability to cycle through a program one step at a time is very useful when checking out software. The 2650 microprocessor is ideally suited for this type of operation because of its static design. An example of the logic necessary to put the ABC1500 card into the step mode for single-instruction execution is shown in Figure 5.

To enter the step mode, the RUN/STEP switch is depressed. This enables the tri-state driver tied to the pause input on the card (pin 27), and immediately causes the PAUSE line of the 2650 to go true or low, since the Q output of the D flip-flop is low for the RUN mode. De-bounce logic is not necessary, since the 2650 will eventually recognize the low condition on the PAUSE line after executing any number of RUN/WAIT cycles.

When the 2650 enters the WAIT state, the RUN/WAIT line (pin 19) goes low, allowing

the flip-flop to be clocked to the SET state (Q goes high) when the momentary contact STEP switch is depressed. With the Q output of the flip-flop high, the PAUSE line will go high taking the 2650 out of the WAIT state. When the 2650 enters the RUN mode, the flip-flop will be reset, and the PAUSE line will again go true, forcing the 2650 into the WAIT state after completing one instruction. This procedure is repeated for each depression of the switch.

It is also possible to step through a program 1 clock period at a time. This procedure is

useful for observing the status of the bus and control lines during each instruction cycle. In this case, instead of pausing the 2650, the clock is controlled with the logic shown in Figure 6.

To work in this mode, the clock jumper W9-W10 is removed and a wire is connected from W9 to the clock input on the first J-K flip-flop in Figure 6, and to one of the inputs on the 2-input NAND gate. When in the RUN mode, the RUN/STEP switch is up, enabling this NAND gate to control the ORing NAND gate. The ORing NAND gate is fed to the clock pin on the ABC card or to W10. This completes a path between the output of the one-shot on the card and the clock input on the 2650.

To enter the step mode, the RUN/STEP switch is depressed. This blocks the clock between W9 and W10. The JK configuration then synchronizes the asynchronous step input with the clock to produce a pulse 1 clock period wide for each depression of the momentary contact STEP switch. This allows 1 clock pulse to be provided to the 2650 for executing instructions 1 clock period at a time.

One approach that can be used for displaying the data and address bus during the single clock period mode is seen in Figure 7. Here the address bus is displayed with an LED and current limiting resistor added to each line. External latches are not required since the 2650 holds the current address state when the clock is low.

To display the data bus will require that the bus be latched, since the bus will be tri-stated when OPREQ is low. One of the ports (Port D in Figure 7) can be used for data storage if that port is not needed by the software. Here the external data bus (DBUS0-DBUS7) is connected to the port bus with LEDs and current limiting resistors on each line. The port READ/WRITE line is controlled by the OPREQ line through an external inverter. When OPREQ is high, the port is being written into from the data bus. When OPREQ is low, the data is latched in the port which is now in the READ mode.

**SYNCHRONOUS/ASYNCHRO-
NOUS OPERATION**

The operation acknowledge (OPACK) input to the 2650 indicates completion of an external operation. This allows for asynchronous control of external devices. The assembled card is configured to work synchronously, with OPACK grounded by jumper W2-W1. This requires input data to be returned to the processor in 850ns or less at a cycle time of 2.4µs. If this timing constraint is too severe, asynchronous operation can be en-

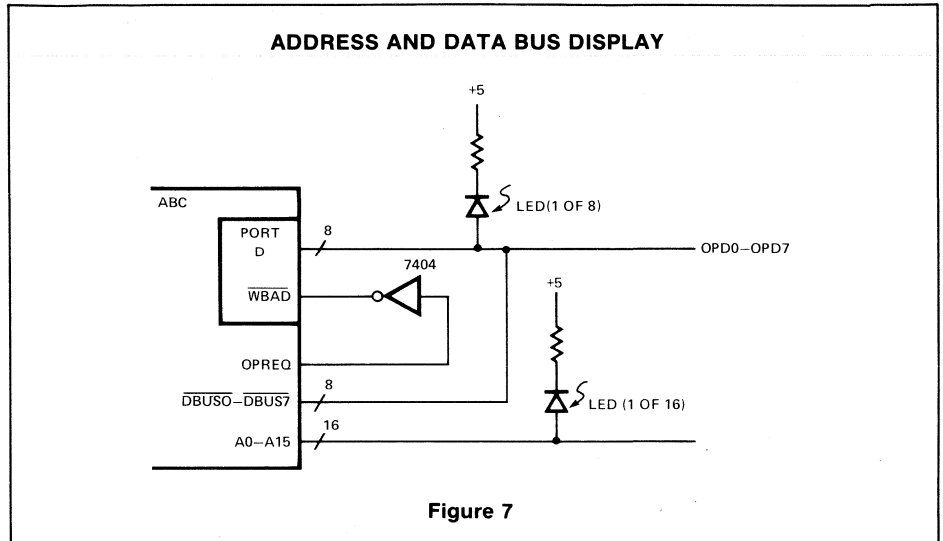


Figure 7

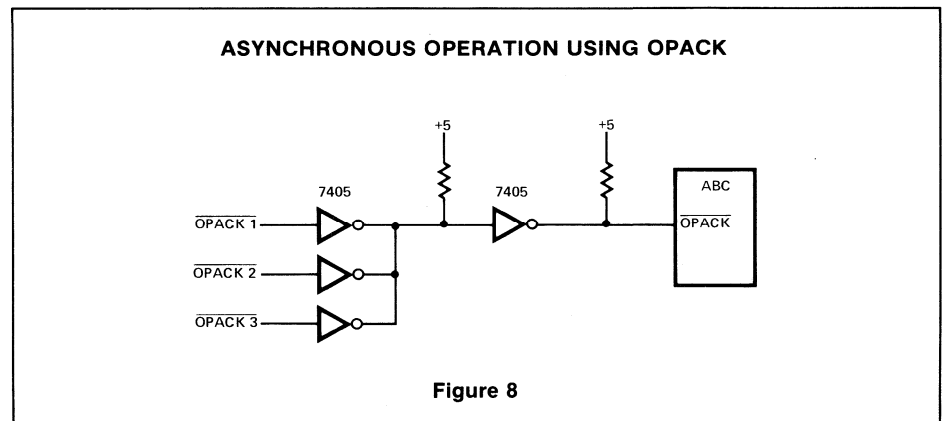


Figure 8

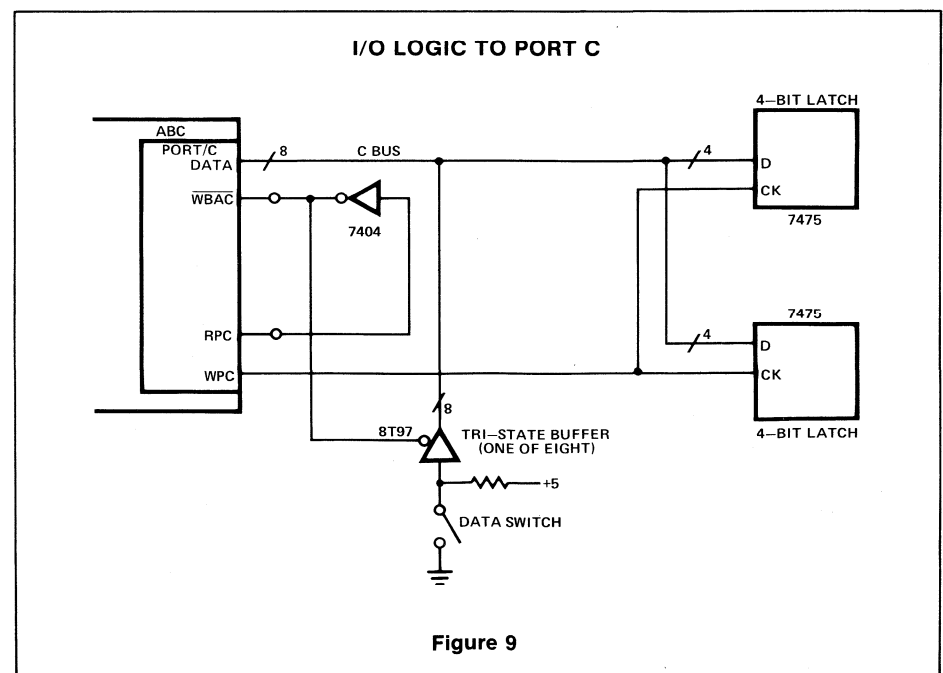


Figure 9

abled by removing the jumper and driving the $\overline{\text{OPACK}}$ line (pin 22).

Figure 8 is a possible configuration for connecting 3 slow devices to the ABC1500 card.

This scheme holds $\overline{\text{OPACK}}$ low (true) until a slow external device is selected, at which time the device drives its respective $\overline{\text{OPACK}}$ line high for the required time. When the device is finished with the operation, it lowers $\overline{\text{OPACK}}$ until it is selected again. When transfers to on-card memory or ports are executed, the $\overline{\text{OPACK}}$ line is held low (true) for synchronous data transfers.

I/O PORT INTERFACE DESIGN EXAMPLES

"Handshaking" signals are provided to simplify communication between the ports and the user's device. Several examples are presented to illustrate possible interface techniques for connecting the 2 ports to external devices.

Example 1—Port C Input/Output Configuration

In this example, port C is accepting data from 8 switches and presenting data to two 4-bit latches. The 2 "handshaking" signals, RPC and WPC, are used in the configuration shown in Figure 9.

The 8 switches are tied to the C bus through tri-state buffers. Input write control line $\overline{\text{WBAC}}$ is controlled by an inverter with its input tied to RPC. When the 2650 performs a

read from port C (REDC), line RPC goes true forcing $\overline{\text{WBAC}}$ low and allowing port C to store data from the C bus. The output of the inverter also controls the tri-state enable of the buffers, turning on the buffers when RPC goes true.

When writing to port C from the 2650, WPC will go true, clocking the data on the C bus into the two 4-bit latches.

Example 2—Synchronizing Data Entry From 2 External Devices

When inputting data from 2 external devices, an interleaving transfer scheme can prevent synchronization conflicts between the devices and the 2650. The configuration is shown in Figure 10.

External device 1 places data onto the C bus and clocks it into port C when the 2650 is reading port D. Likewise, external device 2 places data onto the D bus and clocks it into port D when the 2650 is reading port C, thus preventing conflicts between 2650 activity and loading of the ports from the external devices. Note that alternate read C and read D cycles are required to read the proper data, and that the first read cycle executed will not have valid data associated with it.

Example 3—Synchronizing Data Transfer Between the 2650 and an External Device

The technique illustrated in Figure 11 may be used when transferring data asynchronously between the 2650 and an external device.

In this example, a D latch is used to synchronize data transfers from the 2650 to an external device. When the 2650 loads port C, handshake signal WPC goes true, clocking the D latch to the SET state. The Q output of the latch is tied to the 'SENSE' input (pin Y with jumper W3-W4 in). The 2650 can be programmed to monitor the 'SENSE' line. For the 'SENSE' line HIGH, the program will loop in a WAIT state. When the device has accepted the data, it will reset the latch and force the 'SENSE' line to zero. The 2650 can then place new data in the port.

INTERRUPT OPTION

When responding to an interrupt, the 2650 obtains the interrupt vector by reading the data lines when INTACK is issued. The state of the control lines is such that a read of port C would also be performed (ADR13•E/ $\overline{\text{NE}}$ and ADR14•D/ $\overline{\text{C}}$ are both low). To prevent a conflict between the interrupting device and port C on the card, the INTACK signal is fed to port C to disable the port during interrupts. For certain applications, however, it may be desirable to use port C to input the vector address. This optional operation may be obtained by replacing the W21-W22 jumper with a jumper between W22-W23.

KIT CONSTRUCTION

Kit construction is straightforward requiring only wire, wire cutters, and a soldering iron. Each component has a number which is stamped on the PC card in white. The component number also identifies the location of pin 1 for an IC. The component identification list identifies each number

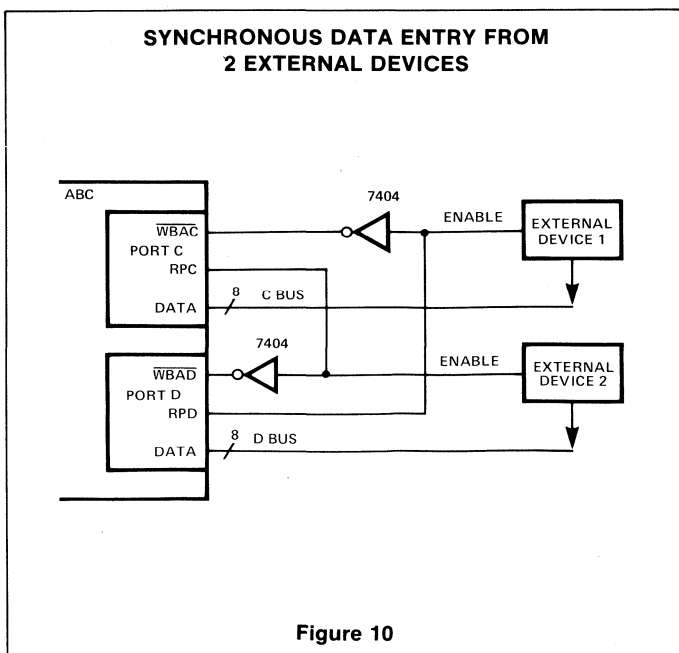


Figure 10

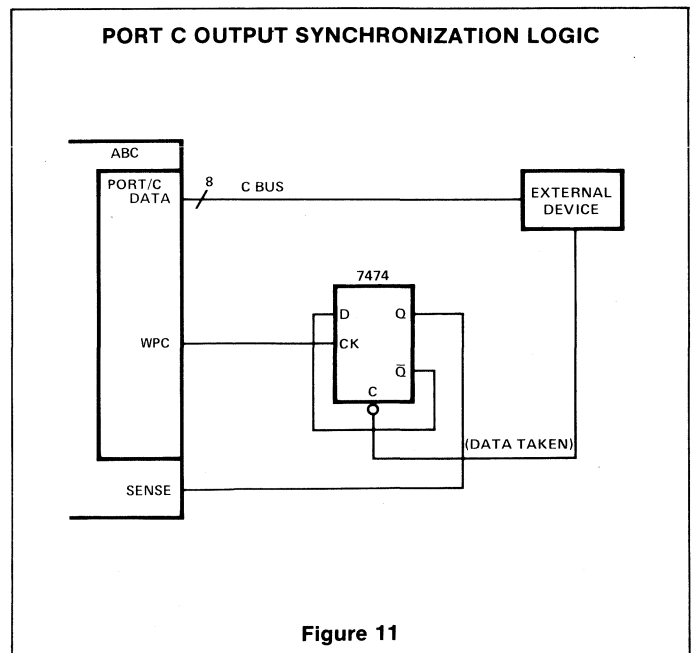


Figure 11

with the appropriate component. Sockets are provided for the 2650 and for the 2608 PIPBUG ROM. If the user expects to use the RAM/PROM/PROM option (see Section 3), he may want to insert sockets in the RAM

holes. Reference should be made to Section 7 for resistor values for the one-shot clock configuration. The kit is shipped with values of 7.5K for R12 and 20K for R13, but it may be necessary to increase or decrease these

values to obtain 1MHz operation. Also, if it is desirable to change the relative position of the RAM and ROM in page zero, the 82S129 control PROM can be re-programmed at the user's discretion.

ABC 1500 EDGE CONNECTOR SIGNAL LIST

PIN #	FUNCTION	PIN #	FUNCTION
1	GND	A	GND
2	GND	B	GND
3	NC*	C	NC*
4	<u>DBUS0</u>	D	OPD 0
5	<u>DBUS1</u>	E	OPD 1
6	<u>DBUS2</u>	F	OPD 2
7	<u>DBUS3</u>	H	OPD 3
8	<u>DBUS4</u>	J	OPD 4
9	<u>DBUS5</u>	K	OPD 5
10	<u>DBUS6</u>	L	OPD 6
11	<u>DBUS7</u>	M	OPD 7
12	NC*	N	NC*
13	A14—D/ \bar{C}	P	TTY SERIAL IN +
14	NC*	R	TTY SERIAL IN -
15	A13—E/ \bar{N}	S	TTY SERIAL OUT +
16	INTACK	T	TTY SERIAL OUT -
17	R/W	U	RS232 GROUND
18	WRP	V	RS232 OUTPUT
19	<u>RUN/WAIT</u>	W	NC*
20	OPREQ	X	NC*
21	M/IO	Y	RS232 INPUT
22	<u>OPACK</u>	Z	NC*
23	CLOCK	a	OPC 0
24	TS	b	OPC 1
25	<u>RESET</u>	c	OPC 2
26	<u>INTREQ</u>	d	OPC 3
27	<u>PAUSE</u>	e	OPC 4
28	NC*	f	OPC 5
29	RBAD	g	OPC 6
30	NC*	h	OPC 7
31	<u>RBAC</u>	j	NC*
32	NC*	k	RPD
33	All	m	<u>WBAD</u>
34	A13—E/ \bar{N}	n	WPD
35	A12	p	CKD
36	A14—D/ \bar{C}	r	NC*
37	A9	s	NC*
38	A10	t	NC*
39	A8	u	NC*
40	A7	v	<u>RPC</u>
41	A6	w	<u>WBAC</u>
42	A5	x	WPC
43	A3	y	CKC
44	A0	z	NC*
45	A1	\bar{a}	NC*
46	A4	\bar{b}	NC*
47	A2	\bar{c}	NC*
48	+12V	\bar{d}	+12V
49	-12V	\bar{e}	-12V
50	+5V	\bar{f}	+5V

*NC = No Connection

ABC 1500 COMPONENT IDENTIFICATION LIST

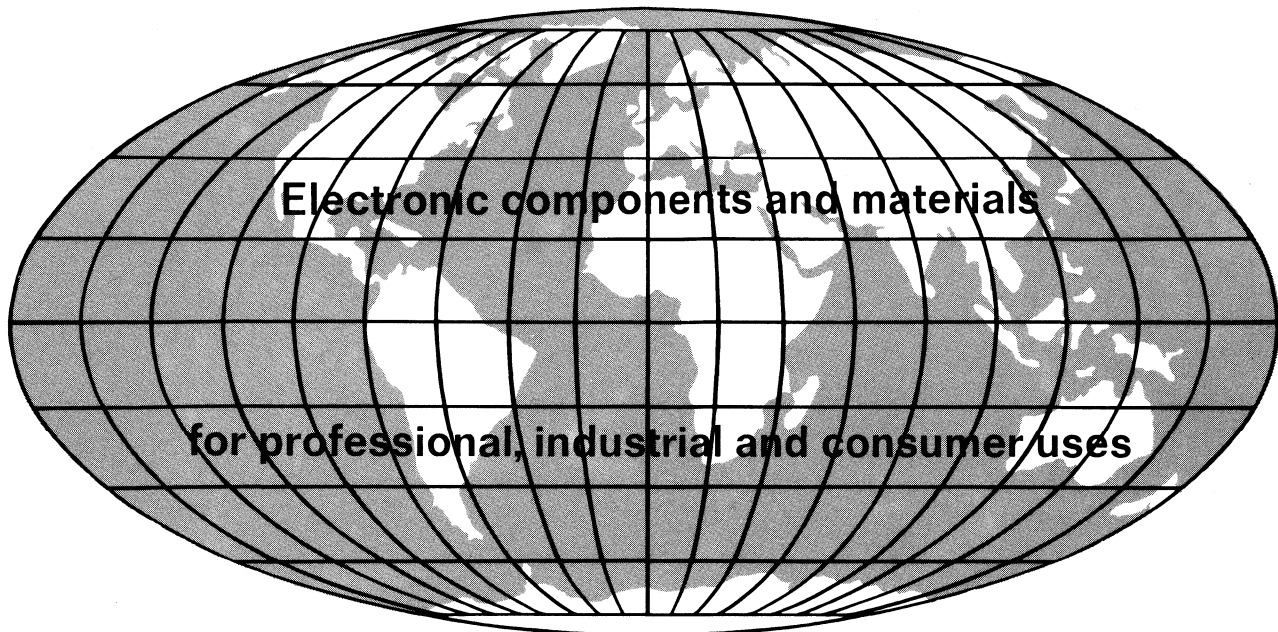
COMPONENT*	DESCRIPTION**
R1, R2, R3	10K Resistor
R4, R5	1K Resistor
R6	2K Resistor
R7	3.3K Resistor
R8, R9	1K Resistor
R10	10K Resistor
R11	220-ohm Resistor
R12	7.5K (typical) Resistor
R13	20K (typical) Resistor
R14	1K Resistor
R15	10K Resistor
R16	2K Resistor
R17, R18	1K Resistor
R19, R20	10K Resistor
C1	300PF Capacitor
C2	50PF Capacitor
C3, C4, C5	4.7µf Capacitor, Tan. 50 DC
C6-C15, C17, C18	0.1µf Capacitor, Ceramic
C16	1.5µf Capacitor, Tan. 20 DC
D1, D2, D3, D4	1N914 Diode
Q1	2N2222 Transistor
1,2,3,4	2112-2 RAM
5,6	82S115 PROM (optional)
7	2608 ROM (socket)
8,9	8T26 Tri-State Driver/Receiver
10,11	8T97 Tri-State Driver
12	8T15 RS232 Driver
13,14	8T26
15	2650 Microprocessor (socket)
16	8T97
17	N7416 Hex Inverter Buffer
18	8T97
19	N74123 Monostable Multivibrator
20,21	8T31 I/O Port
22	N7402 Quad 2-Input NOR
23	82S123 PROM
24	N74S138 3- to 8-line Decoder

* All IC component numbers are located on card at pin 1 of IC.

** All resistors ¼ watt.

Signetics 2650 Microprocessor application memos currently available:

- AS50 Serial Input/Output
- AS51 Bit and Byte Testing Procedures
- AS52 General Delay Routines
- AS53 Binary Arithmetic Routines
- AS54 Conversion Routines
- SP50 2650 Evaluation Printed Circuit Board Level System (PC1001)
- SP51 2650 Demo Systems
- SP52 Support Software for use with NCSS Timesharing System
- SP53 Simulator, Version 1.2
- SP54 Support Software for use with the General Electric Mark III Timesharing System
- SP55 The ABC1500 Adaptable Board Computer
- SS50 PIPBUG
- SS51 Absolute Object Format (Revision 1)
- MP51 2650 Initialization
- MP52 Low Cost Clock Generator Circuits
- MP53 Address and Data Bus Interfacing Techniques
- MP54 2650 Input/Output Structures and Interfaces



from the world-wide Philips Group of Companies

- Argentina:** FAPESA I.y.C., Av. Crovara 2550, Tablada, Prov. de BUENOS AIRES, Tel. 652-7438/7478.
- Australia:** PHILIPS INDUSTRIES HOLDINGS LTD., Elcoma Division, 67 Mars Road, LANE COVE, 2066, N.S.W., Tel. 42 1261.
- Austria:** ÖSTERREICHISCHE PHILIPS BAUELEMENTE Industrie G.m.b.H., Triester Str. 64, A-1101 WIEN, Tel. 62 91 11.
- Belgium:** M.B.L.E., 80, rue des Deux Gares, B-1070 BRUXELLES, Tel 523 00 00.
- Brazil:** IBRAPE, Caixa Postal 7383, Av. Paulista 2073-S/Loja, SAO PAULO, SP, Tel. 287-7144.
- Canada:** PHILIPS ELECTRONICS LTD., Electron Devices Div., 601 Milner Ave., SCARBOROUGH, Ontario, M1B 1M8, Tel. 292-5161.
- Chile:** PHILIPS CHILENA S.A., Av. Santa Maria 0760, SANTIAGO, Tel. 39-40 01.
- Colombia:** SADAPE S.A., P.O. Box 9805, Calle 13, No. 51 + 39, BOGOTA D.E. 1., Tel. 600 600.
- Denmark:** MINIWATT A/S, Emdrupvej 115A, DK-2400 KØBENHAVN NV., Tel. (01) 69 16 22.
- Finland:** OY PHILIPS AB, Elcoma Division, Kaivokatu 8, SF-00100 HELSINKI 10, Tel. 1 72 71.
- France:** R.T.C. LA RADIOTECHNIQUE-COMPELEC, 130 Avenue Ledru Rollin, F-75540 PARIS 11, Tel. 355-44-99.
- Germany:** VALVO, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, D-2 HAMBURG 1, Tel. (040) 3296-1.
- Greece:** PHILIPS S.A. HELLENIQUE, Elcoma Division, 52, Av. Syngrou, ATHENS, Tel. 915 311.
- Hong Kong:** PHILIPS HONG KONG LTD., Comp. Dept., Philips Ind. Bldg., Kung Yip St., K.C.T.L. 289, KWAI CHUNG, N.T. Tel. 12-24 51 21.
- India:** PHILIPS INDIA LTD., Elcoma Div., Band Box House, 254-D, Dr. Annie Besant Rd., Prabhadevi, BOMBAY-25-DD, Tel. 457 311-5.
- Indonesia:** P.T. PHILIPS-RALIN ELECTRONICS, Elcoma Division, 'Timah' Building, Jl. Jen. Gatot Subroto, JAKARTA, Tel. 44 163.
- Ireland:** PHILIPS ELECTRICAL (IRELAND) LTD., Newstead, Clonskeagh, DUBLIN 14, Tel. 69 33 55.
- Italy:** PHILIPS S.P.A., Sezione Elcoma, Piazza IV Novembre 3, I-20124 MILANO, Tel. 2-6994.
- Japan:** NIHON PHILIPS CORP., Shuwa Shinagawa Bldg., 26-33 Takanawa 3-chome, Minato-ku, TOKYO (108), Tel. 448-5611.
(IC Products) SIGNETICS JAPAN, LTD., TOKYO, Tel. (03) 230-1521.
- Korea:** PHILIPS ELECTRONICS (KOREA) LTD., Philips House, 260-199 Itaewon-dong, Yongsan-ku, C.P.O. Box 3680, SEOUL, Tel. 44-4202.
- Mexico:** ELECTRONICA S.A. de C.V., Varsovia No. 36, MEXICO 6, D.F., Tel. 5-33-11-80.
- Netherlands:** PHILIPS NEDERLAND B.V., Afd. Elonco, Boschdijk 525, NL-4510 EINDHOVEN, Tel. (040) 79 33 33.
- New Zealand:** Philips Electrical Ind. Ltd., Elcoma Division, 2 Wagener Place, St. Lukes, AUCKLAND, Tel. 867 119.
- Norway:** ELECTRONICA A/S., Vitaminveien 11, P.O. Box 29, Grefsen, OSLO 4, Tel. (02) 15 05 90.
- Peru:** CADESA, Jr. Ilo, No. 216, Apartado 10132, LIMA, Tel. 27 73 17.
- Philippines:** ELDAC, Philips Industrial Dev. Inc., 2246 Pasong Tamo, MAKATI-RIZAL, Tel. 86-89-51 to 59.
- Portugal:** PHILIPS PORTUGESA S.A.R.L., Av. Eng. Duharte Pacheco 6, LISBOA 1, Tel. 68 31 21.
- Singapore:** PHILIPS SINGAPORE PTE LTD., Elcoma Div., POB 340, Toa Payoh CPO, Lorong 1, Toa Payoh, SINGAPORE 12, Tel. 53 88 11.
- South Africa:** EDAC (Pty.) Ltd., South Park Lane, New Doornfontein, JOHANNESBURG 2001, Tel. 24/6701.
- Spain:** COPRESA S.A., Balmes 22, BARCELONA 7, Tel. 301 63 12.
- Sweden:** A.B. ELCOMA, Lidingövägen 50, S-10 250 STOCKHOLM 27, Tel. 08/67 97 80.
- Switzerland:** PHILIPS A.G., Elcoma Dept., Edenstrasse 20, CH-8027 ZÜRICH, Tel. 01/44 22 11.
- Taiwan:** PHILIPS TAIWAN LTD., 3rd Fl., San Min Building, 57-1, Chung Shan N. Rd, Section 2, P.O. Box 22978, TAIPEI, Tel. 5513101-5.
- Turkey:** TÜRK PHILIPS TICARET A.S., EMET Department, Inonu Cad. No. 78-80, ISTANBUL, Tel. 43 59 10.
- United Kingdom:** MULLARD LTD., Mullard House, Torrington Place, LONDON WC1E 7HD, Tel. 01-580 6633.
- United States:** (Active devices & Materials) AMPEREX SALES CORP., 230, Duffy Avenue, HICKSVILLE, N.Y. 11802, Tel. (516) 931-6200.
(Passive devices) MEPCO/ELECTRA INC., Columbia Rd., MORRISTOWN, N.J. 07960, Tel. (201) 539-2000.
(IC Products) SIGNETICS CORPORATION, 811 East Arques Avenue, SUNNYVALE, California 94086, Tel. (408) 739-7700.
- Uruguay:** LUZILEC I.RON S.A., Rondeau 1507, piso 5, MONTEVIDEO, Tel. 9 43 21.
- Venezuela:** IND. VENEZOLANAS PHILIPS S.A., Elcoma Dept., A. Ppal de los Ruices, Edif. Centro Colgate, Apdo 1167, CARACAS, Tel. 36 05 11.

2650 MICROPROCESSOR APPLICATIONS MEMO

INTRODUCTION

The PIPBUG program is provided as part of the 2650 PC1001 so that the user has immediately available to him the tools necessary to run programs on the 2650 microprocessor. Features include support of a user terminal, papertape load and dump, memory examine and alter, and breakpoints. The 2650 PC1001 card itself is described in detail in applications note SP 50.

DESCRIPTION

The PIPBUG program is started by pressing the reset button on the card. It outputs the user prompt character of '*'. A command is then entered, starting with an alpha character indicating the operation wanted, followed by any required parameters separated by spaces, and all terminated by a carriage return. The parameters must be given as hexadecimal numbers. Leading zeros are unnecessary. For example, '008F' and '8F' are the same address. The error message for an illegal command or parameter is '?', after which the user can enter a new command line. The delete key can be used to delete the previous character.

The program fits in the first 1K bytes of memory in the PROM. Also, the 63 bytes of RAM from location 1024 to 1087 are required for buffers and temporary storage. Locations 0 to 63 are part of the interrupt vector. To fit within 1K bytes the program uses subroutines with a maximum nested depth of three.

In the explanations of the commands CR means the carriage return key and LF means the line feed key. The symbol $\text{\textcircled{b}}$ means there must be at least one space.

COMMANDS

- I. Alter Memory Aaaaa CR
 Action: Outputs $\text{aaaa}\text{\textcircled{b}}\text{cc}$ where 'aaaa' is a memory location and 'cc' is its content. User can respond with:
- 1) CR which ends the command
 - 2) LF which will display the next memory location
 - 3) nn CR which will replace 'cc' by 'nn' at location 'aaaa' and end the command
 - 4) nn LF which will replace 'cc' by 'nn' and then display the next location.
- II. Load from Papertape L CR
 Action: Will start reading papertape expecting blocks of data in the hex object format. In case of illegal characters, a BCC error, or a length error, the papertape will be stopped and the command ended with the standard error message.

At the end of a successful load, control is passed to the address in the EOF block. This would usually be back to the PIPBUG program.

- III. Dump to Papertape Dssss\eeee CR
 Action: Will punch a leader of 50 blanks and then output the contents of locations 'ssss' to 'eeee', inclusive, in hex object format. When done, the EOF block and a trailer of 50 blanks are punched.
- IV. See and Set the Microprocessor Registers Sn CR
 Action: The parameter 'n' is in the range 0 to 8 and selects a particular register;
- 0 = register 0
 - 1 = register 1 bank #0
 - 2 = register 2 bank #0
 - 3 = register 3 bank #0
 - 4 = register 1 bank #1
 - 5 = register 2 bank #1
 - 6 = register 3 bank #1
 - 7 = PSW upper
 - 8 = PSW lower
- The contents will be displayed. The user can respond with:
- 1) CR which ends the command
 - 2) LF which displays the next register's content
 - 3) nn CR which resets the register to 'nn' and ends the command
 - 4) nn LF which resets the register to 'nn' and displays the next register's content
- V. Go To Gaaaa CR
 Action: Control will be transferred to location 'aaaa' after restoring the register contents.
- VI. Clear Breakpoints Ci CR
 Action: Will clear the ith breakpoint. If the ith breakpoint is not set, gives error message.
- VII. Set Breakpoints Bi\aaaa CR
 Action: Will set the ith breakpoint at the address 'aaaa'. The current firmware supports two breakpoints.

BREAKPOINTS

Breakpoints are a way to interrupt the execution of the program and microprocessor's status is saved prior to executing at the breakpoint address. Up to two breakpoints can be set. So i equals 1 or 2.

BREAKPOINTS (Continued)

Setting a breakpoint at location '1053' with the command 'B1 1053' causes the two bytes of program at '1053' and '1054' to be stored in a table in PIPBUG's RAM area. They are replaced by the two byte instruction 'ZBRR *BKP1'. At location 'BKP1' in the interrupt vector is the address of the 1st breakpoint handling routine. There is a separate routine for the 2nd breakpoint.

When the user program executes the instruction at location '1053', the ZBRR instruction jumps to the breakpoint routine. This routine first saves the microprocessor registers, then restores the two bytes of user program to locations '1053' and '1054', prints the breakpoint address '1053', and finally jumps to PIPBUG. Now the user can use the See command to examine the microprocessor registers.

Since the breakpoints are software implemented and are cleared when reached, there will not be another breakpoint when the user program is re-executed. It must be explicitly re-set with the Breakpoint command. Breakpoints will remain in memory until executed or explicitly cleared with the Clear command.

SUGGESTIONS ON USING

Having written and assembled a program, the user has a papertape containing the object code for the program. The Load command is used to read the code into the RAM of

the 2650 PC1001 card. In the operand field of the END directive of the program, the user should put blanks or a zero, so that after reading the tape PIPBUG restarts itself.

Most commonly the loaded program is still under development. The user wants to run and test only parts of the program. He can use the Goto and Breakpoint commands to isolate the particular code sequence. The two breakpoints can be set at the normal and error exits of the code. Using the Goto command the user then transfers control to the starting address of the code. Remember that the microprocessor's registers can be pre-set using the See command.

If there is a bug, the user can make machine language patches to the program with the Alter command. Great care should be taken when doing this, since assemblers are more methodical than people. The Dump command can be used to save on papertape the program and all patches so that the debugging can be continued at some later time.

SUMMARY

- A Alter memory
- B Set Breakpoint
- C Clear Breakpoint
- D Dump memory to papertape
- G Goto address
- L Load memory from papertape
- S See and alter registers

APPENDIX

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 1

LINE ADDR B1 B2 B3 B4 ERR SOURCE

1	0001					P	EQU	1	
2	0002					N	EQU	2	
3	0000					Z	EQU	0	
4	0002					LCOM	EQU	H'02'	LOGICAL COMPARE
5	0001					CAR	EQU	H'01'	CARRY
6	0000					SENS	EQU	H'00'	SENSE
7	0040					FLAG	EQU	H'40'	FLAG
8	0020					II	EQU	H'20'	INTERRUPT INHIB
9	0020					IDC	EQU	H'20'	INTER DIGIT CAR
10	0004					OYF	EQU	H'04'	OYEFFLOW
11	0000					R0	EQU	0	
12	0001					R1	EQU	1	
13	0002					R2	EQU	2	
14	0003					R3	EQU	3	
15	0003					UN	EQU	3	
16	0000					EQ	EQU	0	
17	0002					LT	EQU	2	
18	0001					GT	EQU	1	
19	0000					WC	EQU	H'00'	
20	0010					RS	EQU	H'10'	
21	0020					SPAC	EQU	H'20'	
22	0001					BMAX	EQU	1	NO. BKPTS - 1
23	007F					DELE	EQU	H'7F'	
24	000D					CR	EQU	13	
25	000A					LF	EQU	10	
26	0014					BLEN	EQU	20	
27	003A					STAR	EQU	A':'	
28						*			
29						ORG		0	
30	0000	07	3F			INIT	LODI,R3	63	ZERO MARK VECTOR AND 0
31	0002	20					EORZ	R0	
32	0003	CF	44	00		AINI	STRA,R0	COM.R3,-	
33	0006	5B	7B				BRNR,R3	AINI	
34	0008	04	77				LODI,R0	H'77'	
35	000A	CC	04	09			STRA,R0	XGOT	LOAD THE RAM CODE TO S
36	000D	04	1B				LODI,R0	H'1B'	
37	000F	CC	04	0B			STRA,R0	XGOT+2	
38	0012	04	80				LODI,R0	H'80'	
39	0014	CC	04	0C			STRA,R0	XGOT+3	
40	0017	1B	09				BCTR,UN	MBUG	
41	0019	01	60			VEC	ACON	BK01	BREAKPOINT VECTOR
42	001B	01	6E				ACON	BK02	
43						*			
44						* COMMAND HANDLER			
45	001D	04	3F			EBUG	LODI,R0	A'??'	ERROR RETURN FOR ALL R
46	001F	3F	02	B4			BSTA,UN	COUT	
47	0022	75	FF			MBUG	CPSL	H'FF'	START OF CMD LOOP, RES
48	0024	3F	00	8A			BSTA,UN	CRLF	
49	0027	04	2A				LODI,R0	A'*'	
50	0029	3F	02	B4			BSTA,UN	COUT	
51	002C	3B	2D				BSTR,UN	LINE	DONT CARE IF THERE IS
52	002E	20					EORZ	R0	

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 2

LINE ADDR B1 B2 B3 B4 ERR SOURCE

```

53 002F CC 04 27          STRA,R0      BPTR
54 0032 0C 04 13          LODA,R0      BUFF
55 0035 E4 41             COMI,R0      A'A'
56 0037 1C 00 AB          BCTA,EQ      ALTE
57 003A E4 42             COMI,R0      A'B'
58 003C 1C 01 E5          BCTA,EQ      BKPT
59 003F E4 43             COMI,R0      A'C'
60 0041 1C 01 CA          BCTA,EQ      CLR
61 0044 E4 44             COMI,R0      A'D'
62 0046 1C 03 10          BCTA,EQ      DUMP
63 0049 E4 47             COMI,R0      A'G'
64 004B 1C 01 3A          BCTA,EQ      GOTO
65 004E E4 4C             COMI,R0      A'L'
66 0050 1C 03 B5          BCTA,EQ      LOAD
67 0053 E4 53             COMI,R0      A'S'
68 0055 1C 00 F4          BCTA,EQ      SREG
69 0058 1F 00 1D          BCTA,UN      EBUG
70
71 * INPUT A CMD LINE INTO BUFFER
72 005B 07 FF          * CODE IS 1=CR 2=LF 3=MSG+CR 4=MSG+LF
73 005D CF 04 27          LINE        LODI,R3      -1
74 0060 E7 14             LLIN        STRA,R3      BPTR
75 0062 18 19             BCTR,EQ     BLEN
76 0064 3F 02 86          BSTA,UN     CHIN          ON BUFFER OVERFLOW FOR
77 0067 E4 7F             COMI,R0     DELE          GET CHAR
78 0069 98 0E             BCFR,EQ     ALIN
79 006B E7 FF             COMI,R3     -1          ECHO AND BACK PTR
80 006D 18 71             BCTR,EQ     LLIN
81 006F 0F 64 13          LODA,R0     BUFF,R3
82 0072 3F 02 B4          BSTA,UN     COUT
83 0075 A7 01             SUBI,R3     1
84 0077 1B 67             BCTR,UN     LLIN
85 0079 E4 0D             ALIN        COMI,R0     CR
86 007B 98 18             BCFR,EQ     BLIN
87 007D 05 01             ELIN        LODI,R1     1
88 007F 03             CLIN        LODZ       R3
89 0080 1A 02             BCTR,N      DLIN
90 0082 85 02             ADDI,R1     2
91 0084 CD 04 2A          DLIN        STRA,R1     CODE
92 0087 CF 04 29          STRA,R3     CNT
93 008A 04 0D             CRLF        LODI,R0     CR
94 008C 3F 02 B4          BSTA,UN     COUT
95 008F 04 0A             LODI,R0     LF
96 0091 3F 02 B4          BSTA,UN     COUT
97 0094 17             RETC,UN
98 0095 05 02             BLIN        LODI,R1     2
99 0097 E4 0A             COMI,R0     LF
100 0099 18 64             BCTR,EQ     CLIN
101 009B CF 24 13          STRA,R0     BUFF,R3,+ STROE CHAR AND ECHO
102 009E 3F 02 B4          BSTA,UN     COUT
103 00A1 1F 00 60          BCTA,UN     LLIN
104 *

```

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 3

```

LINE ADDR B1 B2 B3 B4 ERR SOURCE
105          * SUBR THAT STORES DOUBLE PRECISION INTO TEMP
106 00A4 CD 04 0D   STRT   STRA,R1   TEMP
107 00A7 CE 04 0E          STRA,R2   TEMP+1
108 00AA 17          RETC,UN
109          * DISPLAY AND ALTER MEMORY
110 00AB 3F 02 DB   ALTE  BSTA,UN   GNUM
111 00AE 3B 74          LALT  BSTA,UN   STRT
112 00B0 3F 02 69          BSTA,UN   BOUT
113 00B3 0D 04 0E          LODA,R1   TEMP+1
114 00B6 3F 02 69          BSTA,UN   BOUT
115 00B9 3F 03 5B          BSTA,UN   FORM
116 00BC 0D 04 0D          LODA,R1   *TEMP      DISPLAY CONTENT
117 00BF 3F 02 69          BSTA,UN   BOUT
118 00C2 3F 03 5B          BSTA,UN   FORM
119 00C5 3F 00 5B          BSTA,UN   LINE
120 00C8 0C 04 2A          LODA,R0   CODE
121 00CB E4 02          COMI,R0   2
122 00CD 1E 00 22          BCTA,LT   MBUG
123 00D0 18 11          BCTR,EQ   DALT
124 00D2 CC 04 11   CALT  STRA,R0   TEMR
125 00D5 3F 02 DB          BSTA,UN   GNUM
126 00D8 CE 04 0D          STRA,R2   *TEMP      UPDATE CONTENTS
127 00DB 0C 04 11          LODA,R0   TEMR
128 00DE E4 04          COMI,R0   4
129 00E0 9C 00 22          BCFA,EQ   MBUG
130 00E3 06 01   DALT  LODI,R2   1          INCR CURRENT ADDRESS
131 00E5 8E 04 0E          ADDA,R2   TEMP+1
132 00E8 05 00          LODI,R1   0
133 00EA 77 08          PPSL     WC
134 00EC 8D 04 0D          ADDA,R1   TEMP
135 00EF 75 08          CPSL     WC
136 00F1 1F 00 AE          BCTA,UN   LALT
137          * SELECTIVELY DISPLAY AND ALTER REGISTERS
138 00F4 3F 02 DB   SREG  BSTA,UN   GNUM      GET INDEX OF REG
139 00F7 E6 08   LSRE  COMI,R2   8          CHECK RANGE
140 00F9 1D 00 1D          BCTA,GT   EBUG
141 00FC CE 04 11          STRA,R2   TEMR
142 00FF 0E 64 00          LODA,R0   COM,R2   DISPLAY CONTENTS
143 0102 C1          STRZ     R1
144 0103 3F 02 69          BSTA,UN   BOUT
145 0106 3F 03 5B          BSTA,UN   FORM
146 0109 3F 00 5B          BSTA,UN   LINE
147 010C 0C 04 2A          LODA,R0   CODE
148 010F E4 02          COMI,R0   2
149 0111 1E 00 22          BCTA,LT   MBUG      CR
150 0114 18 1C          BCTR,EQ   CSRE      LF
151 0116 CC 04 0F   ASRE  STRA,R0   TEMR      UPDATE CONTENTS, THEN
152 0119 3F 02 DB          BSTA,UN   GNUM
153 011C 02          LODZ     R2
154 011D 0E 04 11          LODA,R2   TEMR
155 0120 CE 64 00          STRA,R0   COM,R2
156 0123 E6 08          COMI,R2   8          MUST UPDATE PSW LOWER

```

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 4

LINE ADDR B1 B2 B3 B4 ERR SOURCE

```

157 0125 98 03          BCFR.EQ      BSRE
158 0127 CC 04 0A          STRA.R0      XGOT+1
159 012A 0C 04 0F          BSRE         LODA.R0      TEMR
160 012D E4 03          COMI.R0      3
161 012F 1C 00 22          BCTA.EQ      MBUG
162 0132 0E 04 11          CSRE         LODA.R2      TEMR
163 0135 86 01          ADDI.R2      1
164 0137 1F 00 F7          BCTA.UN      LSRE
165          * GOTO ADDRESS
166 013A 3F 02 DB          GOTO         BSTA.UN      GNUM
167 013D 3F 00 A4          BSTA.UN      STRT          PUT ADDR IN RAM
168 0140 0C 04 07          LODA.R0      COM+7
169 0143 92          LPSU
170 0144 0D 04 01          LODA.R1      COM+1        BANK ZERO
171 0147 0E 04 02          LODA.R2      COM+2
172 014A 0F 04 03          LODA.R3      COM+3
173 014D 77 10          PPSL         RS           BANK ONE
174 014F 0D 04 04          LODA.R1      COM+4
175 0152 0E 04 05          LODA.R2      COM+5
176 0155 0F 04 06          LODA.R3      COM+6
177 0158 0C 04 00          LODA.R0      COM
178 015B 75 FF          CPSL         H'FF'
179 015D 1F 04 09          BCTA.UN      XGOT        AND BCTA.UN $TEMP
180          *
181          *BREAKPOINT RUNTIME CODE
182 0160 CC 04 00          BK01         STRA.R0      COM          ENTRY FOR BKPT-1 VIA V
183 0163 13          SPSL
184 0164 CC 04 08          STRA.R0      COM+8
185 0167 CC 04 0A          STRA.R0      XGOT+1      IN RAM FOR REG RESTORE
186 016A 04 00          LODI.R0      0           BKPT INDEX
187 016C 1B 0C          BCTR.UN      BKEN
188 016E CC 04 00          BK02         STRA.R0      COM          ENTRY FOR BKPT-2
189 0171 13          SPSL
190 0172 CC 04 08          STRA.R0      COM+8
191 0175 CC 04 0A          STRA.R0      XGOT+1      IN RAM FOR REG RESTORE
192 0178 04 01          LODI.R0      1
193 017A CC 04 11          BKEN         STRA.R0      TEMR
194 017D 12          SPSU
195 017E CC 04 07          STRA.R0      COM+7
196 0181 77 10          PPSL         RS
197 0183 CD 04 04          STRA.R1      COM+4
198 0186 CE 04 05          STRA.R2      COM+5
199 0189 CF 04 06          STRA.R3      COM+6
200 018C 75 10          CPSL         RS           FORCE TO BANK ZERO
201 018E CD 04 01          STRA.R1      COM+1
202 0191 CE 04 02          STRA.R2      COM+2
203 0194 CF 04 03          STRA.R3      COM+3
204 0197 0E 04 11          LODA.R2      TEMR
205 019A 3B 0F          BSTR.UN      CLBK
206 019C 0D 04 0D          LODA.R1      TEMP        PRINT BKPT ADDR
207 019F 3F 02 69          BSTA.UN      BOUT
208 01A2 0D 04 0E          LODA.R1      TEMP+1

```

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 5

LINE	ADDR	B1	B2	B3	B4	ERR	SOURCE
209	01A5	3F	02	69			BSTA.UN BOUT
210	01A8	1F	00	22			BCTA.UN MBUG
211							* SUBR TO CLEAR A BKPT LIKE MANY SUBR HAS REL ADDR
212	01AB	20					CLBK EORZ R0
213	01AC	CE	64	2D			STRA.R0 MARK.R2
214	01AF	0E	64	33			LODA.R0 HADR.R2
215	01B2	CC	04	0D			STRA.R0 TEMP
216	01B5	0E	64	35			LODA.R0 LADR.R2
217	01B8	CC	04	0E			STRA.R0 TEMP+1
218	01B8	0E	64	2F			LODA.R0 HDAT.R2
219	01BE	CC	04	0D			STRA.R0 *TEMP
220	01C1	0E	64	31			LODA.R0 LDAT.R2
221	01C4	07	01				LODI.R3 1
222	01C6	CF	E4	0D			STRA.R0 *TEMP.R3
223	01C9	17					RETC.UN
224							* BREAK POINT MARK INDICATES IF SET
225							* HADR +LADR IS BKPT ADDR. HDAT + LDAT IS TWO BYTE
226	01CA	3B	0B				CLR BSTR.UN NOK
227	01CC	0E	64	2D			LODA.R0 MARK.R2 CLEAR IT IF SET
228	01CF	1C	00	1D			BCTA.Z EBUG
229	01D2	3B	57				BSTR.UN CLBK
230	01D4	1F	00	22			BCTA.UN MBUG
231	01D7	3F	02	DB			NOK BSTA.UN GNUM CHECK RANGE ON BKPT NUMB
232	01DA	A6	01				SUBI.R2 1
233	01DC	1E	02	50			BCTA.N ABRT
234	01DF	E6	01				COMI.R2 BMAX
235	01E1	1D	02	50			BCTA.GT ABRT
236	01E4	17					RETC.UN
237	01E5	3B	70				BKPT BSTR.UN NOK SET BKPT AND CLR ANY E
238	01E7	0E	64	2D			LODA.R0 MARK.R2
239	01EA	BC	01	AB			BSFA.Z CLBK CLEAR EXISTING
240	01ED	CE	04	11			STRA.R2 TEMR
241	01F0	3F	02	DB			BSTA.UN GNUM GET BKPT ADDR
242	01F3	3F	00	A4			BSTA.UN STRT SUBR TO STORE R1-R2 IN
243	01F6	0F	04	11			LODA.R3 TEMR
244	01F9	02					LODZ R2
245	01FA	CF	64	35			STRA.R0 LADR.R3
246	01FD	01					LODZ R1
247	01FE	CF	64	33			STRA.R0 HADR.R3
248	0201	0C	04	0D			LODA.R0 *TEMP SAVE CONTENTS
249	0204	CF	64	2F			STRA.R0 HDAT.R3
250	0207	05	9B				LODI.R1 H'9B' = ZBRR
251	0209	CD	04	0D			STRA.R1 *TEMP
252	020C	06	01				LODI.R2 1
253	020E	0E	E4	0D			LODA.R0 *TEMP.R2
254	0211	CF	64	31			STRA.R0 LDAT.R3
255	0214	0F	62	22			LODA.R0 DISP.R3
256	0217	CE	E4	0D			STRA.R0 *TEMP.R2
257	021A	04	FF				LODI.R0 -1
258	021C	CF	64	2D			STRA.R0 MARK.R3
259	021F	1F	00	22			BCTA.UN MBUG
260	0222	99					DISP DATA VEC+H'00'

```

LINE ADDR B1 B2 B3 B4 ERR SOURCE
261 0223 9B          DATA          VEC+H'80'+2
262
263          *
264 0224 3F 02 86    BIN          BSTA.UN          CHIN
265 0227 3B 1D          BSTR.UN          LKUP
266 0229 D3          RRL.R3
267 022A D3          RRL.R3
268 022B D3          RRL.R3
269 022C D3          RRL.R3
270 022D CF 04 12    STRA.R3          TEMS
271 0230 3F 02 86    BSTA.UN          CHIN
272 0233 3B 11          BSTR.UN          LKUP
273 0235 6F 04 12    IORA.R3          TEMS
274 0238 03          LODZ            R3
275 0239 C1          STRZ            R1
276 023A 3B 01          BSTR.UN          CBCC
277 023C 17          RETC.UN
278          * CALCULATE THE BCC CHAR, EOR AND THEN ROTATE LEFT
279 023D 01          CBCC LODZ            R1
280 023E 2C 04 2C          EORA.R0          BCC
281 0241 D0          RRL.R0
282 0242 CC 04 2C          STRA.R0          BCC
283 0245 17          RETC.UN
284          * LOOKUP ASCII CHAR IN HEX VALUE TABLE
285 0246 07 10          LKUP LODI.R3          16
286 0248 EF 42 59          ALKU  COMA.R0          ANSI.R3,-
287 024B 14          RETC.EQ
288 024C E7 01          COMI.R3          1
289 024E 9A 78          BCFR.LT          ALKU
290          * ABORT EXIT FROM ANY LEVEL OF SUBR
291          * USE RAS PTR SINCE POSSIBLE BKPT PROG USING IT
292 0250 0C 04 07          ABRT LODA.R0          COM+?
293 0253 64 40          IORI.R0          H'40'
294 0255 12          SPSU
295 0256 1F 00 1D          BCTA.UN          EBUG
296 0259 30 31 32 33          ANSI  DATA          A'0123456789ABCDEF
      34 35 36 37
      38 39 41 42
      43 44 45 46
297          * BYTE IN R1 OUTPUT IN HEX
298 0269 CD 04 12          BOUT STRA.R1          TEMS
299 026C 3B 4F          BSTR.UN          CBCC
300 026E 51          RRR.R1
301 026F 51          RRR.R1
302 0270 51          RRR.R1
303 0271 51          RRR.R1
304 0272 45 0F          ANDI.R1          H'0F'
305 0274 0D 62 59          LODA.R0          ANSI.R1
306 0277 3F 02 B4          BSTA.UN          COUT
307 027A 0D 04 12          LODA.R1          TEMS
308 027D 45 0F          ANDI.R1          H'0F'
309 027F 0D 62 59          LODA.R0          ANSI.R1

```

PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 7

LINE	ADDR	B1	B2	B3	B4	ERR	SOURCE
310	0282	3F	02	B4			BSTA,UN COUT
311	0285	17					RETC,UN
312							* 110 BAUD INPUT FOR PAPER TAPE AND CHAR 1MHZ CLOCK
313	0286	77	10				CHIN PPSL RS
314	0288	04	80				LODI,R0 H'80' ENABLE TAPE READER
315	028A	B0					WRTC,R0
316	028B	05	00				LODI,R1 0
317	028D	06	08				LODI,R2 8
318	028F	12					ACHI SPSU
319	0290	1A	74				BCTR,LT CHIN LOOK FOR START BIT
320	0292	20					EORZ R0
321	0293	B0					WRTC,R0 DISABLE TAPE READER
322	0294	3B	17				BSTR,UN DLY
323	0296	3B	10				BCHI BSTR,UN DLAY WAIT TO MIDDLE OF DATA
324	0298	12					SPSU
325	0299	44	80				ANDI,R0 H'80' MOVE BIT 7 OF R0 INTO
326	029B	51					RRR,R1
327	029C	61					IORZ R1
328	029D	C1					STRZ R1
329	029E	FA	76				BDRR,R2 BCHI
330	02A0	3B	06				BSTR,UN DLAY
331	02A2	45	7F				ANDI,R1 H'7F' DELETE PARITY BIT
332	02A4	01					LODZ R1
333	02A5	75	18				CPSL RS+WC
334	02A7	17					RETC,UN
335							* DELAY FOR ONE BIT TIME
336	02A8	20					DLAY EORZ R0
337	02A9	F8	7E				BDRR,R0 \$
338	02AB	F8	7E				BDRR,R0 \$
339	02AD	F8	7E				DLY BDRR,R0 \$
340	02AF	04	E5				LODI,R0 H'E5'
341	02B1	F8	7E				BDRR,R0 \$
342	02B3	17					RETC,UN
343							*
344	02B4	77	10				COUT PPSL RS
345	02B6	76	40				PPSU FLAG
346	02B8	C2					STRZ R2
347	02B9	05	08				LODI,R1 8
348	02BB	3B	6B				BSTR,UN DLAY
349	02BD	3B	69				BSTR,UN DLAY
350	02BF	74	40				CPSU FLAG
351	02C1	3B	65				ACOU BSTR,UN DLAY
352	02C3	52					RRR,R2
353	02C4	1A	04				BCTR,LT ONE
354	02C6	74	40				CPSU FLAG
355	02C8	1B	02				BCTR,UN ZERO
356	02CA	76	40				ONE PPSU FLAG
357	02CC	F9	73				ZERO BDRR,R1 ACOU
358	02CE	3B	58				BSTR,UN DLAY
359	02D0	76	40				PPSU FLAG
360	02D2	75	10				CPSL RS
361	02D4	17					RETC,UN

LINE ADDR B1 B2 B3 B4 ERR SOURCE

```

362
363 *
364 02D5 0C 04 2A DNUM LODA,R0 CODE
365 02D8 18 07 BCTR,Z LNUM SKIP SPACES UNTIL REAC
366 02DA 17 RETC,UN OR SPACE ENDING NUMBER
367 02DB 20 GNUM EORZ R0
368 02DC C1 STRZ R1
369 02DD C2 STRZ R2
370 02DE CC 04 2A STRA,R0 CODE
371 02E1 0F 04 27 LNUM LODA,R3 BPTR
372 02E4 EF 04 29 COMA,R3 CNT CHECK FOR E O B
373 02E7 14 RETC,EQ
374 02E8 0F 24 13 LODA,R0 BUFF,R3,+ GET CHAR
375 02EB CF 04 27 STRA,R3 BPTR
376 02EE E4 20 COMI,R0 SPAC
377 02F0 18 63 BCTR,EQ DNUM
378 02F2 3F 02 46 BNUM BSTA,UN LKUP
379 02F5 04 0F CNUM LODI,R0 H'0F' R1=AB R2=DD
380 02F7 D2 RRL,R2
381 02F8 D2 RRL,R2
382 02F9 D2 RRL,R2
383 02FA D2 RRL,R2
384 02FB 42 ANDZ R2
385 02FC D1 RRL,R1
386 02FD D1 RRL,R1
387 02FE D1 RRL,R1
388 02FF D1 RRL,R1
389 0300 45 F0 ANDI,R1 H'F0'
390 0302 46 F0 ANDI,R2 H'F0' R0=C R1=B0 R2=D0 R3=V
391 0304 61 IORZ R1
392 0305 C1 STRZ R1
393 0306 03 LODZ R3
394 0307 62 IORZ R2
395 0308 C2 STRZ R2 R1=BC R2=DV
396 0309 04 01 LODI,R0 1
397 030B CC 04 2A STRA,R0 CODE
398 030E 1B 51 BCTR,UN LNUM
399 * DUMP TO PAPER TAPE IN OBJECT FORMAT
400 0310 3B 49 DUMP BSTR,UN GNUM START ADDRESS
401 0312 3F 00 A4 BSTA,UN STRT SUBR TO STORE R1-R2 IN
402 0315 3B 44 BSTR,UN GNUM
403 0317 86 01 ADDI,R2 1
404 0319 77 08 PPSL WC
405 031B 85 00 ADDI,R1 0
406 031D 75 08 CPSL WC MAKE END ADDR NOT INCL
407 031F CD 04 0F STRA,R1 TEMQ
408 0322 CE 04 10 STRA,R2 TEMQ+1
409 0325 3B 38 FDUM BSTR,UN GAP
410 0327 04 FF LODI,R0 -1
411 0329 CC 04 29 STRA,R0 CNT
412 032C 3F 00 8A BSTA,UN CRLF PUNCH FOR CR/LF AND ST
413 032F 04 3A LODI,R0 STAR
    
```


PIP ASSEMBLER VERSION 3 LEVEL 1

PAGE 9

LINE	ADDR	B1	B2	B3	B4	ERR	SOURCE
414	0331	3F	02	B4			BSTA,UN COUT
415	0334	20					EORZ R0
416	0335	CC	04	2C			STRA,R0 BCC
417	0338	0D	04	0F			LODA,R1 TEMQ
418	033B	0E	04	10			LODA,R2 TEMQ+1
419	033E	AE	04	0E			SUBA,R2 TEMP+1
420	0341	77	08				PPSL WC
421	0343	AD	04	0D			SUBA,R1 TEMP
422	0346	75	08				CPSL WC
423	0348	1E	00	1D			BCTA,N EBUG
424	034B	19	1C				BCTR,P ADUM
425	034D	5A	1C				BRNR,R2 BDUM
426	034F	07	04				LODI,R3 4
427	0351	3F	02	69		CDUM	BSTA,UN BOUT
428	0354	FB	7B				BDRR,R3 CDUM
429	0356	3B	07				BSTR,UN GAP
430	0358	1F	00	22			BCTA,UN MBUG
431							* SUBRS FOR OUTPUTTING BLANKS
432	035B	07	03			FORM	LODI,R3 3
433	035D	1B	02				BCTR,UN AGAP
434	035F	07	32			GAP	LODI,R3 50
435	0361	04	20			AGAP	LODI,R0 SPAC
436	0363	3F	02	B4			BSTA,UN COUT
437	0366	FB	79				BDRR,R3 AGAP
438	0368	17					RETC,UN
439	0369	06	FF			ADUM	LODI,R2 255
440	036B	CE	04	28		BDUM	STRA,R2 MCNT
441	036E	0D	04	0D			LODA,R1 TEMP
442	0371	3F	02	69			BSTA,UN BOUT
443	0374	0D	04	0E			LODA,R1 TEMP+1
444	0377	3F	02	69			BSTA,UN BOUT
445	037A	0D	04	28			LODA,R1 MCNT
446	037D	3F	02	69			BSTA,UN BOUT
447	0380	0D	04	2C			LODA,R1 BCC
448	0383	3F	02	69			BSTA,UN BOUT
449	0386	0F	04	29		DDUM	LODA,R3 CNT
450	0389	0F	A4	0D			LODA,R0 *TEMP,R3,+
451	038C	EF	04	28			COMA,R3 MCNT
452	038F	18	09				BCTR,EQ EDUM
453	0391	CF	04	29			STRA,R3 CNT
454	0394	C1					STRZ R1
455	0395	3F	02	69			BSTA,UN BOUT
456	0398	1B	6C				BCTR,UN DDUM
457	039A	0D	04	2C		EDUM	LODA,R1 BCC
458	039D	3F	02	69			BSTA,UN BOUT
459	03A0	0E	04	0E			LODA,R2 TEMP+1
460	03A3	8E	04	28			ADDA,R2 MCNT
461	03A6	05	00				LODI,R1 0
462	03A8	77	08				PPSL WC
463	03AA	8D	04	0D			ADDA,R1 TEMP
464	03AD	75	08				CPSL WC
465	03AF	3F	00	A4			BSTA,UN STRT

GET BYTE COUNT

START > END ADDR
CNT > NORMAL BLOCK SI
THIS IS SHORT BLOCK
EOF, PUNCH ZERO BLK

STARTING ADDRESS

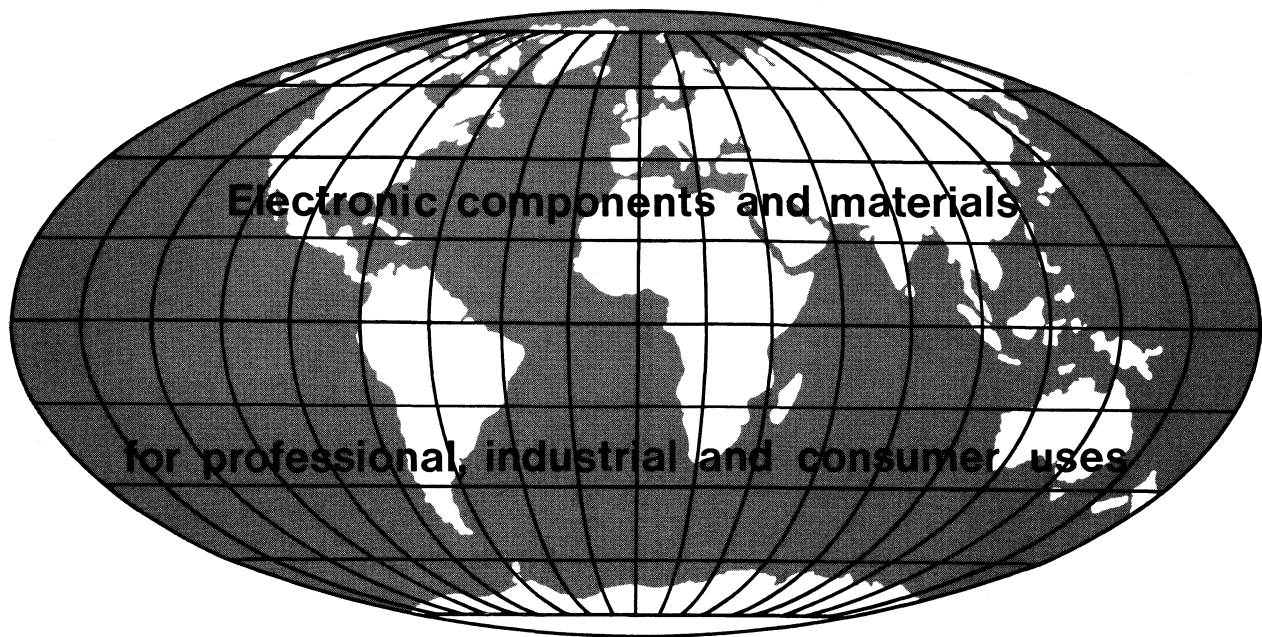
COUNT OF DATA BYTES IN

OUTPUT BCC

```

LINE ADDR B1 B2 B3 B4 ERR SOURCE
466 03B2 1F 03 25          BCTA.UN      FDUM
467                      * LOAD FROM PAPER TAPE IN OBJECT FORMAT
468 03B5 3F 02 86          LOAD  BSTA.UN      CHIN      LOOK FOR START CHAR
469 03B8 E4 3A             COMI.R0      STAR
470 03BA 98 79             BCFR.EQ      LOAD
471 03BC 20                EORZ         R0
472 03BD CC 04 2C          STRA.R0      BCC
473 03C0 3F 02 24          BSTA.UN      BIN          READ ADDR AND COUNT IN
474 03C3 CD 04 0D          STRA.R1      TEMP
475 03C6 3F 02 24          BSTA.UN      BIN
476 03C9 CD 04 0E          STRA.R1      TEMP+1
477 03CC 3F 02 24          BSTA.UN      BIN
478 03CF 59 03             BRNR.R1      ALOA          CNT = 0 MEANS EOF
479 03D1 1F 84 0D          BCTA.UN      *TEMP
480 03D4 CD 04 28          ALOA  STRA.R1      MCNT
481 03D7 3F 02 24          BSTA.UN      BIN          CHECK BCC ON INFORMATI
482 03DA 0C 04 2C          LODA.R0      BCC
483 03DD 9C 00 1D          BCFA.Z       EBUG
484 03E0 C3                STRZ         R3          READ DATA
485 03E1 CF 04 29          BLOA  STRA.R3      CNT
486 03E4 3F 02 24          BSTA.UN      BIN
487 03E7 0F 04 29          LODA.R3      CNT
488 03EA EF 04 28          COMA.R3      MCNT
489 03ED 18 06             BCTR.EQ      CLOA          HAVE READ BCC
490 03EF 01                LODZ         R1
491 03F0 CF E4 0D          STRA.R0      *TEMP.R3      STORE DATA
492 03F3 DB 6C             BIRR.R3      BLOA
493 03F5 0C 04 2C          CLOA  LODA.R0      BCC
494 03F8 9C 00 1D          BCFA.Z       EBUG
495 03FB 1F 03 B5          BCTA.UN      LOAD
496                      *
497                      ORG          H'400'
498                      ***** RAM DEFINITIONS
499 0400          COM      RES      9
500 0409 77 00          XGOT     PPSL      0
501 040B 1B 80          BCTR.UN  *$+2      MUST PREDEED THE TEMP
502 040D          TEMP     RES      2
503 040F          TEMQ     RES      2
504 0411          TEMR     RES      1
505 0412          TEMS     RES      1
506 0413          BUFF     RES      BLEN
507 0427          BPTR     RES      1
508 0428          MCNT     RES      1
509 0429          CNT      RES      1
510 042A          CODE     RES      1
511 042B          OKGO     RES      1
512 042C          BCC      RES      1
513 042D          MARK     RES      BMAX+1
514 042F          HDAT     RES      BMAX+1
515 0431          LDAT     RES      BMAX+1
516 0433          HADR     RES      BMAX+1
517 0435          LADR     RES      BMAX+1

```

from the world-wide Philips Group of Companies

EUROPEAN SALES OFFICES

Austria: Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.

Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.

Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.

Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.

France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.

Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.

Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.

Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.

Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.

Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.

Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.

Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.

Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.

Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.

Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.

Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoğlu, ISTANBUL.

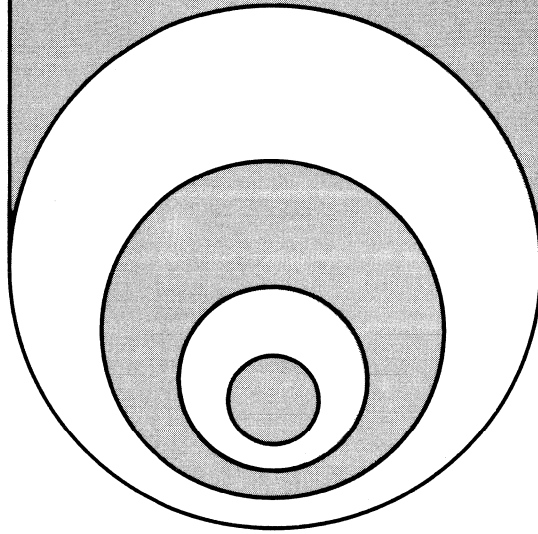
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

© N.V. Philips' Gloeilampenfabrieken

This information is furnished for guidance, and with no guarantees as to its accuracy or completeness; its publication conveys no licence under any patent or other right, nor does the publisher assume liability for any consequence of its use; specifications and availability of goods mentioned in it are subject to change without notice; it is not to be reproduced in any way, in whole or in part, without the written consent of the publisher.

signetics

MICROPROCESSOR



2650 INITIALIZATION.....MP51

APPLICATIONS MEMO

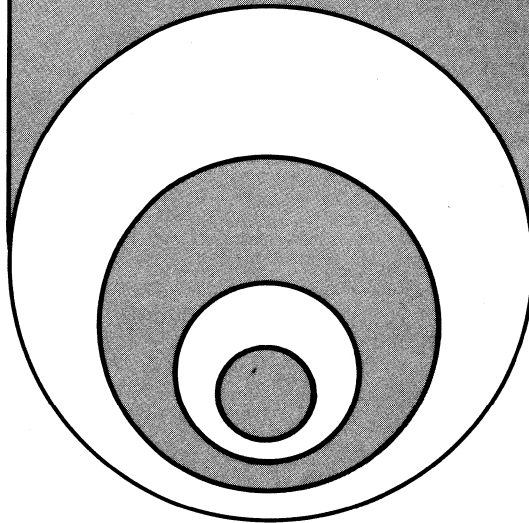
At power-up the status of the 2650 is undefined. The Reset signal should be raised for at least three clock periods. This forces execution of the instruction at location 0. Once the system is started up, the first program to run is generally responsible for initializing the microprocessor, memory, and I/O devices to their desired initial states. The type of I/O initialization is dependent on the particular device. Contents of RAM are undefined at power-up and must be set to their desired initial states.

Program status word initialization:

1. Interrupts can be inhibited as a first step in initialization. The Reset clears the Interrupt Inhibit bit and the internal Interrupt Waiting signal. After the remainder of the status bits, the memory, and the I/O is initialized, interrupts can be permitted. This procedure will prevent unwanted interrupts during system initialization. If the system does not utilize interrupts, the Interrupt Inhibit bit can be left set on when system initialization is complete. This approach will assure that a spurious interrupt will not occur.
2. The Stack Pointer may be initialized to zero. The Stack Pointer should not be modified during the execution of a program. This pointer is under the control of the processor. Modification by a program could have unwanted results, i.e., to the instruction address register.
3. It is generally unnecessary to initialize the Condition Code, Interdigit Carry, Overflow, and Carry bits. These bits are normally set by arithmetic and logical operations before they are tested. However, if the With Carry bit is set on, then the Carry bit should be initialized correctly for the first arithmetic instruction.
4. The Register Select bit should be set to a known state, e.g. if bank 1 registers are reserved for interrupt routines, the register select bit should be initialized to bank 0.
5. The With Carry bit can be initialized to the state desired for most arithmetic and rotate operations. Then if a different state is desired for some operations, the With Carry bit can be changed and then restored after these operations.
6. The same philosophy used for the With Carry bit also applies to the Compare bit. Set the Compare bit initially to the most frequent types of compares made, logical or arithmetic.
7. The Sense bit cannot be modified by a program. The Flag bit may need to be initialized if there is a device connected to it such as a TTY which needs stop bits (binary one) when not receiving data.

signetics

**MOS
MICROPROCESSOR**



**2650 DEMO
SYSTEM
SP51**

2650 MICROPROCESSOR APPLICATIONS MEMO

GENERAL

The Demo System (DS) is a hardware base for use with the 2650 CPU printed circuit board (PC1001). The DS provides the user of the 2650 with a convenient "lab bench" set-up for exercising the PC1001 CPU board. The user may expand memory, implement I/O functions, and step through program instructions one at a time using the DS. When the DS is combined with a CPU board (PC1001) and a keyboard terminal, the user is equipped with everything he needs to exercise any of the software or hardware features of the 2650. There are two versions of the DS, the DS1000 and the DS2000. The two Demo Systems are the same except that the DS2000 has a built-in power supply and therefore does not have the power supply binding posts.

FEATURES

The DS provides several connectors to aid the user in exercising the PC1001 CPU board including one for the CPU

board itself, one for a memory expansion board, four for I/O ports, and two for communicating with the user's terminal. There are four sets of LED lamps that display the information on the address bus, the data bus, and the two non-extended I/O ports. Two control switches (RUN/PAUSE, and STEP) allow the user to place the 2650 in the WAIT mode and step through program execution one instruction at a time. A reset button is provided on the DS. The DS1000 version has five-way binding posts for connection to external power supplies. The DS2000 has built-in power supplies and does not have the five-way binding posts.

CONNECTORS:

2650 CPU Board Edge Connector (J8). The CPU board connector is an Amphenol dual 50-pin connector (series 225) with 0.125-inch contact centers. The 2650 CPU board (PC1001) is inserted into J8 to complete the Demo

CPU BOARD AND USER BOARD CONNECTORS

PIN#	FUNCTION (J7 & J8)	PIN#	FUNCTION (J8 ONLY)*
1	GND	A	GND
2	GND	B	GND
3	NC**	C	NC
4	$\overline{\text{DBUS0}}$	D	OPD 0
5	$\overline{\text{DBUS1}}$	E	OPD 1
6	$\overline{\text{DBUS2}}$	F	OPD 2
7	$\overline{\text{DBUS3}}$	H	OPD 3
8	$\overline{\text{DBUS4}}$	J	OPD 4
9	$\overline{\text{DBUS5}}$	K	OPD 5
10	$\overline{\text{DBUS6}}$	L	OPD 6
11	$\overline{\text{DBUS7}}$	M	OPD 7
12*	EIPD	N	COPD
13	$\overline{\text{D/C}}$	P	TTY SERIAL IN +
14	$\overline{\text{DMA}}$	R	TTY SERIAL IN -
15	$\overline{\text{E/NE}}$	S	TTY SERIAL OUT +
16	INTACK	T	TTY SERIAL OUT -
17	$\overline{\text{R/W}}$	U	RS232 GROUND
18	WRP	V	RS232 OUTPUT
19	$\overline{\text{RUN/WAIT}}$	W	TTY TAPE READER OUT -
20	OPREQ	X	TTY TAPE READER OUT +
21	$\overline{\text{M/I0}}$	Y	RS232 INPUT
22	$\overline{\text{OPACK}}$	Z	COPC
23	CLOCK	a	OPC 0
24	$\overline{\text{OPEX}}$	b	OPC 1
25	RESET	c	OPC 2

PIN#	FUNCTION (J7 & J8)	PIN#	FUNCTION (J8 ONLY)*
26	$\overline{\text{INTREQ}}$	d	OPC 3
27	$\overline{\text{PAUSE}}$	e	OPC 4
28	NC	f	OPC 5
29	NC	g	OPC 6
30	NC	h	OPC 7
31	NC	j	EIPC
32	NC	k	IPD 0
33	ABUS 11	m	IPD 1
34	ABUS 13	n	IPD 2
35	ABUS 12	p	IPD 3
36	ABUS 14	r	IPD 4
37	ABUS 9	s	IPD 5
38	ABUS 10	t	IPD 6
39	ABUS 8	u	IPD 7
40	ABUS 7	v	IPC 0
41	ABUS 6	w	IPC 1
42	ABUS 5	x	IPC 2
43	ABUS 3	y	IPC 3
44	ABUS 0	z	IPC 4
45	ABUS 1	$\overline{\text{a}}$	IPC 5
46	ABUS 4	$\overline{\text{b}}$	IPC 6
47	ABUS 2	$\overline{\text{c}}$	IPC 7
48	+12V	$\overline{\text{d}}$	+12V
49	-12V	$\overline{\text{e}}$	-12V
50	+5V	$\overline{\text{g}}$	+5V

*J7 has no connections to these pins.

**NC = No Connection

TABLE 1

DEMO SYSTEM LAYOUT

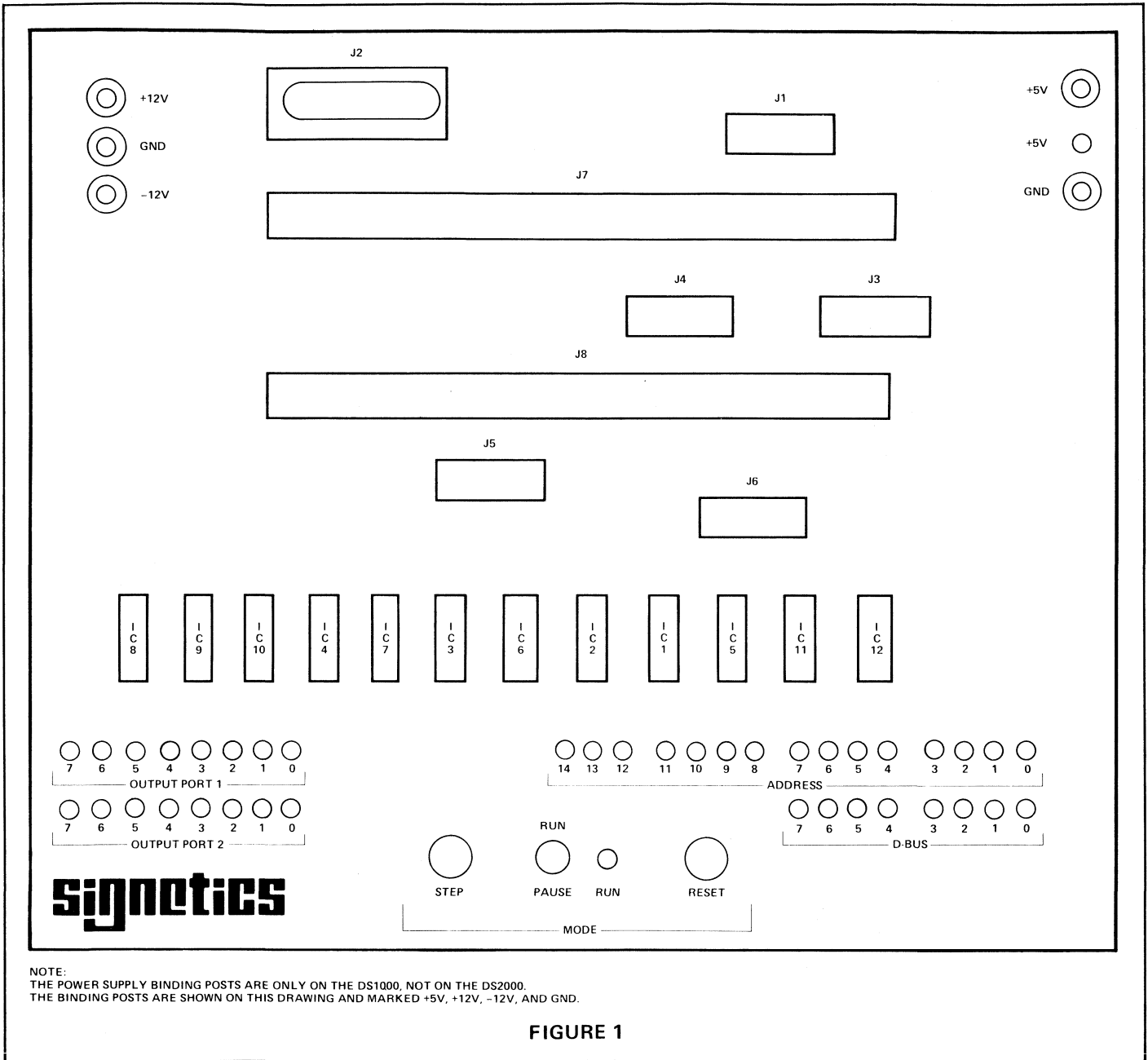


FIGURE 1

CONNECTORS (Continued)

System. The correlation between signal names and pin numbers for J8 is given in Table 1. The location of J8 on the DS is shown in Figure 1.

User Printed Circuit Board Edge Connector (J7). The user board connector is the same type of connector as J8 (the CPU board connector), and makes address, data and control lines available for user-defined interface functions. As shown in Table 1, the numbered pins of J7 and J8 have the same signals on them (except pin 12), while the lettered

pins of J7 (pins A through G) are not used. The J7 connector is typically used for memory expansion. The location of J7 on the DS is shown in Figure 1.

Extended Input/Output DIP Sockets (J5 & J6). The extended I/O DIP sockets make the signals shown in Table 2 available to the user of the DS system. With the signals available on J5 and J6, any type of I/O interface to the 2650 may be implemented. The user of these sockets must supply the cable between his system and the DS, as well as the two 18-pin DIP plugs. The location of J5 and J6 is shown in Figure 1.

EXTENDED INPUT/OUTPUT DIP SOCKETS

PIN #	FUNCTION	
	J5	J6
1	$\overline{\text{DBUS 0}}$	ABUS 0
2	$\overline{\text{DBUS 1}}$	ABUS 1
3	$\overline{\text{DBUS 2}}$	ABUS 2
4	$\overline{\text{DBUS 3}}$	ABUS 3
5	$\overline{\text{DBUS 4}}$	ABUS 4
6	$\overline{\text{DBUS 5}}$	ABUS 5
7	$\overline{\text{DBUS 6}}$	ABUS 6
8	$\overline{\text{DBUS 7}}$	ABUS 7
9	$\overline{\text{OPACK}}$	ABUS 8
10	$\overline{\text{M/IO}}$	ABUS 9
11	OPREQ	ABUS 10
12	$\overline{\text{RUN/WAIT}}$	ABUS 11
13	WRP	ABUS 12
14	$\overline{\text{R/W}}$	ABUS 13
15	INTACK	ABUS 14
16	$\overline{\text{E/NE}}$	$\overline{\text{PAUSE}}$
17	$\overline{\text{DMA}}$	$\overline{\text{INTREQ}}$
18	$\overline{\text{D/C}}$	CLOCK

TABLE 2

Non-Extended Input/Output DIP Sockets (J3 & J4). Each non-extended I/O DIP socket (J3 and J4) makes the signals shown in Table 3 available to the user of the DS system. These sockets may be used for data or command transfer between the 2650 CPU and a user-defined function, but transfers via these channels are initiated by the CPU only. The user of these sockets must supply the cable between his system and the DS, as well as the 18-pin DIP plugs. The location of J3 and J4 is shown in Figure 1.

NON-EXTENDED INPUT/OUTPUT DIP SOCKETS

PIN #	FUNCTION	
	J3	J4
1	OPC 0	OPD 0
2	OPC 1	OPD 1
3	OPC 2	OPD 2
4	OPC 3	OPD 3
5	OPC 4	OPD 4
6	OPC 5	OPD 5
7	OPC 6	OPD 6
8	OPC 7	OPD 7
9	COPC	COPD
10	EIPC	EIPD
11	IPC 7	IPD 7
12	IPC 6	IPD 6
13	IPC 5	IPD 5
14	IPC 4	IPD 4
15	IPC 3	IPD 3
16	IPC 2	IPD 2
17	IPC 1	IPD 1
18	IPC 0	IPD 0

TABLE 3

RS232 Interface Connector (J2). The RS232 interface connector is a TRW 25-pin connector (part #DB25S) for communicating with RS232-compatible input/output devices. The pins used on this connector are shown in Table 4 along with the corresponding signal names. The RS232 driver and receiver are on the PC1001 circuit board and are wired to J2 through the DS circuit board. The location of J2 on the DS board is shown in Figure 1.

RS232 INTERFACE CONNECTOR (J2)

PIN #	FUNCTION – J2
1	RS232 GROUND
2	RS232 INPUT
3	RS232 OUTPUT
5	JUMPER
6	JUMPER
7	RS232 GROUND
8	JUMPER
20	JUMPER

TTY INTERFACE DIP SOCKET (J1)

PIN #	FUNCTION – J1
1	TTY SERIAL IN +
2	TTY SERIAL IN -
8	TTY TAPE READER OUT -
9	TTY TAPE READER OUT +
13	TTL SERIAL OUT -
14	TTL SERIAL OUT +

TABLE 4

TTY Interface DIP Socket (J1). The TTY interface socket is a 14-pin DIP socket and is used for communicating with a current loop serial interface. The pins used on this connector are shown in Table 4 along with the corresponding signal names. The current loop driver and receiver circuits are on the PC1001 board and are wired to J1 through the DS circuit board. The location of J1 on the DS board is shown in Figure 1.

DISPLAYS:

Address Display LEDs. The address display LEDs reflect the information on the address bus (ABUS 0-ABUS 14) when the PC1001 board is plugged into J8. The logic circuits on the DS board loads the information from the address bus into D-type latches on the occurrence of every Operation Request (OPREQ) pulse. Open collector inverters at the output of the D-type latches drive the LED's in a common anode configuration.

Data Bus Display LEDs. The data bus display LEDs reflect the information on the data bus (DBUS 0-DBUS 7) when the PC1001 board is plugged into J8. The information on the data bus is stored into D-type latches on every OPREQ pulse. The LEDs are driven directly from the D-type latches in a common anode configuration.

DISPLAYS (Continued)

Non-Extended Input/Output Channel LEDs. The non-extended I/O channel LEDs are driven by open collector inverters in a common anode configuration. The inverters are driven by the output latches of the two non-extended I/O ports on the PC1001 printed circuit board. Output Port 1 (2), bit 0 corresponds to DBUS 0 and Output Port 1 (2), bit 7 corresponds to DBUS 7. A logic "1" output from the 2650 turns on the LEDs, and a logic "0" turns off the LED.

+5V LED and RUN LED. The +5V LED will glow when a +5 volt power supply is connected to the Demo System. The DS1000 requires an external power supply, but the DS2000 has the +5 volt power supply built into the base. The RUN LED will glow when the RUN/WAIT line from the 2650 is in the "high" logic state. The location of these LED's is shown in Figure 1.

CONTROLS:

RESET Button. The reset button is a momentary switch that is tied directly to the Reset input on J8 (pin 25), and pulls that pin "low" when the button is pushed. This button clears the program counter in the 2650 to zero. The location of the reset button is shown in Figure 1.

PAUSE Switch and STEP Button. The pause switch and the step button are used together to cause the 2650 microprocessor to execute one instruction at a time. When the pause switch is in the RUN position, the step button does not affect the operation of the microprocessor.

When the pause switch is placed into the PAUSE position, the $\overline{\text{PAUSE}}$ line on the 2650 is pulled "low". When the execution of the current instruction is completed, the 2650 will enter the WAIT mode and the $\overline{\text{RUN/WAIT}}$ line will go "low". If the step button is pressed, the $\overline{\text{PAUSE}}$ line to the 2650 will be pulled "high" until the $\overline{\text{RUN/WAIT}}$ line goes "high", indicating that the 2650 is in the RUN mode. As soon as the $\overline{\text{RUN/WAIT}}$ line goes "high", the DS will again pull the $\overline{\text{PAUSE}}$ line "low". The step button will allow one instruction to be executed each time it is pushed as long as the pause switch is in the PAUSE position. When the pause switch is placed back onto the RUN position, the $\overline{\text{PAUSE}}$ line will be pulled "high" and the 2650 will execute instructions in a continuous manner. The address and data displayed on the DS LEDs in the WAIT mode reflect the address and the first byte of the next instruction to be executed. The location of the pause and step switches on the DS base is shown in Figure 1.

LOGIC CIRCUITS

The logic circuits on the DS base are shown in Figure 2. The logic circuits consist of address bus (ABUS) and data bus (DBUS) latches, the pause and step logic, LED drivers,

and a reset switch. The address and data bus are loaded into latches on the DS during every OPREQ. The displays for the address and data bus will flicker while the run LED is "lighted", and will display the address and first byte of the *next* instruction to be executed when in the step mode (run LED off). The pause and step logic allows one instruction to be executed at a time by pushing the step button when the run/pause switch is in the PAUSE position. The non-extended output ports are displayed on the DS, and the reset button provides complete system reset by pushing the button.

ADDRESS BUS:

The address bus latches are 74174 Hex D-type flip-flops (IC1, IC2, IC3). Open collector inverters (IC5, IC6, IC7) invert the "positive true" levels from the ABUS latches and drive the address bus LEDs (L1-L15) in a common anode configuration. A logic ONE on the address bus "lights" the corresponding LED, and ABUS 0 corresponds to the ADDRESS bit 0 LED. The ABUS latch is clocked by the STRB signal which is generated by 4 inverters (IC7, IC10). The inverters provide the logic function $\text{STRB} = \text{OPREQ} \cdot \text{CLOCK}$. The ABUS latches are reset by $\overline{\text{RESET}}$.

DATA BUS:

The data bus latches are also 74174 Hex D-type flip-flops (IC3, IC4). Since the DBUS leaves the PC1001 with "negative true" logic levels, the DBUS latches drive the LEDs directly in a common anode configuration. A logic ONE in the DBUS latches is a low voltage level and "lights" the corresponding LED. The DBUS bit 0 LED corresponds to DBUS 0. The DBUS latch is also clocked by the signal STRB, and reset by $\overline{\text{RESET}}$.

PAUSE AND STEP:

The pause and step switches are de-bounced with S/R latches. The step switch uses two NAND gates (IC11), while the pause switch uses a D-type latch (IC12) to accomplish the de-bounce function. When the pause switch is in the RUN position, SPAUSE is a logic ZERO and the $\overline{\text{PAUSE}}$ line is held at logic ZERO (de-activated).

When the pause switch is set to the PAUSE position, SPAUSE is a logic ONE and $\overline{\text{PAUSE}}$ will switch to a logic ONE. When the $\overline{\text{PAUSE}}$ line switches to a logic ONE, the 2650 will finish executing the current instruction, fetch the first byte of the next instruction from memory, and enter the wait state. The $\overline{\text{RUN/WAIT}}$ line goes to a logic ZERO when the 2650 enters the wait state. If the step switch is pushed, LSTEP clocks a logic ONE into the CLSTEP latch (IC12) which sets $\overline{\text{PAUSE}}$ to a logic ZERO. The 2650 then returns to the run mode, and the $\overline{\text{RUN/WAIT}}$ line goes to a logic ONE. When the $\overline{\text{RUN/WAIT}}$ line switches to a logic ONE, the CLSTEP latch is reset and

DS1000 LOGIC DIAGRAM

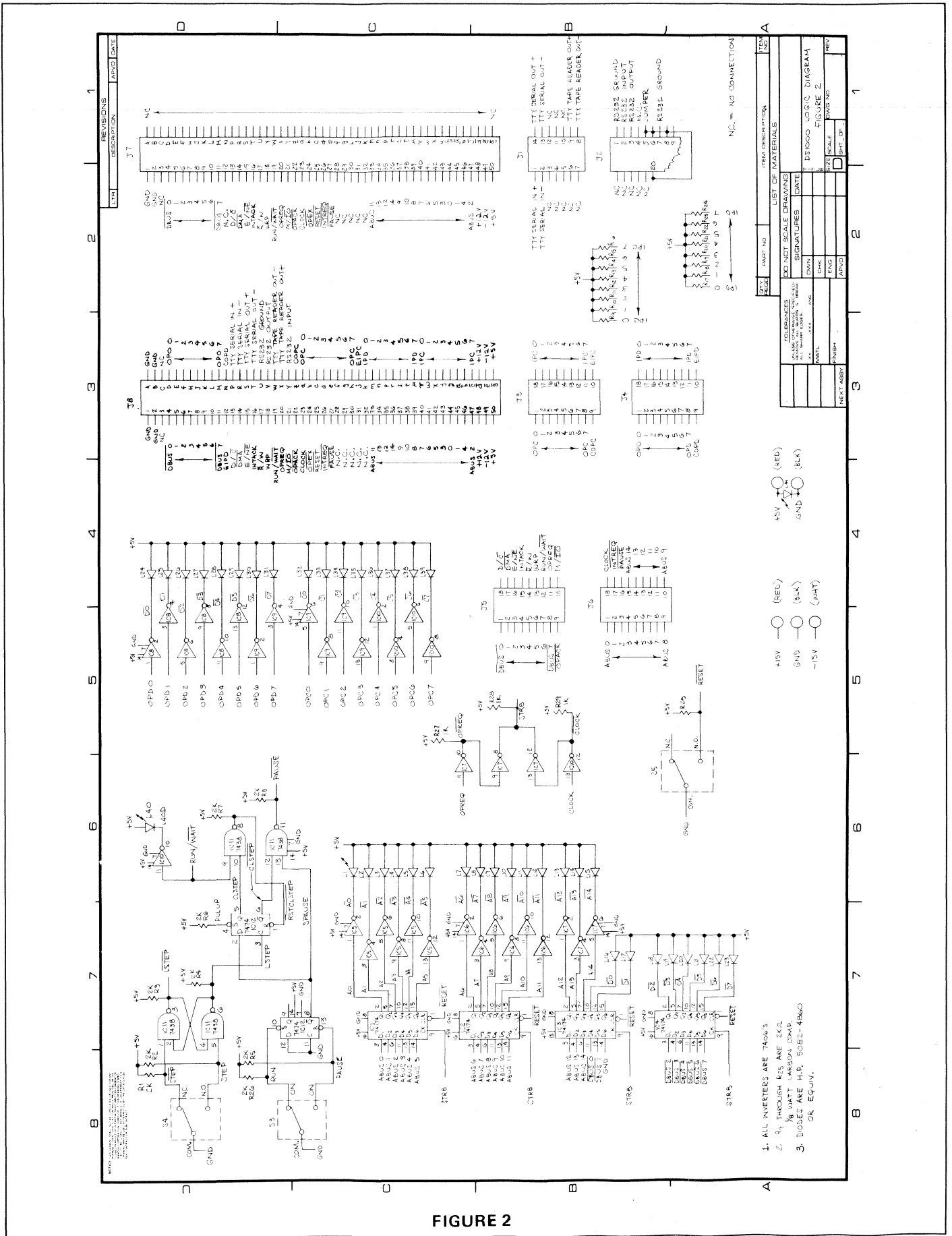


FIGURE 2

$\overline{\text{PAUSE}}$ returns to a logic ONE. This process is repeated once each time the step button is pushed. When the pause switch is returned to the RUN position, the $\overline{\text{PAUSE}}$ line is set to a logic ZERO and the 2650 will return to the run mode. The step/pause function is implemented with IC11 (NAND gate) and IC12 (D-type latch).

OUTPUT CHANNEL DISPLAYS:

The two non-extended output channels implemented on the PC1001 board are displayed on the DS. The output bits, (OPD 0 - OPC 7) are received by open collector inverters which in turn drive the LEDs. A logic ONE output to port 1 (WRTD instruction) will "light" the corresponding OPD LED, while a logic ONE to port 2 (WRTC instruction) will "light" the corresponding OPC LED. Signal OPD 0 corresponds to Output Port 1 bit 0, and OPC 0 corresponds to Output Port 2 bit 0.

RUN AND +5V DISPLAYS:

When the 2650 is in the run mode, the run LED will be "lighted". When +5 volts is applied across the red and black terminals of the DS1000, the +5V LED will be "lighted." When a.c. power is applied to the DS2000 (internal power supply), the +5V LED will be "lighted".

RESET:

The reset switch (S5) pulls the $\overline{\text{RESET}}$ line to a logic ONE when pushed. The $\overline{\text{RESET}}$ line is tied to the corresponding pin on the PC1001 board (pin 25) as well as the ABUS and DBUS latches on the DS.

DEMO SYSTEM PARTS LIST

Item #	Description	ID#	Mfg. and Part #
1.	Base Box	—	—
2.	Printed Circuit Board	—	—
3.	100-Pin Connector	J7, J8	Amphenol, series 225
4.	18-Pin Dip Socket	J3, J4	Cambion
	14-Pin Dip Socket	J6, J6	703-3787-01-04-16
5.	SPDT Push Button Switch	J1	Cambion
			703-4000-01-04-16
6.	SPDT Toggle Switch	S4, S5	Alco, MSP105F
7.	LED	S3	Alco, MTA106D
8.	5-Way Binding Post	L1-L41	H.P.
			5082-4870450
9.	RS232 Connector	J2	H.H. Smith
10.	Carbon Composition Resistors — 2K Ω	R1-R29	TRW Cinch DB25S
11.	Aluminum Standoff		Allan Bradley
			RC05GF202J
12.	Tinnerman Speed Nuts		H.H. Smith 8352
13.			Tinnerman
			C8093-632

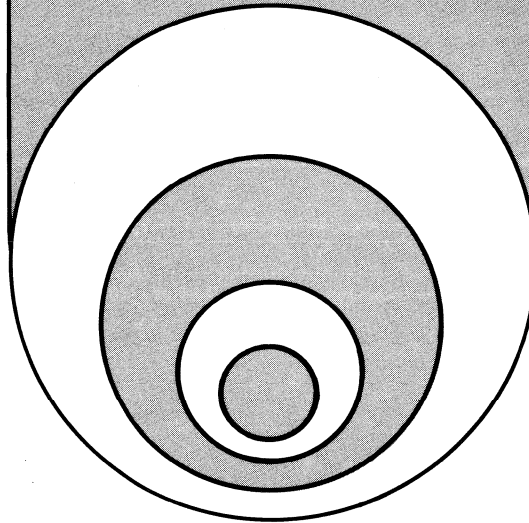
POWER SUPPLY SPECIFICATIONS

(DS1000 Only, Power Supply Included With DS2000)

5 Volt Power Supply	Line Regulation 0.1%
	Load Regulation 0.1%
	Ripple 10m Volts (maximum)
	Response Time 30 usec (maximum)
	Output Current 4 amps (To supply PC1001 only)
	Overvoltage Protection
	Current Overload Protection
± 12 Volt Power Supply	Line Regulation 0.1%
	Load Regulation 0.1%
	Ripple 10m Volts (maximum)
	Response Time 30 usec (maximum)
	Output Current 50 milliamps (To supply PC1001 only)
	Overvoltage Protection
	Current Overload Protection

signetics

MOS
MICROPROCESSOR



SUPPORT SOFTWARE
FOR USE WITH THE
NCSS TIMESHARING
SYSTEMSP52

1. INTRODUCTION

A series of programs is described that provide the microprocessor application's design engineer with on-line support for the development of programs to be run on the Signetics 2650 microprocessor. These programs include a cross-assembler, a cross-simulator, and two utility programs that convert the object file produced by the assembler into either one of two tape formats — one suitable for loading into the 2650 microprocessor and the other suitable for burning PROMs. The programs are accessed through a communications terminal connected to a National CSS Data Center via standard telephone lines.

The first few sections describe the available programs and provide detailed instructions for using them. All available usage options are included as reference information. A final section, called "Operating Instructions," provides the user with step-by-step procedures for generating, editing, assembling, punching, and simulating Signetics 2650 programs. These procedures explain some of the more commonly used features of both the NCSS and the Signetics facilities and demonstrate how to use them.

2. USAGE OVERVIEW

The user creates the source file for his assembly language program by using the EDIT facility available on the NCSS system, or he may have his program punched onto cards and read into the system at a NCSS Data Center. Once the source file resides on the system, the user executes the assembler, which translates symbolic source statements into machine language instructions, and generates both an assembled listing of the source file and an object file. If the assembler reports any errors in the source file, the EDIT facility may again be invoked to correct the source file. The corrected source file is then resubmitted to the assembler. Once the assembler reports no errors, the user may input the object file to the simulator which then simulates execution of the program. The simulator provides the following capabilities:

- 1) Establishes initial program conditions.
- 2) Monitors execution sequences.
- 3) Modifies the program until it operates as desired.

Once the program operates correctly, the user may repeat the entire cycle: correct his source file; reassemble; and test the new program using the simulator. When the program is fully tested and debugged, it may be punched onto tape.

3. EXECUTING 2650 SUPPORT PROGRAMS

A. GENERAL

To execute any of the 2650 support programs, the following command must be entered:

```
ATTACH P2650
```

This causes the P2650 "PROTECT" Exec to execute. It prints:

```
P2650 Attached as XXX, (Y) RUN? >  
P2650 - Version "No." - "Date"  
Run on "DATE"  
ENTER COMMAND (e.g., HELP) >
```

At this point the user may enter any one of the following commands:

HELP	Print Command List
HELP 'NAME'	Print Command in Detail
QUIT	Exit P2650 (Return to VP/CSS)
NEW	Print New Features
PIPHASM	Assemble 2650 Program
PIPSIM	Simulate 2650 Program
PIPTAP	Punch PIPBUG Tape
PIPTAP	Punch PROM Burning Tape

No other CSS command may be executed while under control of the P2650 "PROTECT" Exec; e.g., you cannot edit your file until you exit P2650 by typing "QUIT":

```
ENTER COMMAND > QUIT
```

B. HELP - AN ON-LINE INFORMATION RESOURCE FACILITY

To determine what commands are currently available on P2650, type:

```
HELP
```

To obtain information on how to enter any command except HELP or QUIT, type HELP followed by the name of the desired command; e.g.,

```
HELP PIPHASM
```

A description of the command and its format will be printed.

4. PROGRAM DESCRIPTIONS

A. PIPHASM - SIGNETICS 2650 PIP ASSEMBLER

PIPHASM supports the 2650 assembler languages as specified in the basic manual set (2650 BM 1000). It outputs a hexadecimal object module in a format acceptable to the two tape-punching programs, PIPHTAP and PIPSTAP, and to the simulator, PIPSIM.

Following is the format of the command for executing the assembler:

PIPHASM SOURCE (DISPLAY) (WIDTH)*

where

PIPHASM causes the assembler to execute.

SOURCE is the name of the user's source file. This file has a type of "SYSIN".

DISPLAY is an optional parameter specifying that the listing is to be printed either on the user's console (CON) or on the off-line printer (PTR). If this parameter is missing, CON is assumed.

WIDTH is an optional parameter specifying the line width of the user's console in characters per line—either 80 characters (1) or 120 characters (0). If no parameter is specified, 120 characters per line is assumed. This parameter may be specified only if CON has been specified by DISPLAY.

The object file produced by the assembler will have the same file name as the input file with ".OBJ" concatenated at the end; it will have a filetype of "DATA".

B. SIGNETICS 2650 SIMULATOR

The 2650 simulator, a program written in FORTRAN IV, simulates the execution of a program without using the 2650 processor. The simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the program, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. The user may request traces of the processor status, dumps of the memory contents, and program timing statistics. Multiple simulations of the same program with different parameters may be executed during one simulation run.

The simulator requires as input both the program object module produced by the 2650 assembler and a file of user commands. It produces a listing of user commands, executes the program, and prints ("displays") both static and dynamic information as requested by the user commands. The user may direct the input of the simulator either to a terminal or to a line printer.

PIPSIM SOURCE COMMAND (DISPLAY)

where

PIPSIM causes the simulator to execute.

*Parenthesis indicate an optional parameter with a default value.

SOURCE is the name of the source file originally submitted to the assembler. The simulator concatenates .OBJ onto the name of the source file and uses the designator, SOURCE.O, to find the file containing the object module of the program to be executed. File names are limited to eight characters. This object module is ordinarily produced by the assembler and has a filetype of "DATA."

COMMAND is the name of a file containing the user's commands. This file has a filetype of "DATA."

DISPLAY is an optional parameter specifying the destination of all printed output either to the user's console (CON) or to the off-line printer (PRT). If no parameter is specified, the user's console is assumed.

C. PAPER TAPE UTILITIES

1) PIPHTAP

PIPHTAP punches the "hex" object file onto tape in a format acceptable as input to the 2650 Prototyping Card (2650 PC 1000). See Signetics Applications Memo SS51 for the tape format specification.

The command format for PIPHTAP is:

PIPHTAP SOURCE

where

SOURCE is the name of the source file originally submitted to the assembler.

When "EXECUTION:" is printed, turn the punch on.

2) PIPSTAP

PIPSTAP punches the "hex" object file onto tape in a form suitable for burning PROMs in SMS format. PIPSTAP uses the same command format as PIPHTAP; i.e.,

PIPSTAP SOURCE

where

SOURCE is the name of the source file originally submitted to the assembler.

PIPSTAP responds with a request for the following information:

- The name of your object file.
- The value (two hexadecimal digits) representing the unburned state of your PROM.
- The byte size (four decimal digits) of the PROMs to be burned.
- Up to eight pairs of START/END addresses (four hexadecimal digits). Each address pair identifies an area of code in the object module.

NOTE: All numbers entered *must* contain leading zeros; e.g., when entering the size of a PROM as 256, you must enter 0256.

A START address larger than 7FFF, e.g., 8000, terminates the input mode. Once the input mode is terminated, the punch must be turned on. PIPSTAP punches and prints a record for each PROM specified.

START/END addresses are rounded down/up to the limits of the affected PROM. Thus if:

INITIAL PROM VALUE = FF,
 PROM SIZE = 0256,
 START ADDR = 0040,

and

END ADDR = 0240,

PIPSTAP punches three records: 0000 – 00FF, 0100 – 01FF, and 0200 – 02FF. Each of the records is preceded by its initial address (0000, 0100, and 0200). This initial address is punched into the tape so that it is visible. This enables the tape to be separated into individual strips for each PROM. The areas 0000 – 003F and 0241 – 02FF are filled with FFs.

Each record is punched in exactly the order that its START/END address was entered so that multiple records may be punched for the same PROM. When PIPSTAP stops punching, turn the punch off.

5. OPERATING INSTRUCTIONS FOR USING THE NCSS TIMESHARING SERVICE

A. GENERAL

1. The computer requests the user to type information by printing a > character at the start of a line.
2. The user terminates each line typed with a carriage return.
3. The user deletes (tells the computer to ignore) characters that were erroneously typed by typing the @ character. The computer deletes one preceding character for each @ character typed; e.g., the message LANE@@@INE corrects the word LANE to LINE. The [character deletes all characters previously typed on the line.
4. In all of the following examples, lines typed by the user are underlined to distinguish them from lines printed by the computer.

B. LOGGING IN TO CSS

1. Set the terminal to "LINE" mode.

2. Select the half-duplex mode using the HALF/FULL duplex switch on your terminal (not required on some terminals).
3. Dial the NCSS-supplied telephone number.
4. When you hear a high-pitched tone (indicating that you have established communication with the computer), place the telephone receiver in the modem coupler.
5. Log on by typing an 'S' or a 'O' followed by a carriage return; i.e.,
 - S carriage return (when using a 10 cps terminal)
 - O carriage return (when using a 30 cps terminal)

In response, the system types

CSS ONLINE – XXXX

to signal that you have reached an NCSS monitor. XXXX is the name of the NCSS system with which you have established a connection. The system also types the prompt character >, indicating that it is ready to accept additional input from your terminal. In response, you should type:

> L WEST XXXXXX

where XXXXXX is your user ID number.

The system will respond with

PASSWORD

XXXXXXXXXX

providing a blocked-out area in which you enter your password. Type the password on top of the blocked-out area and press the carriage return.

When the system responds with

A/C INFO:

press the carriage return. (You may optionally enter some accounting information if you desire.)

Messages from the NCSS system are printed here.

CSS.211.data

time>

C. USING THE EDITOR TO CREATE A NEW SOURCE FILE AND/OR TO EDIT AN EXISTING SOURCE FILE

1) Creating a New Program Source File

- a. On NCSS every file has a file name (FN) and a file type (FT). A file name is the unique name to be assigned to your program. Assign your program a file name of 1-to-4 alphanumeric characters beginning with an alphabetic character. The file type of your source program is "SYSIN." The object file created

by the assembler is your unique file name plus the .OBJ appendage. The file type of object files is "DATA."

- b. The timesharing computer stores all source and object files on disk. The user may obtain a directory of the files stored in his user area by typing the letter, L, e.g.,

```
time > L
FILENAME  FILETYPE  MODE  ITEMS
PROG      SYSIN     P     40
PROG.OBJ  DATA     P     5
```

- c. To create a new program source file, the user calls the editor program with an indication of the file name and file type to be created. The editor recognizes that the file name specified is not in the directory and creates a new file.

```
time > E filename SYSIN
NEW FILE.
```

INPUT:

Type your program here. If you make mistakes, use the @ key or finish typing your program and make corrections as specified in step d below. Type an additional carriage return after the last line to exit the new file input mode. The system responds with:

EDIT:

- d. If you wish to edit your program (i.e., correct any typing errors or omissions), proceed to step 2b below. If you do not wish to edit your program at this time, type "FILE" to exit the editor.

2) Editing a Program Source File

- a. The edit mode can be entered directly when the editor is called by specifying a filename-filetype already on disk; e.g., if PROG SYSIN already exists on disk, enter:

```
time > E PROG SYSIN
```

EDIT:

Enter edit commands.

- b. The system's editing capabilities are based on the pointer concept; i.e., any line in a file can be located by an imaginary pointer. This pointer can be moved up or down, positioned at the beginning or end of the file or positioned at a specific line. The position of the pointer determines where the next edit request takes place. The position of the pointer is referred to as the "current line."

- c. Following is a list of some of the most frequently used editing commands: (NOTE: Whenever "n" is indicated in a command, it represents a decimal

number. If "n" is left off the command, the number 1 is assumed.)

>I Moves the pointer to the first line of the file.

>DO n Moves the pointer down n lines and prints the new current line.

>UP n Moves the pointer up n lines and prints the new current line.

> L/string/ Moves the pointer to the next line which contains the character string specified between the slash delimiters. It then prints that line. It does not search the current line for the string. If the character string contains a /, then some other character, such as the \$, may be used as the delimiter.

>P n Print n lines starting with the current line. Also move the pointer to the last line printed. If n = 1 or is absent, the current line is printed and the pointer is not moved.

>DE n Delete n lines starting with the current line.

>R text Replace the entire line following the pointer with the text on the R line. The text is separated from the R by only 1 blank. Any additional spaces are considered part of the text.

>C /string 1/ string 2/ Replace character string 1 in the current line with character string 2. If the / character appears in either of the strings, use some other character, such as the \$, as the string delimiter.

>I INPUT: An I followed by a carriage return puts the editor into input mode. This request is issued to insert lines after the current line. After the "INPUT:" message is printed, the user types one or more lines to be inserted into the program. The last line typed should be followed by two carriage returns to return to EDIT mode. The pointer is moved to point to the last line inserted.

- d. Error Messages

Editor error messages are as follows:

? Invalid edit request.

EOF: The end of file is reached by an edit request. The request is terminated, and the pointer is positioned after the last line of the file.

TRUNCATED The following line was truncated as shown. Only 72 character lines are permitted.

e. Exiting the Editor

To exit the editor and save your new file, type:

>FILE

To exit the editor without changing your original file, type:

>QUIT

D. ASSEMBLING THE PROGRAM SOURCE FILE TO CREATE A HEXADECIMAL FORMAT OBJECT FILE FOR PROGRAM SIMULATION AND FOR PROGRAM DEBUGGING ON THE PROTOTYPING SYSTEM

```
time > ATTACH P2650
P2650 ATTACHED AS 192, (T)
P2650 - Version 2.0 - 1/5/76
RUN ON 'DATE'
P2650 COMMAND (e.g., HELP) > PIPHASM filename
P2650 ASSEMBLER ...
RUN ... (YES OR NO)? > YES
EXECUTION:
```

(Your assembly listing will be printed here. Be patient—there may be a short delay before printing starts.)

TOTAL ASSEMBLER ERRORS = X

E. LOGGING OFF NCSS

Exit P2650, log off the NCSS system, and review your program for logical and syntactical errors.

```
ENTER COMMAND > QUIT
time > LOGOUT
XXXX VPU'S,XX CONNECT HRS,XX I/O
LOGGED OFF AT time ON date
```

F. CHECKING OUT YOUR PROGRAM USING THE SIMULATOR

1. Log on the system as described in step B.
2. Using the editor program, create a file containing the simulator commands. This file is of type DATA. The file name may be the same as the source file name but with a .TST appendage:

```
time > E filename.TST DATA
NEW FILE
```

INPUT:

NOTE: The directions for using the editor described in steps C.1 and C.2 apply here also.

Enter commands here.

EDIT:

>FILE

3. Request a simulator run.


```
time > ATTACH P2650
P2650 ATTACHED AS 192, (T)
P2650 - Version 2.0 - 1/5/76
RUN ON 'DATE'
P2650 COMMAND; e.g., 'HELP' > PIPSIM filename
filename.TST
P2650 SIMULATOR ...
RUN ... (YES OR NO)? > YES
EXECUTION:
```

The simulator listing is printed here.

G. LOGGING OFF

Exit P2650, log off the NCSS system, and review the simulator listing to determine program correctness.

```
P2650 COMMAND > QUIT
time > LOGOUT
XXXX VPU'S,XX CONNECT HRS,XX I/O
LOGGED OFF AT time ON date.
```

H. PUNCHING A PAPER TAPE FOR DEBUGGING ON THE PROTOTYPE CARD SYSTEM

Check to ensure that the punch is off. After the "EXECUTION:" message is printed by the computer, turn the punch on. Turn the punch off after it stops punching.

```
P2650 COMMAND > PIPHTAP filename
```

(NOTE: Do not use the .OBJ extension on the filename. The punch program assumes this is the .OBJ file and automatically adds this extension.)

EXECUTION:

.OBJ file will be listed here.

```
P2650 COMMAND > QUIT
```

Log off the system as in step G above.

I. PUNCHING A PAPER TAPE FOR BURNING PROMS

Check to see that the punch is off, and log off the system using the procedures outlined in step B.

Execute PIPSTAP:

```
P2650 COMMAND (e.g., HELP) > PIPSTAP filename
P2650 PIPSTAP ...
RUN ... (YES OR NO)? > YES
```

PIPSTAP responds with a request for the unburned state of your PROM. Since PIPSTAP punches data into each location of the PROM, if your object module does not fill the entire PROM, PIPSTAP requires a value that can be used for the other locations. This value must be entered as two hexadecimal digits:

INITIAL PROM VALUE? 00

PIPSTAP then asks for the size (in bytes) of your PROM, which must be entered in four decimal digits. The maximum allowable size is 1024.

PROM SIZE? 0256

PIPSTAP requests both a START and an END address for the code to be punched. Use four hexadecimal digits for each address as shown below. Don't forget the leading zeros.

START ADDR? 0000
END ADDR? 000A

PIPSTAP will request up to eight pairs of START/END addresses. Enter a number larger than 7FFF, e.g., 8000, when you have completely described the object module:

START ADDR? 8000

When you press

Carriage Return

PIPSTAP punches 50 frames of leader followed by the PROM record specified by your START and END addresses. The START address of your PROM, 0000, is punched into the tape so that it is visible.

When punching is complete, turn the punch off and log off the system.

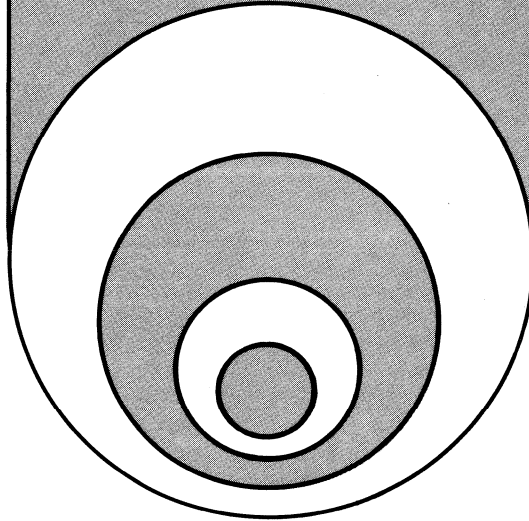
6. REFERENCE DOCUMENTS

For additional information, consult the following manuals:

- Signetics 2650 Microprocessor Manual (2650BM 1000)
- VP/CSS Reference Manual (Form 106-3, available from NCSS)
- VP/CSS Edit Command (Form 108-4, available from NCSS)

signetics

**MOS
MICROPROCESSOR**



**SUPPORT SOFTWARE
FOR USE WITH GE'S
MARK III TIMESHARING
SYSTEM SP54**

1. SUMMARY

A series of programs is described that provide the micro-processor application's design engineer with on-line support for the development of programs to be run on the Signetics 2650 microprocessor. These programs include a cross-assembler, a cross-simulator, and two tape utility programs that convert the object file produced by the assembler into either a "hex" format, suitable for loading into system memory by "PIPBUG," or into a format suitable for burning PROMs. The programs are accessed through a communications terminal connected to General Electric's Mark III Timesharing System via standard telephone lines.

2. USAGE OVERVIEW

The user creates the source file for his assembly language program by using the editing facility or his program may be punched onto cards and read into the system. Once the source file resides in the system, the user executes the assembler, which translates symbolic source statements into machine language instructions, and generates both an assembled listing of the source file and an object file. If the assembler reports any errors in the source file, the user may again invoke the editing facility to correct the errors. The corrected source file is then resubmitted to the assembler. Once the assembler reports no errors, the user may input the object file to the simulator which simulates execution of the program.

The simulator provides the following capabilities:

- 1) Establishes initial program conditions.
- 2) Monitors execution sequences.
- 3) Modifies the program until it operates as desired.

Once the program operates correctly, the user may repeat the entire cycle: correct his source file, reassemble, and test the new program using the simulator. When the program is fully tested and debugged, it may be punched onto tape in a format for loading into system memory and/or for burning PROMs.

3. PROGRAM DESCRIPTIONS

The next few sections describe the available programs and provide detailed instructions for using them. All available usage options are included as reference information. A final section, called "Operating Instructions," provides step-

by-step procedures for generating, editing, assembling, simulating, and punching Signetics 2650 programs. These procedures explain some of the more commonly used features of both the General Electric Timesharing System and the Signetics facilities and demonstrate how to use them.

A. PIPHASM – SIGNETICS 2650 PIP ASSEMBLER (HEX TAPE FORMAT)

PIPHASM supports the 2650 assembler language as specified in the basic manual set (2650 BM 1000). It outputs a hexadecimal object module in a format acceptable to the two tape-punching programs, PIPHTAP and PIPSTAP, and to the simulator, PIPSIM.

To execute the assembler, enter the command:

```
/PIPHASM
```

The assembler will start executing and will request the following information:

- The name of the input (source) file.
- The name assigned to the assembler-produced object file. It is suggested that some naming convention be adopted; e.g., always name the object file with the first four letters from the name of the source file followed by ".OBJ".
- The width of your terminal carriage. Enter "0" if your terminal carriage has 120 characters; otherwise, enter "1".

To assemble your program, the assembler creates a scratch file on your user ID. If the assembly runs to completion, this file will be purged. But if the assembly is aborted, the file may remain on your user ID. You may collect up to ten of these scratch files before the assembler will be unable to assemble because it cannot find a scratch file name. The scratch file names that must be purged are referred to as: A 00, A 01, , A 09.

B. SIGNETICS 2650 SIMULATOR

The 2650 simulator, a program written in FORTRAN IV, simulates the execution of a 2650 program without using the 2650 processor. The simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the 2650 program, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. The user may

request traces of the processor status, dumps of the contents of memory, and program timing statistics. Multiple simulations of the same program with different parameters may be executed during one simulation run.

The simulator requires as input both the program object module produced by the 2650 assembler and a file of user commands. It produces a listing of the user's commands, executes the program, and prints ("displays") both static and dynamic information as requested by the user's commands.

The Signetics Basic Manual Set (2650 BM 1000) contains a description of the user commands and the general operation of the simulator.

To execute the simulator, enter the command:

```
/PIPSIM
```

The simulator starts executing and requests the following information:

- The name of the object module produced by the assembler for your program.
- The name of the file of simulator commands.

C. PAPER TAPE UTILITIES

The two paper tape utility programs, PIPHTAP and PIPSTAP, complete the series of programs discussed in this memo.

1) PIPHTAP

PIPHTAP punches the "hex" object file onto tape in a format acceptable as input to the 2650 Prototyping Card (2650 PC 1001). Refer to Signetics Applications Memo SS51 for the tape format specifications.

To execute PIPHTAP, enter the command:

```
/PIPHTAP
```

PIPHTAP responds with a request for the name of your object (input) file; it then requests that the punch be turned on and that the carriage return key be depressed. PIPHTAP punches about 50 frames of leader before it punches the object module. When the system responds with "READY," turn the punch off.

2) PIPSTAP

PIPSTAP punches the "hex" object file onto tape in a form suitable for burning a PROM. To execute PIPSTAP, enter the following command:

```
/PIPSTAP
```

PIPSTAP responds with a request for the following information:

- The name of the object file.
- The value (two hexadecimal digits) representing the unburned state of your PROM.
- The size in bytes (four decimal digits) of the PROMs to be burned.
- Up to eight pairs of START/END addresses (four hexadecimal digits). Each address pair identifies an area of code in the object module.

NOTE: All numbers entered must contain leading zeros; e.g., when entering the size of a PROM as 256, you must enter 0256.

A START address larger than 7FFF, e.g., 8000, terminates the input mode.

Once the input mode is terminated, PIPSTAP requests that the punch be turned on. It then punches and prints a record for each PROM specified.

START/END addresses are rounded down/up to the limits of the affected PROM. Thus if:

```
INITIAL PROM VALUE = FF,
PROM SIZE           = 0256,
START ADDR          = 0040
```

and

```
END ADDR           = 0240,
```

PIPSTAP punches three records: 0000 - 00FF, 0100 - 01FF, and 0200 - 02FF. Each of the records is preceded by its initial address (0000, 0100, 0200) punched into the tape so that it is visible. This enables the tape to be separated into individual strips for each PROM. The areas 0000 - 003F and 0241 - 02FF are filled with FFs.

Each record is punched in exactly the order in which its START/END address was entered so that multiple records may be punched for the same PROM. When the system types "READY," turn the punch off.

4. OPERATING INSTRUCTIONS

This section provides a synopsis of operating instructions for using the GE Mark III Timesharing Service to generate, edit, assemble, simulate, and punch Signetics 2650 programs. For more detailed information on the capabilities of the GE Mark III Timesharing Service, refer to the following manuals available from General Electric's Information Services Business Division:

1) *Command System* — Mark III Foreground Reference Manual No. 3501.01J.

2) *Editing Commands* — Mark III Foreground Reference Manual No. 3400.01F.

When using high-speed terminals (120 cps and up) or in the event of any difficulty, contact your local General

Electric Sales Office. A list of General Electric Sales Offices is provided at the end of this document.

A. LOGGING IN

- Set the terminal to "LINE" mode.
- Select the half-duplex mode, using the HALF/FULL duplex switch (if necessary).
- When you hear the high-pitched tone (indicating that you have established communication with the computer), place the telephone receiver in the modem coupler.

NOTE: In the following examples data typed by the user is underlined to distinguish it from data printed by the computer.

Log in as follows

H carriage return

Depressing the carriage return key terminates all input lines. Some General Electric personnel recommend that four Hs, HHHH, be entered instead of one. The timesharing system determines the speed of your terminal from the speed at which these characters are received

The computer will respond to your H carriage return entry with

U#=-

At this point enter your user ID (3 alphabetic characters and 5 numeric characters) and press the carriage return:

U#=AAANNNNN

The system responds

PASSWORD

XXXXXXXX

providing a blocked-out area in which you may enter your password. Type the password on top of the blocked-out area and press the carriage return. At this point the system may send an informative message to your terminal. Some user IDs are equipped with a short log-on sequence. If this is true, the system responds with

READY

If this is not true, the system responds with

ID:

This is a request for accounting information. If you do not wish to enter any accounting information, simply press the carriage return:

ID: carriage return

The computer will respond with:

SYSTEM:

Specify FORTRAN IV

SYSTEM: FIV

since both the assembler and the simulator are written in FORTRAN IV. The system will respond with:

NEW OR OLD

This is the same as the READY message. The system is now ready to perform any task you request.

B. ERROR RECOVERY

Prior to issuing any commands, it is essential to know how to delete an unwanted command.

- *Character Delete:* To delete the last character typed, hold down the shift key and depress zero (0) (ASCII decimal code 95). The ASCII decimal code is included since the actual key used may differ from terminal to terminal.
- *Line Delete:* To abort a line before the carriage return key is depressed, hold down the control key and depress "X" (ASCII decimal code 24).
- *Break:* To abort a command while it is being executed (e.g., stop printing a long file), depress the BREAK or interrupt key twice.

C. CREATING AND/OR EDITING A SOURCE FILE

Both the assembler and the simulator expect you to identify a source file that you have created. The assembler expects the 2650 program source file and the simulator expects the user's command source file. To create the source file, the name of the file must be specified:

NEW FILENAME

This command assigns the name, FILENAME, to the temporary working file. At this point, the file is empty. Notice that the file name, FILENAME, is eight characters long. We recommend that the first four characters be meaningful. Acceptable file names are 1-to-8 characters long using only the letters A through Z, numerals 0 through 9, and the period (.).

At this point enter each line of the source file into the temporary buffer:

```
100 *PROCESSOR SYMBOLS
110 R0 EQU 0
120 R3 EQU 3
130 *PROGRAM VARIABLE STORAGE
140 ORG H'100'
150 TLEN EQU 3 TABLE LENGTH
160 TBLA RES TLEN TABLE A
170 TBLB RES TLEN TABLE B
180 *MOVE DATA IN TBLA TO TBLB. TLEN MUST
185 *BE LESS THAN 256 BYTES
```

```

190      ORG      0
200      LODI,R3  TLEN
210 LOOP  LODA,R0  TBA-1,R3
220      STRA,R0  TBB-1,R3
226      NOP
228      NOP
230      HALT
240      END
    
```

Note that each line starts with a line number followed by a space and then the source data itself. Lines may be entered out of order, since the system will sort the source lines by line number. Once the data is entered, this temporary file must be saved in permanent storage using the following command:

SAVE

The system responds with a READY message, and the temporary file remains intact.

To list the contents of your temporary file, type:

LIST

The system responds by printing your file.

Should you want to change your source file, bear in mind that the only file that can be modified (or edited) is the temporary working file. At this point your source program still resides in the working file; however, if your source program resided in a permanent rather than a working file, enter the following command:

OLD FILENAME

The OLD command reads the contents of the permanent file, named FILENAME, into the temporary file and assigns the name, FILENAME, to the temporary file.

The source file is now ready for editing.

To add a line, simply type the line with a new line number:

225 BDRR,R1 LOOP

To change a line, retype the line using the same line number:

225 BDRR,R3 LOOP

To change all occurrences of the letters "TB" to "TBL" from lines 210 through line 220, enter the following command:

CHAVC 210/TB/TBL/220

This command changes the following two source lines:

```

210 LODA,R0 TBLA-1,R3
220 STRA,R0 TBLB-1,R3
READY
    
```

Lines 226 and 228 may be deleted with either one of the following two commands:

EDI DEL 226-228

or

EDI DEL 226,228

The first command deletes lines 226 through 228, while the second command deletes lines 226 and 228.

List your temporary file and verify all changes:

LIST

The system prints your file here and then prints:

READY

Save your file in the permanent file that was created with the SAVE command:

REPLACE

READY

The SAVE command creates a permanent file with the same name as the one assigned to the temporary file. The REPLACE command takes the content of the temporary file and stores it in the already existing permanent file that has the same file name.

NOTE: Most system commands may be shortened to the first three letters; e.g., REPLACE = REP.

D. ASSEMBLING THE PROGRAM TO CREATE AN OBJECT MODULE

The editing facility assumes that each line of your source program has a line number at the beginning. Since neither the assembler nor the simulator will accept these line numbers, the following command must be executed to remove them:

EDI DES FILENAME

READY

The assembler is now ready to be executed. Enter the command:

/PIPHASM

The assembler responds with a request for the name of your source program:

INPUT FILENAME? FILENAME

The assembler then requests the name of your object module:

OBJECT FILENAME? FILE.OBJ

This is a file that the assembler generates. Your file must be assigned a name. One useful technique is to use the first four letters of the name of the source program with .OBJ concatenated onto the end.

The computer prints:

TYPE '0' FOR WIDE CARRIAGE or
TYPE '1' FOR NARROW CARRIAGE 1

If your terminal prints 120 characters per line, type '0'.
If your terminal prints less than 120 characters per line,
type '1'.

The assembler responds by printing your listing. When the
listing is complete, the system prints:

READY

Now that your listing is complete, you may restore the line
numbers to your file by entering the following command.
This is only necessary if you plan to edit your file.

EDI RES FILENAME

E. LOGGING OFF

Log off the GE Timesharing System and review your pro-
gram for logical and syntactical errors.

BYE
00024.11 CRU 0000.41 TCH 0009.74 KC
OFF AT 16:20PDT 10/15/75

F. USING THE SIMULATOR TO TEST AND DEBUG YOUR PROGRAM

1. Log onto the system using the procedures outlined in
step A.
2. Create a file containing the simulator commands. As
with the object module, you could name this file by
concatenating .TST onto the first four letters of FILE-
NAME.

NEW FILE.TST
READY
100 PATCH 100,01 101,02 103,03
120 DUMP A, 100 - 105
130 FEND
SAVE

3. Request a simulator run.

First, you must remove the line numbers from the
command file:

EDI DES FILE.TST
READY
REP
READY

Then execute the simulator by entering the following
command:

/PIPSIM

The simulator responds with a request for the following
information:

OBJECT MODULE NAME? FILE.OBJ

Enter the name of the object module generated by the
assembler.

COMMAND FILE NAME? FILE.TST

Enter the name of the simulator command file.

The simulator prints its output at this time.

*Log off the General Electric Timesharing system and
review the simulator listing to determine if any program
corrections are required.*

BYE

G. PUNCHING A PAPER TAPE FOR DEBUGGING ON THE PROTOTYPE CARD SYSTEM

Check to see that the punch is off, and log onto the system
using the procedures outlined in step A.

When the system responds with

READY

enter the command:

/PIPHTAP

PIPHTAP responds with a request for the name of your
input file:

ENTER INPUT FILE NAME? FILE.OBJ

When the input file name is entered, PIPHTAP prints the
following instructional message:

TURN ON PUNCH AND HIT CARRIAGE RETURN.

When the carriage return key is depressed, PIPHTAP punches
50 frames of leader and then punches your object module.
The object module is also printed.

When punching is complete, the system responds with

READY

Turn the punch off, and log off the system.

H. PUNCHING A PAPER TAPE FOR BURNING PROMS

Check to see that the punch is off, and log onto the system
using the procedures outlined in step A.

When the system responds with

READY

enter the command:

/PIPSTAP

PIPSTAP responds with a request for the name of your input file:

ENTER OBJECT FILE NAME? FILE.OBJ

PIPSTAP then requests that you enter the unburned state of your PROM. (Since PIPSTAP punches data into each location of the PROM, PIPSTAP requires a value that can be used for the other locations):

INITIAL PROM VALUE? 00

This value must be entered as two hexadecimal digits.

PIPSTAP then asks for the size of your PROM (in bytes) which must be entered in four decimal digits. The maximum allowable size is 1024.

PROM SIZE? 0256

PIPSTAP requests both a START and an END address for the code you want punched. Use four hexadecimal digits for each address as shown below. Don't forget the leading zeros.

START ADDR? 0000

END ADDR? 000A

PIPSTAP will request up to eight pairs of START/END addresses. Enter a number larger than 7FFF, e.g., 8000, when you have completely described the object module:

START ADDR? 8000

PIPSTAP prints the following message:

TURN ON PUNCH AND HIT CARRIAGE RETURN

When you press

Carriage Return

PIPSTAP punches 50 frames of leader followed by the PROM record specified by your START and END addresses. The START address of your PROM, 0000, is punched into the tape so that it is visible. Part of the object module will be printed.

When punching is complete, the system responds with:

READY

Turn the punch off, and log off the system.

5. GENERAL ELECTRIC SALES OFFICES

CHICAGO

233 South Wacker Drive
Chicago, Illinois 60666
(312) 781-7840

INDIANAPOLIS

Castleview Building
8000 Knue Road
Indianapolis, Indiana 46250
(317) 842-0100

NEW YORK INDUSTRIAL

Mc-Graw Hill Building
1221 Avenue of the Americas
New York, New York 10020
(212) 997-0351

DETROIT

22150 Greenfield Road
Oak Park, Michigan 48237
(313) 968-8100

FT. WAYNE

Lakeside II Building
2250 Lake Avenue
Ft. Wayne, Indiana 46805
(219) 423-1406

LONG ISLAND

1 Huntington Quadrangle
Huntington Station
L. I. New York 11746
(516) 694-7636

MINNEAPOLIS

1500 Lilac Drive, South
Minneapolis, Minnesota 55416
(612) 546-0990

CLEVELAND

1000 Lakeside Avenue, N.E.
Cleveland, Ohio 44114
(216) 523-6251

EAST ORANGE TELEPHONE BRANCH

33 Evergreen Place
East Orange, New Jersey 07018
(201) 672-0700

MILWAUKEE

615 East Michigan Street
Milwaukee, Wisconsin 53202
(414) 271-7900

COLUMBUS

Harrington Building
90 E. Wilson Bridge Road
Worthington, Ohio 43085
(614) 438-2170

PHILADELPHIA

1700 Market Street
Philadelphia, Pennsylvania 19103
(215) 864-7474

CINCINNATI

580 Walnut Street
Cincinnati, Ohio 45202
(513) 559-3660

PITTSBURGH

Two Gateway Center
Pittsburgh, Pennsylvania 15222
(412) 566-4330

HARRISBURG

3800 Market Street
Camp Hill, Pennsylvania 17011
(717) 761-1481

LOUISVILLE

Citizens Plaza
Louisville, Kentucky 40202
(502) 452-4211

NEW YORK FINANCIAL

Mc-Graw Hill Building
1221 Avenue of the Americas
New York, New York 10020
(212) 997-0317

SCHENECTADY

650 Granklin Street, 3rd Floor
Schenectady, New York 12305
(518) 372-6436

PITTSFIELD

395 Main Street
Dalton, Massachusetts
(413) 494-4308

BOSTON

98 Galen Street
Watertown, Massachusetts 02172
(617) 926-2911

BUFFALO

3980 Sheridan Drive
Buffalo, New York 14226
(716) 839-5222

SYRACUSE

202 Twin Oaks Drive
Syracuse, New York 13206
(315) 456-1995

ROCHESTER

One Marine Midland Plaza
Rochester, New York 14604
(716) 232-6523

STAMFORD

2777 Summer Street
Stamford, Connecticut 05905
(203) 359-2985

HARTFORD

111 Founders Plaza
East Hartford, Ct. 06108
(203) 289-7941

LOS ANGELES NORTH

3550 Wilshire Blvd.
Los Angeles, California 90010
(213) 388-9626

SAN FRANCISCO TELCO BRANCH

One Embarcadero Center
San Francisco, California 94111
(415) 781-1155

SEATTLE

1218 Bank of California Center
Seattle, Washington 98164
(206) 575-2990

PORTLAND

2154 N. E. Broadway
Portland, Oregon 97232
(503) 288-6916

SAN FRANCISCO

One Embarcadero Center
San Francisco, California 94111
(415) 989-1100

PALO ALTO BRANCH

1120 San Antonio Road
Palo Alto, California 94303
(415) 969-3772

LOS ANGELES SOUTH

3550 Wilshire Boulevard
Los Angeles, California 90010
(213) 385-9411

ATLANTA

2200 Century Parkway, N. E.
Atlanta, Georgia 30345
(404) 325-9889

BIRMINGHAM

300 Office Park Drive
Birmingham, Alabama 35223
(205) 879-1298

NASHVILLE

293 Plus Park Boulevard
Nashville, Tennessee 37217
(615) 259-4570

CHARLOTTE

301 S. McDowel Street
Charlotte, North Carolina 28204
(704) 374-1783

GREENSBORO

604 Green Valley Road
Greensboro, N. C. 27408
(919) 292-7230

GREENVILLE

252 South Pleasantburg Drive
Greenville, S. C. 29607
(803) 233-5335

MIAMI

8410 N.W. 53rd Terrace
Miami, Florida 33166
(305) 592-7610

TAMPA

5420 Bay Center Drive
Tampa, Florida 33609
(813) 877-8294

BETHESDA

4720 Montgomery Lane
Bethesda, Maryland 20014
(301) 654-7061

BALTIMORE

25 South Charles Street
Baltimore, Maryland 21201
(301) 539-6770

RICHMOND

Willow Oaks Office Building
6767 Forest Hill Avenue
Richmond, Virginia 23235
(804) 320-0192

WASHINGTON

777 - 14th Street, N.W.
Washington, D.C. 20005
(202) 628-4000

ST. LOUIS

1015 Locust Street
St. Louis, Missouri 63101
(314) 342-7780

KANSAS CITY

911 Commerce Tower
Kansas City, Missouri 64199
(816) 842-9745

DALLAS

1341 West Mockingbird Lane
917 East Tower
Dallas, Texas 75247
(214) 631-0910

SHREVEPORT

208-A Beck Building
Shreveport, Louisiana 71102
(318) 425-2476

HOUSTON

601 Jefferson
Houston, Texas 77002
(713) 224-8294

DENVER

201 University Boulevard
Denver, Colorado 80206
(303) 320-3174

PHOENIX

3225 North Central Avenue
Phoenix, Arizona 85004
(602) 264-7881

TULSA

1900 Fourth National Bank Building
Tulsa, Oklahoma 74119
(918) 582-0800

OKLAHOMA CITY

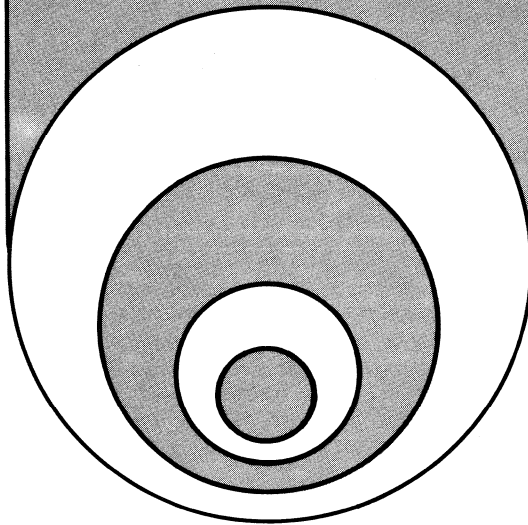
5700 North Portland
Oklahoma City, Oklahoma 73112
(405) 947-2376

© N.V. Philips' Gloeilampenfabrieken

This information is furnished for guidance, and with no guarantees as to its accuracy or completeness; its publication conveys no licence under any patent or other right, nor does the publisher assume liability for any consequence of its use; specifications and availability of goods mentioned in it are subject to change without notice; it is not to be reproduced in any way, in whole or in part, without the written consent of the publisher

signetics

**MOS
MICROPROCESSOR**



**ABSOLUTE
OBJECT
FORMAT
SS51**

(REVISION NO. 1)

INTRODUCTION

The format for absolute code produced for the 2650 is described in this application note.

The absolute object code is formatted into blocks. The first character of every block is a colon. Inside of a block, all the characters are hexadecimal, i.e., 0 to 9 or A to F, inclusive. Only non-printing ASCII control characters may occur within an interblock gap. These are the characters in the first two columns (columns 0 and 1) of the ASCII standard code table. A CR/LF is used within the interblock gap to reset the TTY or terminal after each block.

Each block is independent. For example, paper tape can be positioned prior to any block and a load started. The loading of absolute object code will be halted by:

- A BCC error on the address + count fields
- A BCC error on the data field
- An incorrect block length
- A non-hex character within the block

The block length field contains the number of bytes of actual data which is half the number of hex characters in the data field. While the size of the data field can range from 2 to 510 characters, a standard size of 60 characters has been established so that the tape may be easily generated and read on a variety of terminals and systems. A block length of zero indicates an End of File (EOF) block. The address field of an EOF block contains the start address of the loaded program.

The Block Control Character is 8 bits formed from the actual bytes and not from the ASCII characters. The bytes

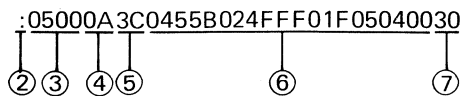
are in turn exclusive or'ed to the BCC byte, and then the BCC byte is left rotated one bit. It appears as two hex characters. Both the address and count fields and the data field are followed by a BCC character pair. The BCC prevents storing data at an invalid memory address or storing bad data into memory.

EXAMPLE: An object tape that loads ten bytes starting at location 500
:05000A3C0455B024FFF01F05040030
:000000

FORMAT

1. Interblock gap of any non-printing characters including spaces
2. Start of block character; a colon
3. Address field; four hex characters
4. Count field; two hex characters in range 0 to 1E
5. BCC for address and count fields; two hex characters
6. Data field; twice the value in the count field which is the number of memory locations loaded by the current block
7. BCC for the data field; two hex characters

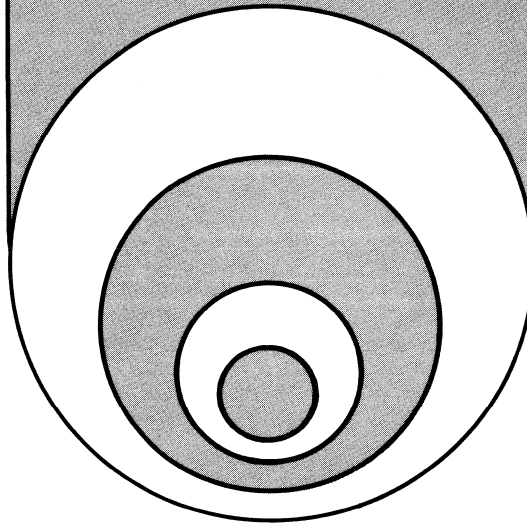
EXAMPLE OF OBJECT FORMAT



- 2 – Start of block character (colon)
- 3 – Starting address for block (H'0500')
- 4 – Number of bytes in block (H'0A' = 10)
- 5 – BCC byte for fields 3 and 4 (H'3C')
- 6 – Data, two characters per byte
- 7 – BCC byte for field 6 (H'30')

signetics

MOS
MICROPROCESSOR



LOW COST CLOCK GENERATOR CIRCUITS MP52

2650 MICROPROCESSOR APPLICATIONS MEMO

GENERAL

The clock circuit requirements for microprocessors range from tightly specified, two-phase, non-overlapping types to simple single-phase, TTL compatible types. To lower system cost, the Signetics 2650 Microprocessor was designed to operate with a single-phase, TTL-level clock without any special clock driver circuitry. The clock input specifications for the 2650 are summarized in Table I.

This Applications Memo describes several clock generator circuits that may be used with the 2650. These circuits use standard TTL logic elements (7400 series). They include RC, LC, and crystal oscillator type circuits.

The stability required by the user's application will determine the type of clock generator that should be used. Tables showing the measured frequencies at several temperatures and supply voltages are presented.

RC OSCILLATOR

A circuit diagram of an RC oscillator is given in Figure 1.

The first inverter is biased into its linear region by resistor R. The positive feedback capacitor (C) from node (B) to node (A) causes the circuit to oscillate. The third inverter acts as a buffer to drive the clock input of the 2650. The oscillation period is approximately equal to 3 RC. Measurements taken on this circuit showed a 10 ns rise time and a 7 ns fall time.

Table II shows how the frequency of the RC oscillator is affected by variations in V_{CC} and ambient temperature.

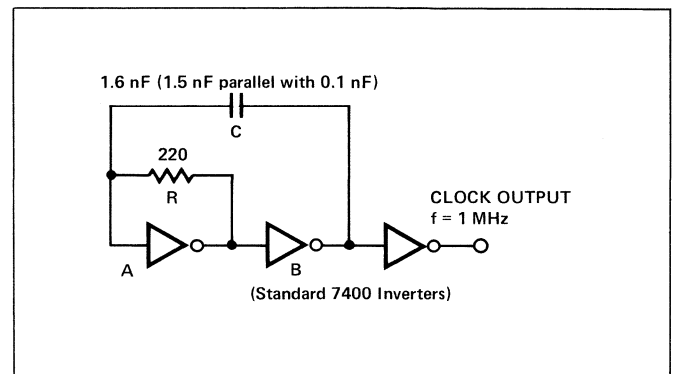


FIGURE 1. RC Clock Generator

TABLE I
2650 CLOCK INPUT SPECIFICATIONS

SYMBOL	PARAMETER	TEST CONDITIONS	LIMITS		UNIT
			MIN.	MAX.	
I_{LI}	Input Load Current	$V_{IN} = 0$ to 5.25V		10	μA
V_{IL}	Input Low Voltage		-0.6	0.8	V
V_{IH}	Input High Voltage		2.2	V_{CC}	V
C_{IN}	Input Capacitance	$V_{IN} = 0V$		10	pF
t_{CH}	Clock High Phase		400	10,000	nsec
t_{CL}	Clock Low Phase		400	∞	nsec
t_{CP}	Clock Period		800	∞	nsec
t_r	Clock Rise Time			20	nsec
t_f	Clock Fall Time			20	nsec

Timing Reference = 1.5V
 $T_A = 0^\circ$ to $70^\circ C$
 $V_{CC} = 5V \pm 5\%$

TABLE II
RC OSCILLATOR STABILITY
Ambient Temperature (T_A)

	0°C	25°C	70°C	Stability _T * (V _{CC} = constant)
V _{CC} = 4.75V	1044.50 KHz	1028.95 KHz	998.50 KHz	+1.51%, -2.96%
V _{CC} = 5.0V	1043.20 KHz	1023.65 KHz	990.45 KHz	+1.91%, -3.24%
V _{CC} = 5.25V	1038.80 KHz	1013.63 KHz	979.65 KHz	+2.48%, -3.35%
Stability _V ** (T _A = constant)	+0.12% -0.42%	+0.52% -0.98%	+0.20% -1.1%	

*Stability_T with respect to T_A = 25°C
**Stability_V with respect to V_{CC} = 5.0V

A second type of RC oscillator uses a monostable multi-vibrator circuit (N74123) as illustrated in Figure 2. The pulse width of each monostable is determined by the external resistor and capacitor:

$$t_w = (0.28) (R_{ext}) (C_{ext}) \left(1 + \frac{0.7}{R_{ext}}\right)$$

where

R_{ext} is in KΩ

C_{ext} is in pF,

and

t_w is in ns.

In this circuit, the oscillation is caused by the triggering of each monostable by the other one. The oscillation frequency can be derived from the following equation:

$$f_{osc} = \frac{1}{t_{w1} + t_{w2}}$$

where:

t_{w1} is the pulse width of the first monostable, and
t_{w2} is the pulse width of the second monostable.

Measurements on frequency stability with a load of one TTL input are presented in Table III.

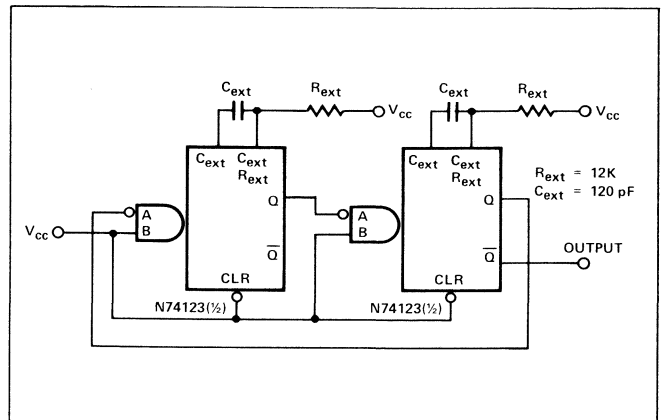


FIGURE 2. RC Clock Generator with Monostable Circuit N74123

TABLE III
MONOSTABLE MULTIVIBRATOR OSCILLATOR
STABILITY

Ambient Temperature (T_A)

	0°C	25°C	70°C	Stability _T * (V _{CC} = constant)
V _{CC} = 4.75V	1063.65 KHz	1046.72 KHz	1041.16 KHz	+1.62%, -0.53%
V _{CC} = 5.0V	1063.80 KHz	1042.83 KHz	1032.63 KHz	+2.01%, -0.98%
V _{CC} = 5.25V	1063.80 KHz	1039.95 KHz	1024.02 KHz	+2.29%, -1.53%
Stability _V ** (T _A = constant)	+0.00% -0.014%	+0.276% -0.373%	+0.826% -0.833%	

*Stability_T with respect to T_A = 25°C
**Stability_V with respect to V_{CC} = 5.0V

The observed rise and fall times at the output of this circuit were 10 ns and 8 ns, respectively. The stability of this circuit reflected a slight improvement over the stability of the RC oscillator shown in Figure 1.

LC OSCILLATOR

Figure 3 shows an LC oscillator circuit using standard TTL inverters.

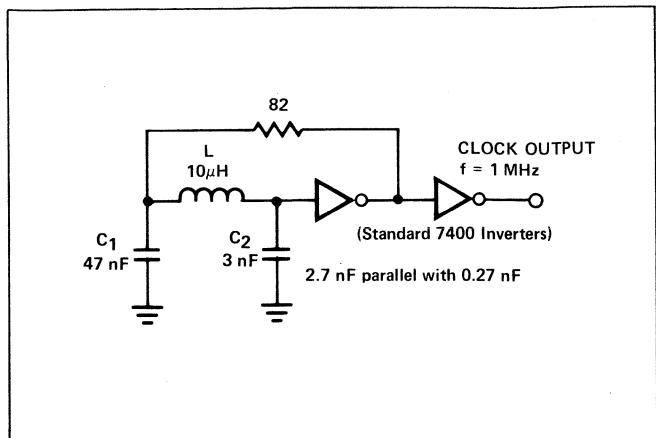


FIGURE 3. LC Clock Generator

The first inverter combined with the passive components forms a Colpitts oscillator. The resistor provides a feedback path for the first inverter and forces it into its linear region.

The second inverter "squares" the oscillator signal and provides an output buffer. The oscillator frequency can be derived from the following equation:

$$f_{osc} = \frac{1}{2\pi \sqrt{L} \left[\frac{(C1) \cdot (C2)}{(C1) + (C2)} \right]}$$

Measurements from the circuit in Figure 3 showed a 10 ns rise time and a 7 ns fall time. Measurements on frequency stability are provided in Table IV.

CRYSTAL OSCILLATORS

In 2650 Microprocessor applications requiring a highly stable clock, a crystal oscillator may be required. Some examples of crystal oscillator circuits are shown in Figures 4 and 5. The circuit shown in Figure 4 uses a 1.025 MHz crystal while the circuit shown in Figure 5 uses a low cost 4.433618 MHz crystal commonly found in European manufactured color TV sets. The output of the oscillator is divided by four to obtain a clock frequency of 1.1 MHz.

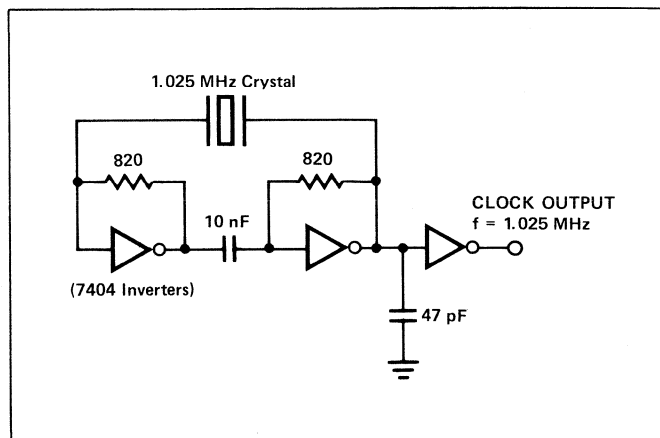


FIGURE 4. Clock Generator Using a Non-TV Standard Crystal

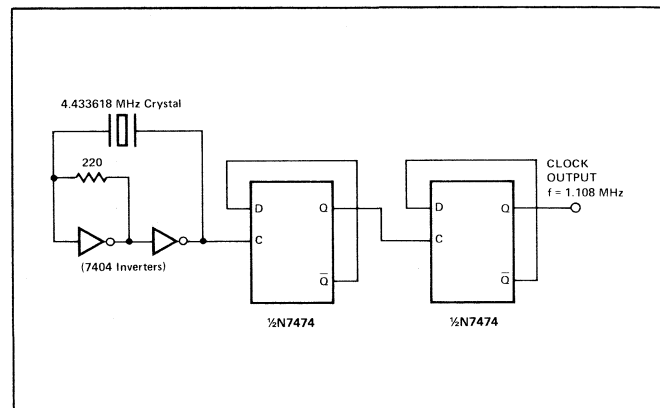


FIGURE 5. Low Cost Color TV Crystal Clock Generator

**TABLE IV
LC OSCILLATOR STABILITY
Ambient Temperature (T_A)**

	0°C	25°C	70°C	Stability* (V _{CC} = constant)
V _{CC} = 4.75V	1027.14 KHz	1017.75 KHz	1004.46 KHz	+0.92%, -1.31%
V _{CC} = 5.0V	1026.62 KHz	1016.99 KHz	1004.11 KHz	+0.95%, -1.26%
V _{CC} = 5.25V	1025.82 KHz	1016.30 KHz	1003.73 KHz	+0.94%, -1.24%
Stability** (T _A = constant)	+0.05% -0.08%	+0.07% -0.07%	+0.03% -0.04%	

*Stability_T with respect to T_A = 25°C
 **Stability_V with respect to V_{CC} = 5.0V

The circuit of Figure 5 can also be used with a 3.5795 MHz United States color TV crystal to provide an output frequency of 895 KHz.

The stability of the crystal oscillator circuits is mainly determined by the stability of the crystal used. The circuits shown in Figures 4 and 5 had a stability of 0.003% over the 0°C to 70°C temperature range and 0.002% over a variation of power supply voltage from 4.75V to 5.25V.

SUMMARY

Table V is a summary of the stability measurements made for the oscillator circuits described in this application note. As the table shows, the crystal circuits exhibit great stability relative to the RC and LC oscillators, but they suffer the added expense of the crystal. Any of the oscillator circuits shown in this application note can be used to drive the 2650 microprocessor clock input.

**TABLE V
SUMMARY OF OSCILLATOR STABILITY**

CIRCUIT TYPE	STABILITY					
	(4.75V to 5.25V)			(0°C to 70°C)		
	0°C	25°C	70°C	4.75V	5.0V	5.25V
RC	+0.12% -0.42%	+0.52% -0.98%	+0.2% -1.1%	+1.51% -2.96%	+1.91% -3.24%	+2.48% -3.35%
RC MONO-STABLE	+0.00% -0.014%	+0.276% -0.373%	+0.826% -0.833%	+1.62% -0.53%	+2.01% -0.98%	+2.29% -1.53%
LC	+0.05% -0.08%	+0.07% -0.07%	+0.03% -0.04%	+0.92% -1.31%	+0.95% -1.26%	+0.94% -1.24%
CRYSTAL	+0.0003%	-0.0001%	+0.0002%	+0.001%	±0.0001%	+0.0004%

Signetics 2650 Microprocessor application memos currently available:

AS50 Serial Input/Output
AS51 Bit and Byte Testing Procedures
AS52 General Delay Routines
AS53 Binary Arithmetic Routines
AS54 Conversion Routines
SP50 2650 Evaluation Printed Circuit Board Level System
(PC1001)
SP51 2650 Demo Systems
SP52 Support Software for use with the NCSS Timesharing
System
SP53 Simulator, Version 1.2
SP54 Support Software for use with the General Electric Mark III
Timesharing System
SS50 PIPBUG
SS51 Absolute Object Format (Revision 1)
MP51 2650 Initialization
MP52 Low Cost Clock Generator Circuits



Electronic
components
and materials

PHILIPS

ADDRESS AND DATA BUS

INTERFACING TECHNIQUES MP53

AN APPLICATION MEMO

signetics

1. INTRODUCTION

The Signetics 2650 Microprocessor has a 15-bit address bus and an 8-bit bi-directional data bus. The address bus allows a maximum of 32K words of memory. The drive capability of the 2650 address and data busses limits the number of chips that can be connected to the system. If the system load exceeds the 2650 drive capability, buffer circuits must be added.

This applications memo provides several examples of interfacing the 2650 address and data busses with ROMs and RAMs such as the 2608, 2606, and 2602. Examples are included for both small and large systems.

2. SMALL SYSTEMS WITHOUT BUFFERING

Address Bus Loading

All 2650 output signals are TTL-compatible. Each output can source 100 μA at 2.4V minimum and sink 1.6 mA at 0.45V maximum. The 2650 inputs require a load current of only 10 μA regardless of the logic level on the input.

The 2608, 2606, 2604, and 2602 MOS ROMs and RAMs all require an input current of 10 μA . This means that, based on d-c loading considerations, a maximum of ten inputs of this type can be driven from one 2650 address output without the use of buffering.

Data Bus Loading with the 2606 RAM (256 x 4)

The bi-directional data bus of the 2606 RAM (256 x 4) makes this device ideally suited for use with the 2650 Microprocessor. The maximum number of input/output connections can be calculated from the diagram shown in Figure 1.

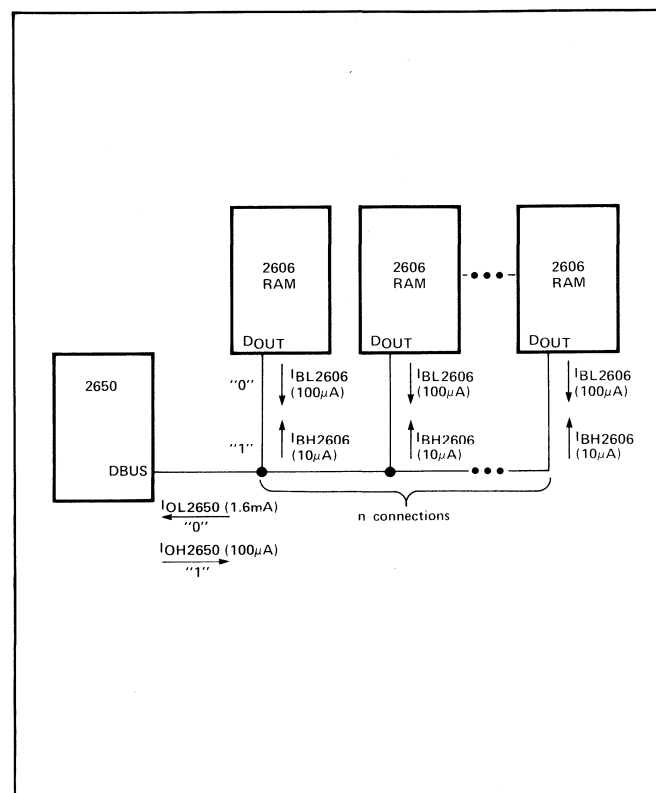


FIGURE 1 The 2606 RAM with the 2650

In Figure 1, n 2606 memory chips are driven by the 2650. The 2606 memory chips load the bus with a leakage current of 100 μA in the logic ZERO state and with 10 μA in the logic ONE state. When the data bus is driven to a logic "1", the required source current of the 2650 output will be:

$$I_{OH2650} = (n) \cdot I_{BH2606} \\ = (n) \cdot (10\mu\text{A})$$

where:

$$I_{BH2606} = \text{output logic ONE leakage current of the 2606 RAM;}$$

TABLE I

TYPICAL 2650 MEMORY CONFIGURATIONS WITHOUT BUFFERING

Number of Chips Connected to One Address Output	Memory Capacity
Eight 2606 RAMs (256 x 4) Two 2608 ROMs (1024 x 8)	1K byte RAM; 2K bytes ROM
Eight 2602 RAMs (1024 x 1) Two 2608 ROMs (1024 x 8)	1K byte RAM; 2K bytes ROM

If bipolar PROMs such as the 82S114 or 82S115 are used, fewer chips can be connected because of higher input current requirements.

and

I_{OH2650} = output logic ONE drive current of the 2650.

From this equation we calculate n_{max} :

$$n_{max} = \frac{I_{OH2650 \max}}{10 \mu A} = \frac{100 \mu A}{10 \mu A} = 10$$

In the logic ZERO state, the output current required of the 2650 is:

$$I_{OL2650} = (n) \cdot I_{BL2606} \\ = (10) \cdot (100 \mu A) = 1000 \mu A$$

where:

I_{BL2606} = output logic ZERO leakage current of the 2606 RAM;

and

I_{OL2650} = output logic ZERO drive current of the 2650.

This is less than the maximum drive capability of 1.6 mA for the 2650.

When the 2606 drives the data bus, the logic ONE loading is the same as that seen by a 2650 driving a data bus (previously described as I_{OH2650}). The logic ZERO load on the 2606 chip is:

$$I_{OL2606} = (n-1) I_{BL2606} + I_{LOL2650} \\ = [(9) \cdot (100 \mu A)] + 10 \mu A \\ = 910 \mu A$$

where:

$I_{LOL2650}$ = output logic ZERO leakage current of the 2650.

This is below the 1.9 mA sink current capability of the 2606. It can thus be concluded that when using MOS RAMs or ROMs with the 2650, the number n is normally limited by the maximum output logic ONE current of the driving device.

Data Bus Loading with the 2602 RAM

In contrast to the 2606, the 2602 RAM (1024 x 1) has separate input and output data paths. The data output for this device is switched to tri-state with the chip enable input. For bi-directional data transfers, however, the data output signal must be disabled during the write mode to avoid a drive conflict between the 2650 and the RAM. This is done by inserting a tri-state buffer into the data-out line as shown in Figure 2. The buffers are only enabled when OPREQ is a "HIGH", \bar{R}/W (the READ/WRITE control line from the 2650) is a "LOW", and the RAM is selected for access.

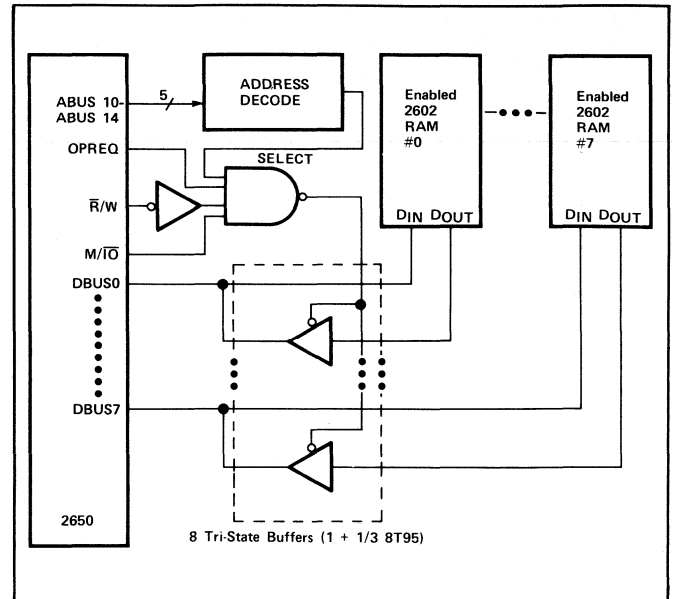


FIGURE 2 The 2602 RAM with the 2650

A-C Loading Considerations

The 2650 address bus, data bus, and control lines will drive a 100 pF capacitive load and one standard TTL load. The capacitive loading calculations must include the 2650 output capacitance and the external wiring capacitance. The 2606 presents a 10 pF capacitive load to the data bus and a 7 pF load to all other inputs. The number (n) of 2606 RAMs that can be driven directly by the 2650 is given by the following equations:

$$C_{LOAD} = C_{OUT2650} + C_{WIRING} + [(n_d) \cdot C_{OUT2606}]$$

or

$$C_{LOAD} = C_{OUT2650} + C_{WIRING} + [(n_a) \cdot C_{IN2606}]$$

- $C_{OUT2650}$ = Output capacitance for the 2650 = 10 pF
- C_{WIRING} = Wiring capacitance = 10 pF
- C_{IN2606} = Load capacitance for the 2606 address bus = 7 pF
- $C_{OUT2606}$ = Load capacitance for the 2606 data bus = 10 pF
- C_{LOAD} = 100 pF

therefore:

$$n_a = \frac{80 \text{ pF}}{7 \text{ pF}} \cong 11 \text{ address bus loads}$$

$$n_d = \frac{80 \text{ pF}}{10 \text{ pF}} \cong 8 \text{ data bus loads}$$

The 2606 is a 256-location by 4-bit RAM and requires two chips for each 256 bytes. As seen from the above calcula-

tions, the 2650 will drive eleven 2606s (n_a), or five pairs ($n_a/2$) of 2606s (1280 bytes) directly. Since this number is less than the number of d-c loads that the 2650 is capable of driving (10), it can be concluded that the a-c loading is the limitation for full-speed operation.

Increasing Fan-Out by Pull-Up Resistor

The fan-out of the 2650 bus in the logic ONE state can be increased with a pull-up resistor. This increases the d-c fan-out of the outputs in the logic ONE state by supplying supplementary drive current. This can be seen from the example shown in Figure 3.

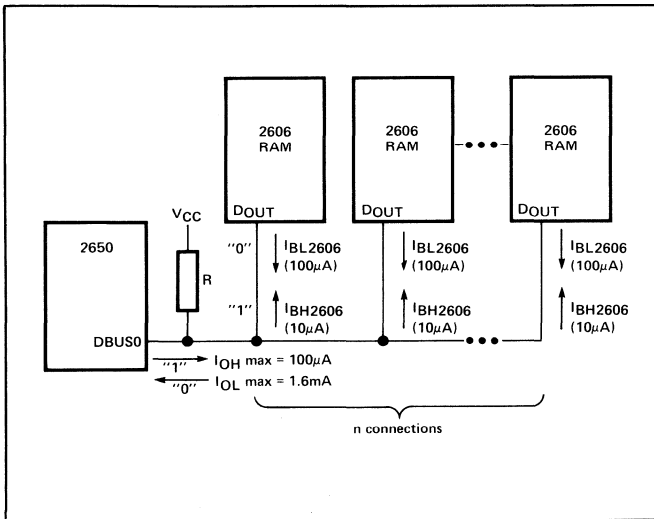


FIGURE 3 Pull-up Resistors for Increased Fan-out

$$\text{Logic ONE state } I_R = \frac{V_{CCmin} - V_{OHmax2650}}{R} - [(n) \cdot I_{BH2606}] - I_{OH2650}$$

$$\text{Logic ZERO state } I_R = \frac{V_{CCmax} - V_{OLmin2650}}{R} - I_{OL2650} - [(n) \cdot I_{BL2606}]$$

where:

$$V_{CCmin} = 4.75 \text{ volts}$$

$$V_{CCmax} = 5.25 \text{ volts}$$

$$V_{OHmax2650} = 2650 \text{ maximum logic ONE output voltage}$$

$$= V_{CC} - 0.5 \text{ volts}$$

$$V_{OLmin2650} = 2650 \text{ minimum logic ZERO output voltage}$$

$$= 0 \text{ volts}$$

$$I_{BH2606} = \text{output logic ONE leakage current of the 2606 RAM}$$

$$= 10 \mu\text{A}$$

$$I_{BL2606} = \text{output logic ZERO leakage current of the 2606 RAM}$$

$$= 100 \mu\text{A}$$

$$I_{OH2650} = \text{output logic ONE current of the 2650}$$

$$= 100 \mu\text{A}$$

$$I_{OL2650} = \text{output logic ZERO current of the 2650}$$

$$= 1.6 \text{ mA}$$

n = the number of 2606 type loads that can be driven by the 2650

From the above equations, R can be calculated to be 17.5K ohms. The number of 2606 loads (n) is calculated to be 12. Six pairs of 2606 chips can be driven when the pull-ups are added. These calculations are for d-c loading, and the a-c (capacitive) load limitations must still be considered.

$$\text{With } V_{CCmin} = 4.5\text{V and } V_{CCmax} = 5.5\text{V: } n = 10$$

$$R = 9\text{K}\Omega$$

$$\text{With } V_{CCmin} = 4.75\text{V and } V_{CCmax} = 5.25\text{V: } n = 12$$

$$R = 12\text{K}\Omega$$

3. LARGE BUFFERED SYSTEMS

In larger microcomputers it is necessary to increase the drive capability of the CPU by adding drivers to the outputs. A generalized 2650 microcomputer system using additional bus drivers is illustrated in Figure 4.

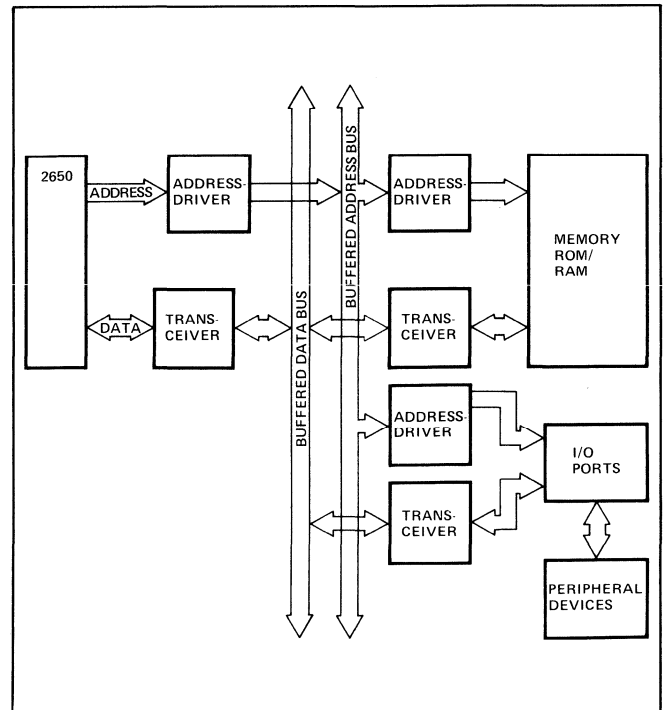


FIGURE 4 General-Purpose Microcomputer System

This system has a buffered address bus and a buffered data bus. To ensure minimal loading, buffers are also included between the memory and I/O ports. With this arrangement, the system can easily be expanded, and each additional device adds a single load to the shared bus.

In some cases, the configuration in Figure 4 can be simplified as shown in Figure 5. The memory and I/O ports are directly driven by the address driver and transceiver circuits.

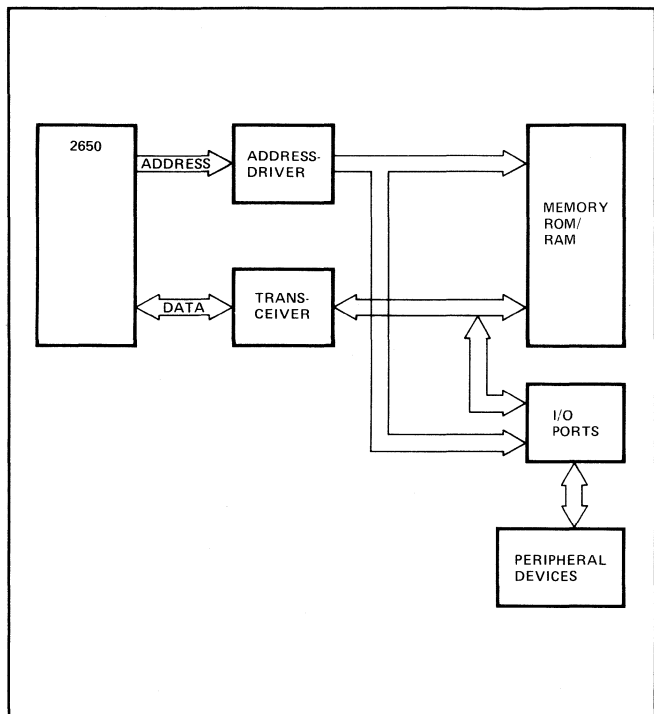


FIGURE 5 Microprocessor with Buffered Address and Data Bus

Address Driver

The address bus driver may be a non-inverting interface element of the 8T family, such as the 8T95 or 8T97 shown in Figure 6.

The tri-state control inputs (DIS4 and DIS2) can be connected to ground if these buffers are always active. For DMA operations, the control inputs can be switched to a HIGH to disconnect the processor from the bus. These Schottky-TTL devices have typical propagation delays of 6 ns. (See Table III.)

Standard TTL buffers may be used to drive the address bus. If buffers with open-collector outputs or tri-state capability are used, DMA operations can be performed.

Data Transceivers

The 2650 bi-directional data lines can be driven with the 8T26 (inverting) and 8T28 (non-inverting) transceivers (Figure 7).

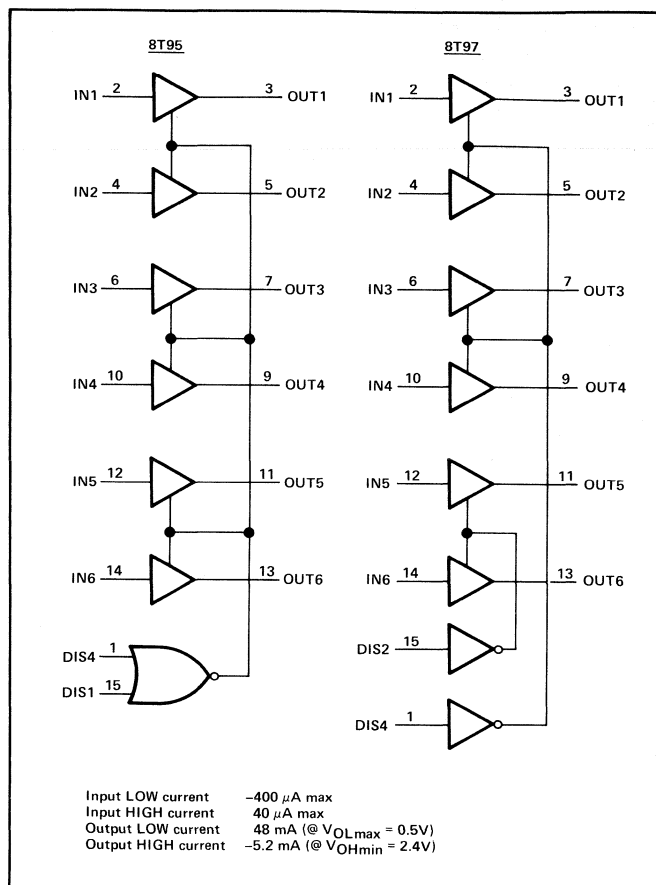


FIGURE 6 8T95 and 8T97 Hex Tri-State Buffers

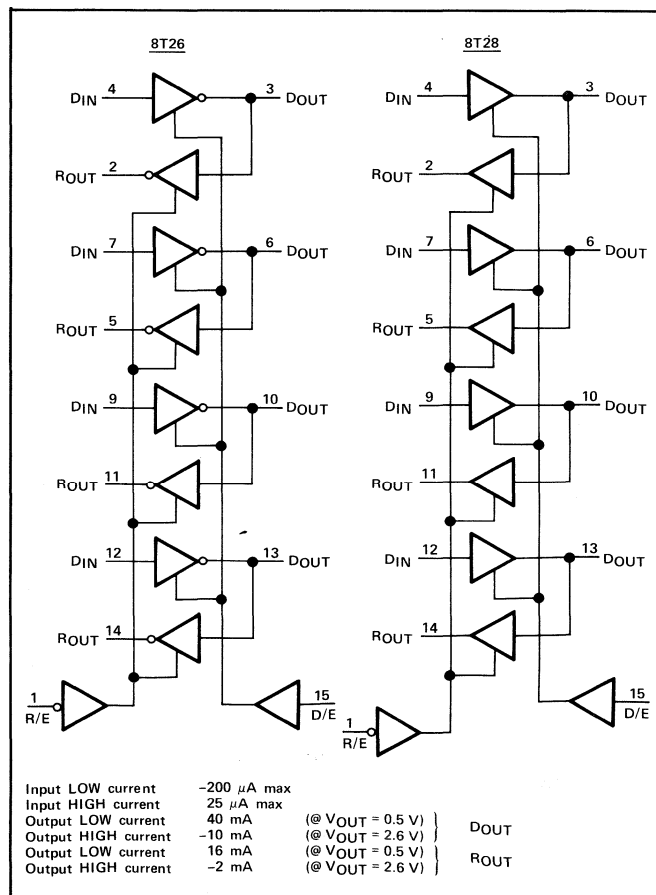


FIGURE 7 Tri-State Quad Bus Transceivers

ADDRESS AND DATA BUS INTERFACING TECHNIQUES ■ MP53

The driver can be enabled by the driver enable line (D/E, active high). The receiver can be enabled by the receiver enable line (R/E, active low). To drive the 2650 bi-directional data bus, the D_{IN} and R_{OUT} signals can be tied together to provide a bi-directional data path.

Figure 8 shows a typical application of the transceiver circuit for bi-directional data buffering. The 8T28 features a propagation delay of 20 ns with a 300 pF capacitive load.

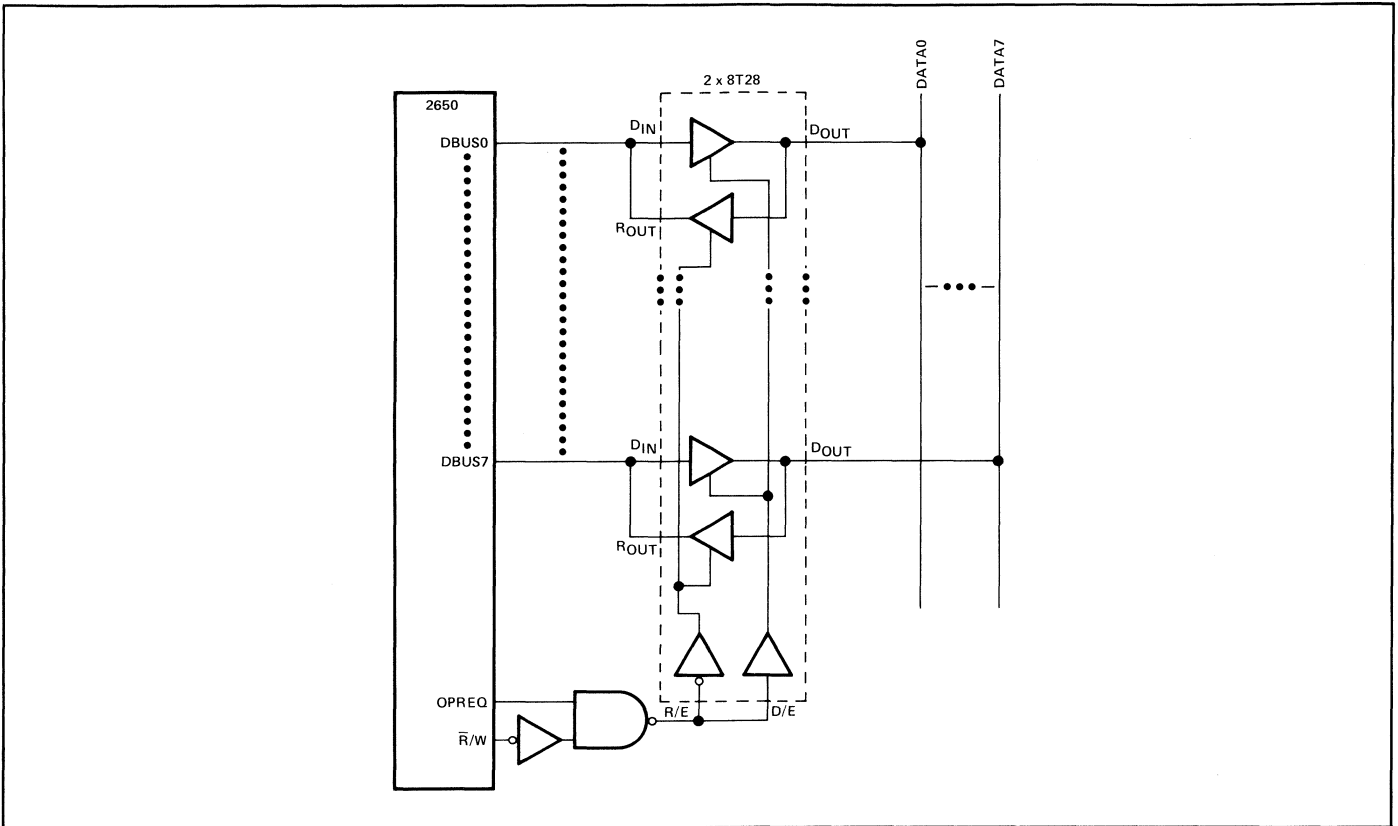


FIGURE 8 Typical Application of the Transceiver Circuit

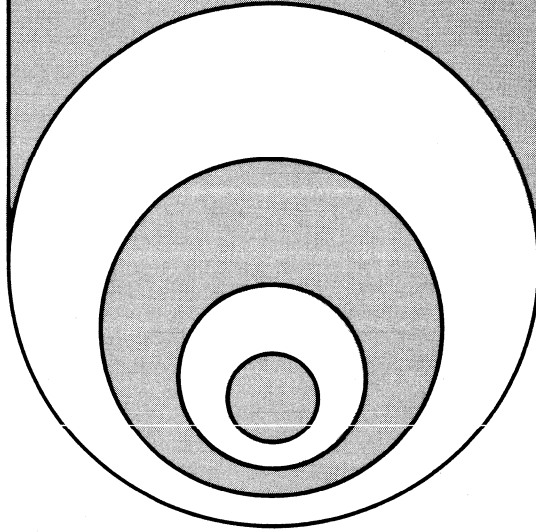
TABLE II
MOS RAMs - SURVEY OF D-C ELECTRICAL CHARACTERISTICS

PARAMETER	SYMBOL	2606 (256 x 4)	2602 (1024 x 1)	2604 (4096 x 1)	UNIT	TEST CONDITIONS
Maximum input load current	I_{IL}	10	10		μA	$V_{IN} = 0$ to 5.25V
				10	μA	$V_{IN} = +5V$
Maximum input LOW voltage	V_{IL}	0.65	0.65	0.6	V	
Minimum input HIGH voltage	V_{IH}	2.2	2.2	2.2	V	
Maximum output LOW voltage	V_{OL}	0.45	0.45		V	$I_{OL} = 1.9$ mA
				0.4	V	$I_{OL} = 3.2$ mA
Minimum output HIGH voltage	V_{OH}	2.4	2.4		V	$I_{OH} = -100$ μA
				2.4	V	$I_{OH} = -2$ mA
Maximum output HIGH leakage current	I_{BH}	10	10	10*	μA	$\bar{C}E = 2.2V$; $V_{OUT} = 4.0V$
Maximum output LOW leakage current	I_{BL}	-100	-100		μA	$\bar{C}E = 2.2V$; $V_{OUT} = 0.45V$
Maximum input capacitance	C_{IN}	7	5	7	pF	$V_{IN} = 0V$
Maximum bus input capacitance	C_{OUT}	10	10	6	pF	$V_{OUT} = 0V$
Common I/O		X				
Separate I/O			X	X		

*Test conditions $\bar{C}S = 2.2V$; $V_{OUT} = 5V$

signetics

MICROPROCESSOR



SIMULATOR, VERSION 1.2.....SP53

APPLICATIONS MEMO

A new version of the Simulator is available. This version performs the same functions as Version 1.0 (see Simulator Manual) with the following additional features:

1. Hexadecimal Object Module

The Simulator accepts an object module produced by the Assembler in either decimal or hexadecimal format. The Simulator assumes that the object module is hexadecimal, unless the user specifies a decimal module by adding a fourth parameter, **FORMAT**, to the "EXECUTE SIMULATOR" command. This command is formatted differently depending upon the computer system on which the Simulator is installed.

2. 8K (8192 bytes) Object Module

The Simulator reads and executes an object module with up to 8192 bytes.

3. Decimal Input to LIMIT Command

The LIMIT command expects the number of instructions to be entered in decimal, not hexadecimal. Thus, a "LIMIT 40" command causes the program to execute 40₁₀ not 64₁₀ instruction. All other commands still expect their input parameters to be in hexadecimal.

4. Stack Wraparound Notification

Whenever a RETC or a RETE is executed with the stack pointer equal to 0 or whenever a branch to subroutine instruction is executed with the stack pointer equal to 7, the Simulator prints the following message:

STACK WRAPAROUND, IAR=XXXX

Where XXXX identifies the address at which the wrap around occurred.

5. Termination Messages

The Simulator prints a message for every kind of program termination:

TYPE OF TERMINATION	SIMULATOR RESPONSE
1. STOP. command	A trace of the last instruction executed is printed.
2. HALT instruction	A trace of the last instruction executed is printed.
3. LIMIT command	"LIMIT REACHED=XXXX, IAR=XXXX" is printed. A trace of the last instruction executed is printed.
4. Attempt to access area outside of memory	"ADDRESS OUT OF RANGE, IAR=XXXX" is printed. A trace of the last instruction executed is printed.
5. Attempt to execute instruction outside of memory	"IAR EXCEEDS MEMORY, IAR=XXXX" is printed.

6. Simulator Version Notification

The simulator prints the following message whenever it starts to execute a program:

2650 SM 1000 "PIPSIM" VERSION X.X

X.X identifies the version of the simulator currently executing.

TABLE III
BUFFERS - SURVEY OF ELECTRICAL CHARACTERISTICS

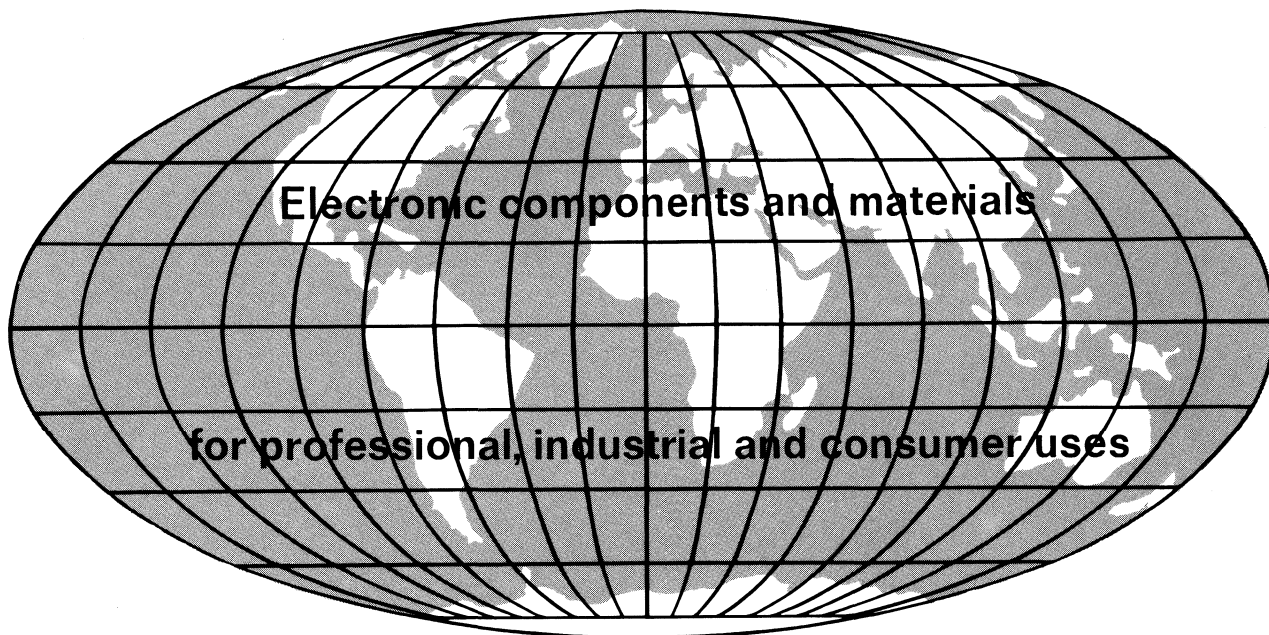
PARAMETER	SYMBOL	8T09 (quad)	8T95/97 (hex)	8T96/98 (hex)	UNIT	TEST CONDITIONS
Inverting		X		X		
Non-inverting			X			
Maximum input LOW current	I _{ILmax}	-2			mA	V _I = 0.4V; DIS = 0.4V
			-0.4	-0.4	mA	V _I = 0.5V; DIS = 0.5V
Maximum input HIGH current	I _{IHmax}	40			μA	DIS = 4.5V
			40	40	μA	V _I = 2.4V
Maximum input LOW voltage	V _{ILmax}	0.8	0.8	0.8	V	V _{CC} = MIN; T _A = 25°C
Minimum input HIGH voltage	V _{IHmin}	2.0	2.0	2.0	V	V _{CC} = MIN; T _A = 25°C
Maximum output LOW voltage	V _{OLmax}	0.4			V	I _{OL} = 40 mA
			0.5	0.5	V	I _{OL} = 48 mA
Minimum output HIGH voltage	V _{OHmin}	2.4	2.4	2.4	V	I _{OH} = -5.2 mA
Maximum output leakage current HIGH	I _{BH}	40	40	40	μA	V _O = 2.4V
Maximum output leakage current LOW	I _{BL}	-40	-40*	-40*	μA	V _O = 0.4V
Propagation delay (data to output)	t _{ON}	20**	5***	5***	ns	
	t _{OFF}	20	6	6	ns	
Propagation delay (disable to output)	High Z/0	22	12	12	ns	
	High Z/1	22	10	10	ns	

*Test condition V_O = 0.5V
 **Test condition C_L = 300 pF
 ***Test condition C_L = 50 pF

TABLE IV
(P)ROMs - SURVEY OF D-C ELECTRICAL CHARACTERISTICS

PARAMETER	SYMBOL	2608 (1024 x 8)	82S114 (256 x 8) 82S115 (512 x 8)	82S130 (512 x 4) 82S131 (512 x 4)	82S126 (256 x 4) 82S129 (256 x 4)	UNIT	TEST CONDITIONS
Maximum input load current	I _{IL}	10				μA	
Maximum input LOW current	I _{ILmax}	10	-100	-100	-100	μA	V _{IN} = 0.45V
Maximum input HIGH current	I _{IHmax}	10	25	40	40	μA	V _{IN} = 5.5V
Maximum input LOW voltage	V _{ILmax}	0.65	0.85	.85	0.85	V	
Minimum input HIGH voltage	V _{IHmin}	2.2	2.0	2.0	2.0	V	
Maximum output LOW voltage	V _{OLmax}	0.45				V	I _{OL} = 1.6 mA (2608)
			0.5			V	I _{OL} = 9.6 mA (82S114, 82S115)
				0.45	0.5	V	I _{OL} = 16 mA (82S130, 82S131, 82S126, 82S129)
Minimum output HIGH voltage	V _{OHmin}	2.4				V	I _{OH} = -100 μA
			2.7			V	I _{OH} = -2 mA
				2.4	2.4	V	I _{OH} = -2.4 mA
Maximum output leakage current	I _{BH}	10*	40	40	40	μA	V _O = 5.5V; device deselected
Maximum output leakage current L	I _{BH}	-10**	-40	-40	-40	μA	V _O = 0.5V; device deselected
Maximum input capacitance	C _{IN}	7.5	5	5	5	pF	
Maximum output capacitance	C _{OUT}	15	8	8	8	pF	

*Test conditions V_O = 2.4V **Test conditions V_O = 0.4V



from the world-wide Philips Group of Companies

- Argentina:** FAPESA I.y.C., Av. Crovara 2550, Tablada, Prov. de BUENOS AIRES, Tel. 652-7438/7478.
- Australia:** PHILIPS INDUSTRIES HOLDINGS LTD., Elcoma Division, 67 Mars Road, LANE COVE, 2066, N.S.W., Tel. 42 1261.
- Austria:** ÖSTERREICHISCHE PHILIPS BAUELEMENTE Industrie G.m.b.H., Triester Str. 64, A-1101 WIEN, Tel. 62 91 11.
- Belgium:** M.B.L.E., 80, rue des Deux Gares, B-1070 BRUXELLES, Tel 523 00 00.
- Brazil:** IBRAPE, Caixa Postal 7383, Av. Paulista 2073-S/Loja, SAO PAULO, SP, Tel. 287-7144.
- Canada:** PHILIPS ELECTRONICS LTD., Electron Devices Div., 601 Milner Ave., SCARBOROUGH, Ontario, M1B 1M8, Tel. 292-5161.
- Chile:** PHILIPS CHILENA S.A., Av. Santa Maria 0760, SANTIAGO, Tel. 39-40 01.
- Colombia:** SADAPE S.A., P.O. Box 9805, Calle 13, No. 51 + 39, BOGOTA D.E. 1., Tel. 600 600.
- Denmark:** MINIWATT A/S, Emdrupvej 115A, DK-2400 KØBENHAVN NV., Tel. (01) 69 16 22.
- Finland:** OY PHILIPS AB, Elcoma Division, Kaivokatu 8, SF-00100 HELSINKI 10, Tel. 1 72 71.
- France:** R.T.C. LA RADIOTECHNIQUE-COMPELEC, 130 Avenue Ledru Rollin, F-75540 PARIS 11, Tel. 355-44-99.
- Germany:** VALVO, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, D-2 HAMBURG 1, Tel. (040) 3296-1.
- Greece:** PHILIPS S.A. HELLENIQUE, Elcoma Division, 52, Av. Syngrou, ATHENS, Tel. 915 311.
- Hong Kong:** PHILIPS HONG KONG LTD., Comp. Dept., Philips Ind. Bldg., Kung Yip St., K.C.T.L. 289, KWAI CHUNG, N.T. Tel. 12-24 51 21.
- India:** PHILIPS INDIA LTD., Elcoma Div., Band Box House, 254-D, Dr. Annie Besant Rd., Prabhadevi, BOMBAY-25-DD, Tel. 457 311-5.
- Indonesia:** P.T. PHILIPS-RALIN ELECTRONICS, Elcoma Division, 'Timah' Building, Jl. Jen. Gatot Subroto, JAKARTA, Tel. 44 163.
- Ireland:** PHILIPS ELECTRICAL (IRELAND) LTD., Newstead, Clonskeagh, DUBLIN 14, Tel. 69 33 55.
- Italy:** PHILIPS S.P.A., Sezione Elcoma, Piazza IV Novembre 3, I-20124 MILANO, Tel. 2-6994.
- Japan:** NIHON PHILIPS CORP., Shuwa Shinagawa Bldg., 26-33 Takanawa 3-chome, Minato-ku, TOKYO (108), Tel. 448-5611.
(IC Products) SIGNETICS JAPAN, LTD., TOKYO, Tel. (03) 230-1521.
- Korea:** PHILIPS ELECTRONICS (KOREA) LTD., Philips House, 260-199 Itaewon-dong, Yongsan-ku, C.P.O. Box 3680, SEOUL, Tel. 44-4202.
- Mexico:** ELECTRONICA S.A. de C.V., Varsovia No. 36, MEXICO 6, D.F., Tel. 5-33-11-80.
- Netherlands:** PHILIPS NEDERLAND B.V., Afd. Elonco, Boschdijk 525, NL-4510 EINDHOVEN, Tel. (040) 79 33 33.
- New Zealand:** Philips Electrical Ind. Ltd., Elcoma Division, 2 Wagener Place, St. Lukes, AUCKLAND, Tel. 867 119.
- Norway:** ELECTRONICA A/S., Vitaminveien 11, P.O. Box 29, Grefsen, OSLO 4, Tel. (02) 15 05 90.
- Peru:** CADESA, Jr. Ilo, No. 216, Apartado 10132, LIMA, Tel. 27 73 17.
- Philippines:** ELDAC, Philips Industrial Dev. Inc., 2246 Pasong Tamo, MAKATI-RIZAL, Tel. 86-89-51 to 59.
- Portugal:** PHILIPS PORTUGESA S.A.R.L., Av. Eng. Duharte Pacheco 6, LISBOA 1, Tel. 68 31 21.
- Singapore:** PHILIPS SINGAPORE PTE LTD., Elcoma Div., POB 340, Toa Payoh CPO, Lorong 1, Toa Payoh, SINGAPORE 12, Tel. 53 88 11.
- South Africa:** EDAC (Pty.) Ltd., South Park Lane, New Doornfontein, JOHANNESBURG 2001, Tel. 24/6701.
- Spain:** COPRESA S.A., Balmes 22, BARCELONA 7, Tel. 301 63 12.
- Sweden:** A.B. ELCOMA, Lidingsvägen 50, S-10 250 STOCKHOLM 27, Tel. 08/67 97 80.
- Switzerland:** PHILIPS A.G., Elcoma Dept., Edenstrasse 20, CH-8027 ZÜRICH, Tel. 01/44 22 11.
- Taiwan:** PHILIPS TAIWAN LTD., 3rd Fl., San Min Building, 57-1, Chung Shan N. Rd. Section 2, P.O. Box 22978, TAIPEI, Tel. 5513101-5.
- Turkey:** TÜRK PHILIPS TICARET A.S., EMET Department, Inonu Cad. No. 78-80, İSTANBUL, Tel. 43 59 10.
- United Kingdom:** MULLARD LTD., Mullard House, Torrington Place, LONDON WC1E 7HD, Tel. 01-580 6633.
- United States:** (Active devices & Materials) AMPEREX SALES CORP., 230, Duffy Avenue, HICKSVILLE, N.Y. 11802, Tel. (516) 931-6200.
(Passive devices) MEPCO/ELECTRA INC., Columbia Rd., MORRISTOWN, N.J. 07960, Tel. (201) 539-2000.
(IC Products) SIGNETICS CORPORATION, 811 East Arques Avenue, SUNNYVALE, California 94086, Tel. (408) 739-7700.
- Uruguay:** LUZILECTRON S.A., Rondeau 1567, piso 5, MONTEVIDEO, Tel. 9 43 21.
- Venezuela:** IND. VENEZOLANAS PHILIPS S.A., Elcoma Dept., A. Ppal de los Ruices, Edif. Centro Colgate, Apdo 1167, CARACAS, Tel. 36 05 11.



Electronic
components
and materials

PHILIPS

2650 INPUT/OUTPUT STRUCTURES
AND INTERFACES MP54

AN APPLICATION MEMO

signetics

INTRODUCTION

Interfacing a microprocessor to peripheral devices is an important part of a total microcomputer system design. The characteristics of the interface depend to a large extent on total system requirements and other factors such as CPU loading and data speed. The use of interrupts and/or DMA structures also have an impact on the system input/output structure. The design of an I/O interface is not limited to hardware, and hardware/software trade-offs must be considered.

This applications memo examines the use of the 2650's versatile set of I/O instructions and the interface between the 2650 and I/O ports. Interrupt and DMA-controlled I/O are not discussed. A number of application examples for both serial and parallel I/O are given. Several types of input, output, and bidirectional interface devices are also examined.

Basic I/O Structure of the 2650

The 2650 is equipped with extensive and versatile input and output facilities. It can perform both single bit input/output and 8-bit parallel input/output.

The single bit input and output, called Sense (pin 1) and Flag (pin 40), are associated with the Program Status Word Upper (PSWU). The Flag output always reflects the value of bit 6 of the PSWU, while bit 7 of the PSWU always reflects the value of the Sense input signal. The Sense and Flag signals can be monitored and controlled with the PSW instructions.

Parallel I/O can be accomplished using the extended or non-extended read and write instructions. The extended and non-extended types are distinguished by the state of the E/\overline{NE} output of the 2650.

The non-extended I/O instructions are single-byte instructions which accomplish a 1-byte data transfer into or out of the 2650. They also control the state of the D/\overline{C} output, which can be used as a 1-bit device address in small systems.

The extended I/O instructions are 2-byte instructions. When executing extended I/O instructions, the second byte of the instruction is output on the lower 8 bits of the address bus (ADR0-ADR7). This information is normally used as an I/O device address to select 1 of up to 256 input or output devices, but may also be used to output control or status signals.

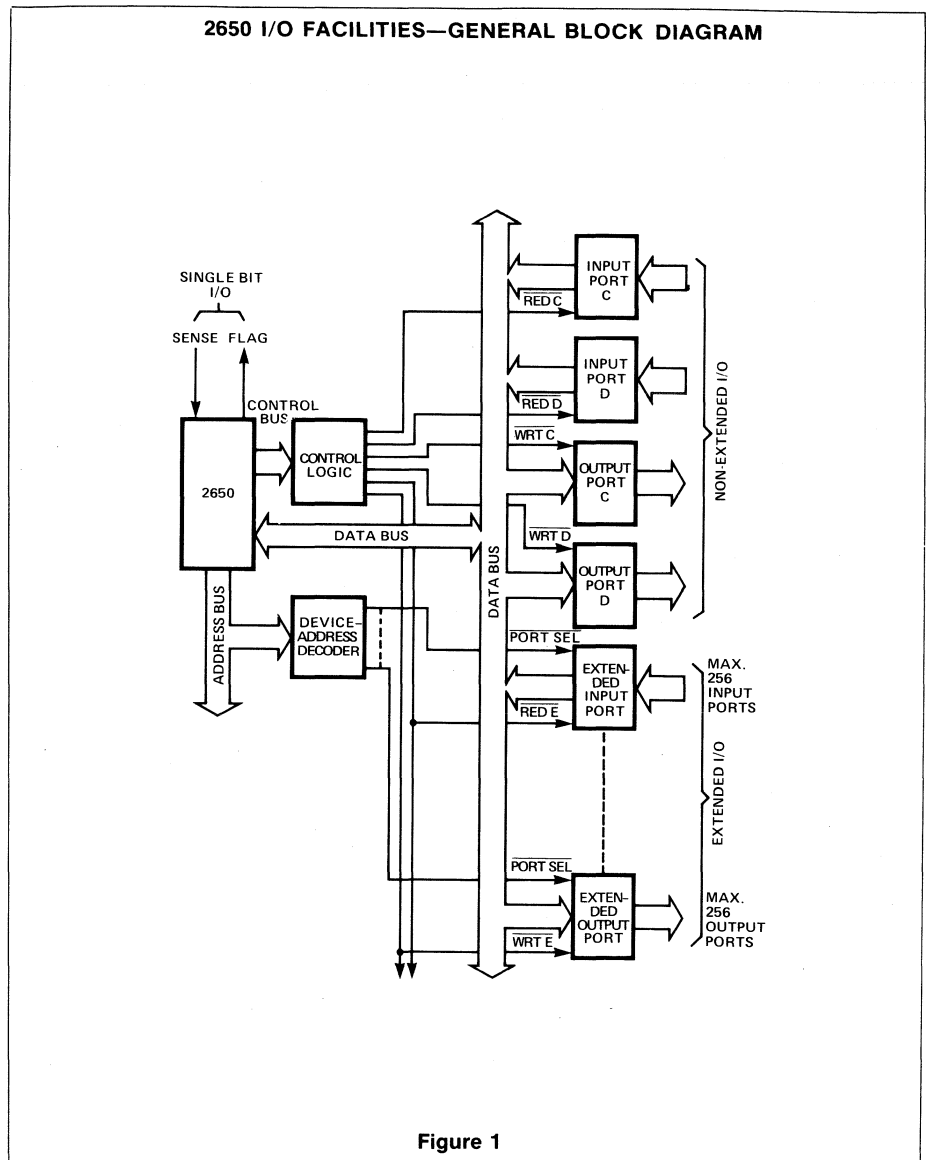


Figure 1

Parallel I/O operations may use any CPU register as the data source or destination. This offers significant flexibility in writing I/O software, because there is not a single accumulator register to create a "bottle-neck" in the data flow. The functional block diagram in Figure 1 illustrates the various I/O facilities.

I/O As Part of the Memory Address Space

The 2650 user may choose to transfer data into or out of the processor using the memory control signals. The advantage of this technique is that the data can be read or written by the program with memory load and store instructions, and data may be directly operated upon with logical and

arithmetic instructions. The memory referencing instructions can take advantage of the flexible addressing modes provided by the 2650, such as indexing and indirect addressing. A possible disadvantage of this method is that it may be necessary to decode more address lines to determine the device address than with the other I/O facilities.

To make use of this technique, the designer must assign memory addresses to I/O devices and design the device interfaces to respond to the same signals as memory.

I/O Interface Signals

Table I summarizes the state of the 2650 I/O interface signals for the various methods of I/O which are available.

SERIAL I/O USING THE SENSE INPUT AND FLAG OUTPUT

One of the I/O capabilities of the 2650 is provided by the sense input and flag output. The sense and flag pins may be used for single-bit input or output of status or control information. They can also be used to implement a serial data communications channel. Two examples of this application are given below.

Asynchronous Serial Communications Port

In applications where a serial type of terminal (like a teletypewriter) must be connected to the microcomputer system, the sense pin and flag pin can be used to interface with the terminal. The basic character format for asynchronous serial I/O is shown in Figure 2.

A number of parameters of this character format, and the transmission speed, are different for various types of terminals. The variable parameters are:

Baud rate (bits per second): 110, 150, 300, 600, 1200, 2400, 4800, and 9600 baud.

Number of bits per character: 5, 6, 7, or 8 bits.

Parity mode: even, odd, and no parity

Number of stop bits: 1 or 2

The control of the sense and flag pins for asynchronous serial I/O, with the appropriate parameters and baud rate, can be done completely with software. The hardware involved is limited to a simple line driver and receiver circuit which may be either an RS-232 interface or a 20mA current loop interface. The interface hardware is shown in Figure 3.

The software necessary to accomplish the serial I/O for a full-duplex line can be divided into 3 parts:

- The start bit detection and verification. After each start bit detection, the start-bit level is verified for a low level at time intervals of 1/6 of 1 bit time. This prevents false start-bit recognition caused by line noise.
- The sampling of the data bits at the mid-bit time, echoing the data bit to the flag output, and loading the data bit into a CPU register.
- The input, echo and check of parity bit and stop bits.

A timing diagram showing the start bit sampling and the bit echo appears in Figure 4.

TYPE OF I/O OPERATION	OPREQ	M/I \bar{O}	\bar{R}/W	ADRO-ADR7	ADR13 (E/ \bar{N} E)	ADR14 (D/ \bar{C})
Sense (Input)	X	X	X	X	X	X
Flag (Output)	X	X	X	X	X	X
Extended Read	H	L	L	Second Byte of Instruction	H	X
Extended Write	H	L	H	Second Byte of Instruction	H	X
Non-Extended Read C	H	L	L	X	L	L
Non-Extended Read D	H	L	L	X	L	H
Non-Extended Write C	H	L	H	X	L	L
Non-Extended Write D	H	L	H	X	L	H
Memory I/O Read	H	H	L	ADR0-ADR7	ADR13	ADR14
Memory I/O Write	H	H	H	ADR0-ADR7	ADR13	ADR14

X = Don't Care

Table 1 I/O INTERFACE SIGNAL STATE

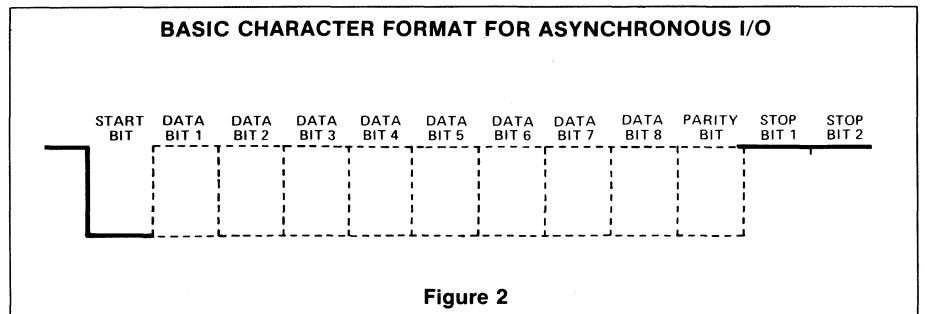


Figure 2

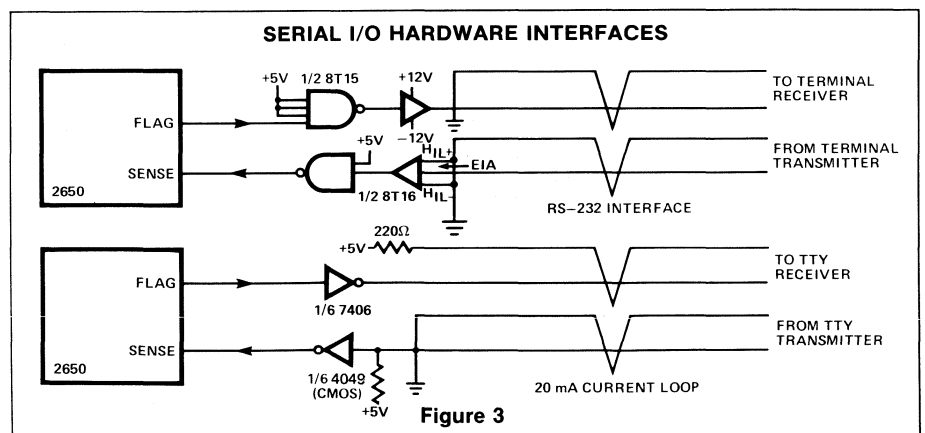


Figure 3

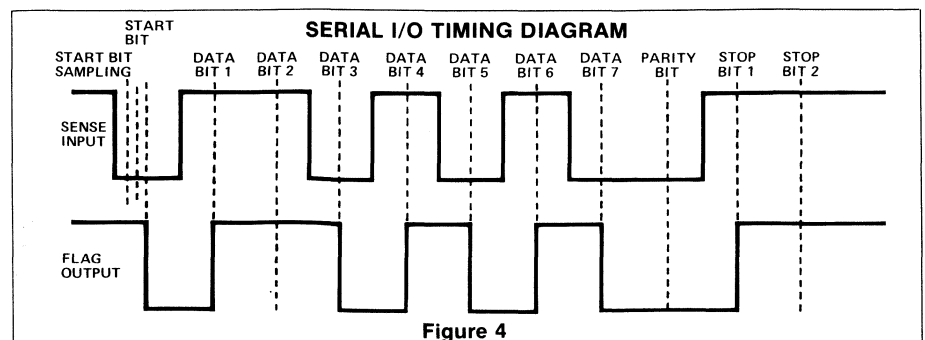


Figure 4

Three examples of the serial I/O routine with different speeds and parameters are presented in Figures 5 through 9. The bit and sample delay numbers (hexadecimal) in the definition listing (Figure 6) are for a CPU clock frequency of 1MHz. The hexadecimal delay numbers for a frequency of 1.25MHz are given in Table II. This table also lists the number of BDRR,R0 instructions that are necessary in the "bit delay and echo subroutine" to count cycles for the appropriate baud rate.

The examples of figures 7, 8, and 9 have the following parameters:

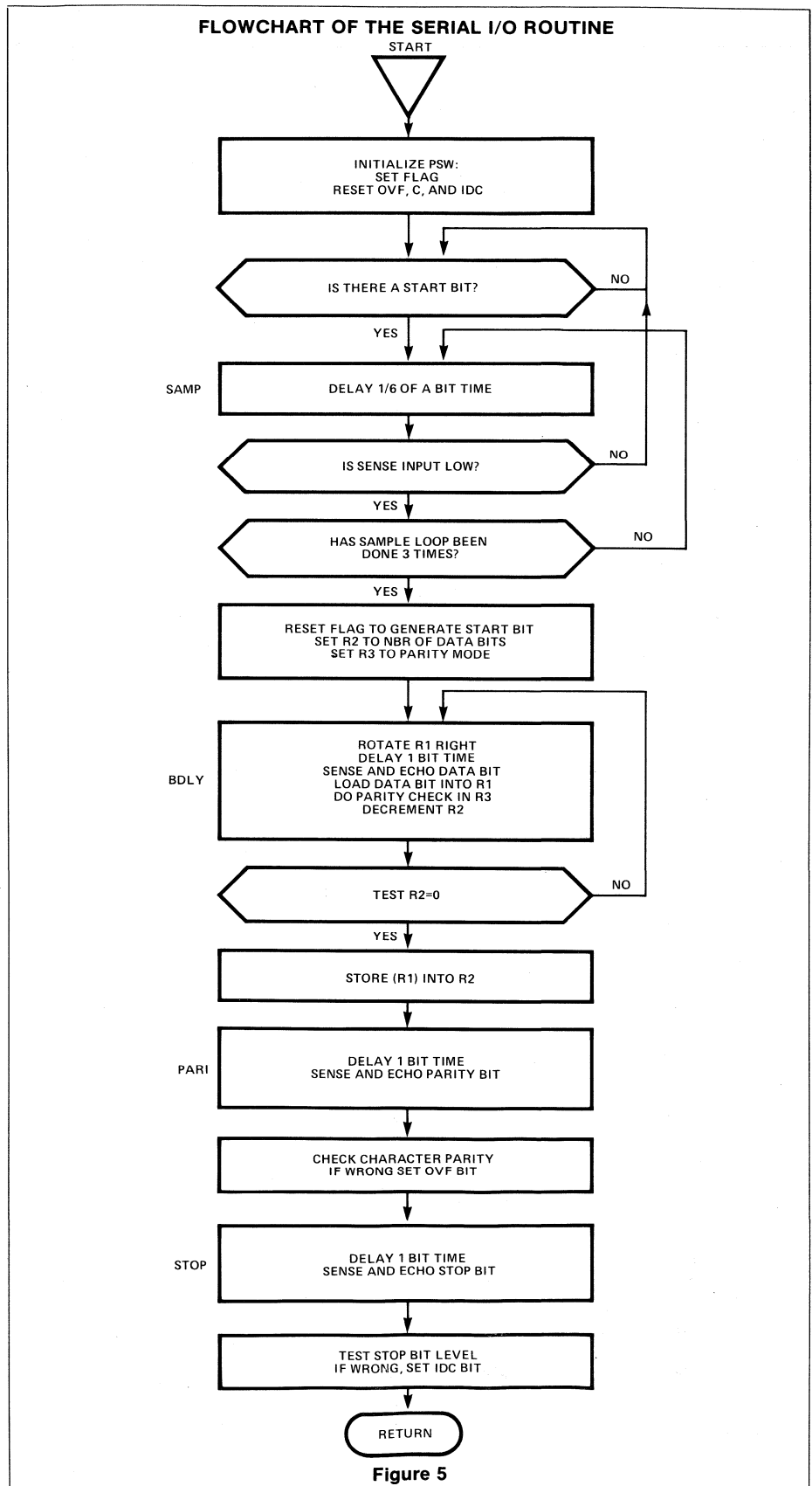
Figure 7: 110 baud, 7 data bits, even parity and 1 stop bit.

Figure 8: 600 baud, 7 data bits, odd parity and 2 stop bits.

Figure 9: 2400 baud, 8 data bits, no parity and 1 stop bit.

The serial I/O routine uses 4 CPU registers (1 bank and R0) and affects 7 of the Program Status Word bits; namely, Sense, Flag, Overflow, Carry, Interdigit Carry, and the 2 Condition Code bits. The program also uses 1 level of the return address stack.

A parity error will set the Overflow bit, and a framing error (wrong stop bit level) will set the Interdigit Carry bit. At the end of the routine, the input character is stored in register R2.



SERIAL I/O ASSEMBLY LISTING

EXAMPLE 2

```
TWIN ASSEMBLER VER 1.0 PAGE 0003
LINE ADDR OBJECT E SOURCE
0073 *****
0074 * EXAMPLE 2: FULL DUPLEX (BIT BY BIT ECHO), 600 BAUD,
0075 * 7 DATA BITS, ODD PARITY AND 2 STOP BITS.
0076 *
0077 0000 ORG H'0500'
0078 0500 7640 STRT PPSU F SET FLAG TO SWITCH OFF THE LINE
0079 0502 7525 CP5L OVF+C+IDC
0080 0504 12 TEST SPSU WAIT FOR START BIT
0081 0505 1A7D BCTR,N TEST
0082 0507 060C LODI,R2 H'03' SET R2 TO NUMBER OF SAMPLES
0083 0509 051C SAMP LODI,R1 5D66 SET R1 TO SAMPLE DELAY
0084 050B F37E BARR,R1 $
0085 050D 12 SPSU TEST FOR START BIT VALIDITY
0086 050E 1A74 BCTR,N TEST IF NOT VALID, GO BACK TO TEST
0087 0510 FA77 BARR,R2 SAMP
0088 0512 0780 LODI,R3 0P SET R3 TO ODD PARITY MODE
0089 0514 0607 LODI,R2 D87 SET R2 TO NUMBER OF DATA BITS
0090 0516 7440 CPSU F GENERATE START BIT
0091 0518 51 BITS RRR,R1
0092 0519 3B1A BSTR,UN B0LY GO TO DELAY AND ECHO ROUTINE
0093 051B FA7B BARR,R2 BITS TEST FOR NUMBER OF DATA BITS
0094 051D 01 LOD2 R1
0095 051E C2 STR2 R2 LOAD R2 WITH CHARACTER
0096 051F 3B14 PARI BSTR,UN B0LY
0097 0521 9A02 BCFR,N ST01
0098 0523 7704 PPSL OVF IF WRONG PARITY, SET OVF
0099 0525 0700 ST01 LODI,R3 0 CLEAR R3
0100 0527 3B0C BSTR,UN B0LY
0101 0529 1A02 BCTR,N ST02 TEST STOP BIT LEVEL
0102 052B 7720 PPSL IDC IF WRONG, SET IDC BIT
0103 052D 0700 ST02 LODI,R3 0 CLEAR R3
0104 052F 3B04 BSTR,UN B0LY
0105 0531 16 EX11 RETC,N TEST STOP BIT 2 LEVEL
0106 0532 7720 PPSL IDC IF WRONG, SET IDC BIT
0107 0534 17 EX12 RETC,UN
0108 *
0109 *****
0110 * BIT DELAY AND ECHO SUBROUTINE
0111 *
0112 0535 04B0 B0LY LODI,R0 B066 SET R0 TO BIT DELAY NUMBER
0113 0537 F87E BARR,R0 $
0114 0539 12 SPSU TEST DATA BIT LEVEL
0115 053A 1A04 BCTR,N ONE
0116 053C 7440 CPSU F IF LOW, ECHO A ZERO
0117 053E 1B04 BCTR,UN BIT1
0118 0540 7640 ONE PPSU F IF HIGH, ECHO A ONE
0119 0542 6540 IORI,R1 BP8 INSERT DATA BIT INTO R1
0120 0544 23 BIT1 EOR2 R3
0121 0545 C3 STR2 R3 DO PARITY CHECK
0122 0546 17 RETC,UN
0123 *
0124 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000
```

Figure 8

EXAMPLE 3

```
TWIN ASSEMBLER VER 1.0 PAGE 0003
LINE ADDR OBJECT E SOURCE
0073 *****
0074 * EXAMPLE 3: FULL DUPLEX (BIT BY BIT ECHO), 2400 BAUD,
0075 * 8 DATA BITS, NO PARITY AND 1 STOP BIT.
0076 *
0077 0000 ORG H'0500'
0078 0500 7640 STRT PPSU F SET FLAG TO SWITCH OFF THE LINE
0079 0502 7525 CP5L OVF+C+IDC
0080 0504 12 TEST SPSU WAIT FOR START BIT
0081 0505 1A7D BCTR,N TEST
0082 0507 060C LODI,R2 H'03' SET R2 TO NUMBER OF SAMPLES
0083 0509 0505 SAMP LODI,R1 3D24 SET R1 TO SAMPLE DELAY
0084 050B F37E BARR,R1 $
0085 050D 12 SPSU TEST FOR START BIT VALIDITY
0086 050E 1A74 BCTR,N TEST IF NOT VALID, GO BACK TO TEST
0087 0510 FA77 BARR,R2 SAMP
0088 0512 0608 LODI,R2 D88 SET R2 TO NUMBER OF DATA BITS
0089 0514 7440 CPSU F GENERATE START BIT
0090 0516 51 BITS RRR,R1
0091 0517 3B0C BSTR,UN B0LY GO TO DELAY AND ECHO ROUTINE
0092 0519 FA7B BARR,R2 BITS TEST FOR NUMBER OF DATA BITS
0093 051B 01 LOD2 R1
0094 051C C2 STR2 R2 LOAD R2 WITH CHARACTER
0095 051D 0700 STOP LODI,R3 0 CLEAR R3
0096 051F 3B04 BSTR,UN B0LY
0097 0521 16 EX11 RETC,N TEST STOP BIT LEVEL
0098 0522 7720 PPSL IDC IF WRONG, SET IDC BIT
0099 0524 17 EX12 RETC,UN
0100 *
0101 *****
0102 * BIT DELAY AND ECHO SUBROUTINE
0103 *
0104 0525 0425 B0LY LODI,R0 BR24 SET R0 TO BIT DELAY NUMBER
0105 0527 F87E BARR,R0 $
0106 0529 12 SPSU TEST DATA BIT LEVEL
0107 052B 1A04 BCTR,N ONE
0108 052C 7440 CPSU F IF LOW, ECHO A ZERO
0109 052E 1B04 BCTR,UN BIT1
0110 0530 7640 ONE PPSU F IF HIGH, ECHO A ONE
0111 0532 6500 IORI,R1 BP8 INSERT DATA BIT INTO R1
0112 0534 C3 BIT1 STR2 R3
0113 0535 17 RETC,UN
0114 *
0115 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000
```

Figure 9

BAUD RATE	SAMPLE DELAY NUMBER AT 1.25MHz	BIT DELAY NUMBER AT 1.25MHz	NUMBER OF BDRR,R0 INSTRUCTIONS AT 1.25MHz	NUMBER OF BDRR,R0 INSTRUCTIONS AT 1MHz
110	D0	E5	5	4
300	4A	C5	2	2
600	24	DE	1	1
1200	11	6A	1	1
2400	07	30	1	1

Table 2 BIT DELAY PROGRAM CONSTANTS AT A CLOCK FREQUENCY OF 1.25MHz (HEXADECIMAL)

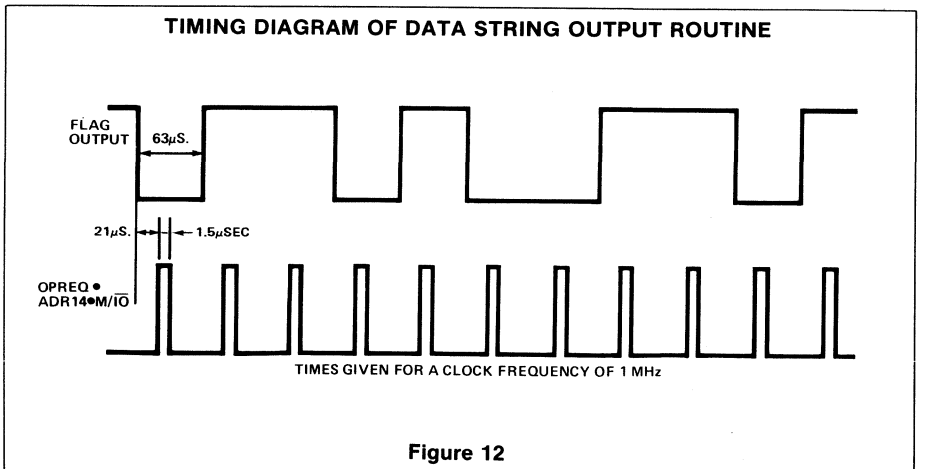
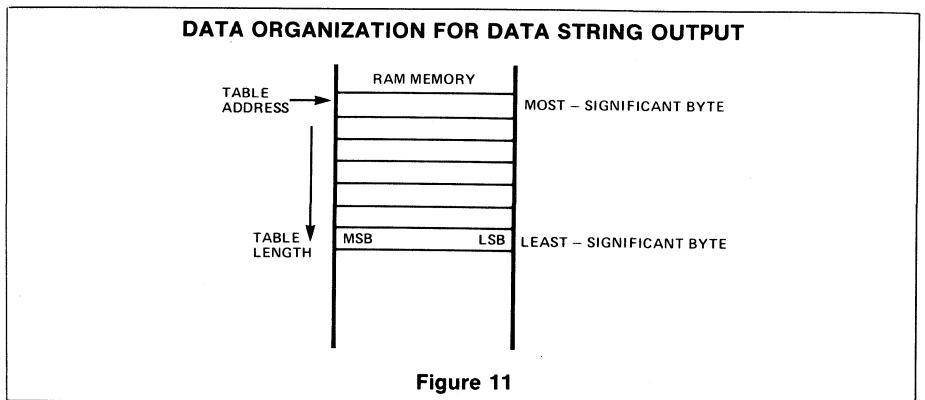
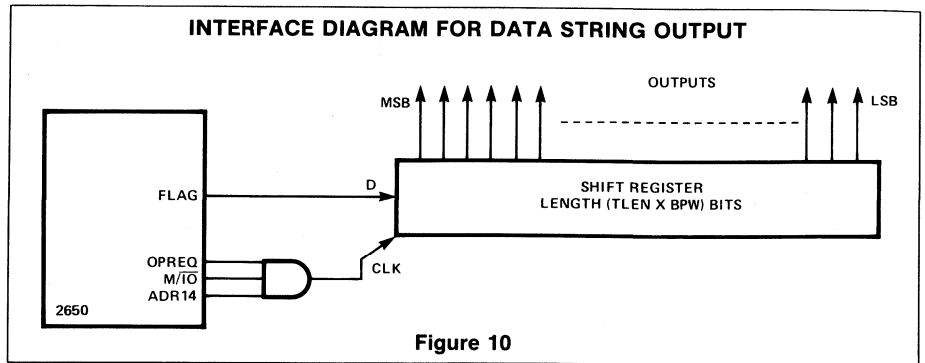
Data String Output

A typical application for the flag output is a data string output. The advantage of this output method is that it can provide a large number of output bits with little address or control logic decoding. For example, this method can be used to output data for an array of numeric displays, single bit indicators, or column drivers of a parallel numeric printer. An example of the hardware required to implement this type of output channel is given in Figure 10.

Here, the Address 14 output is used as a data strobe signal. However, the data strobe signal could also be built up by decoding more address bits so that the system memory size would not be limited to 16K bytes as in this example.

A listing of the program required is given in Figure 14. The data is assumed to be located in the system's RAM as illustrated in Figure 11.

The least-significant bit of the least-significant byte will be output first. The table length (TLEN) and the number of bits per byte (BPW) can be adapted as necessary by software modifications. The data strobe pulse on output ADR14 is generated by doing the dummy instruction STRA,R0 to address H'4000'.



FLOWCHART OF DATA STRING OUTPUT ROUTINE

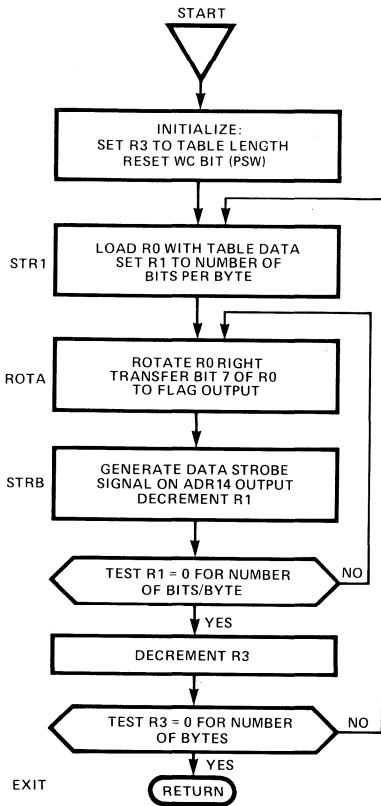


Figure 13

ASSEMBLY LISTING OF DATA STRING OUTPUT ROUTINE

```

TWIN ASSEMBLER VER 1.0                                PAGE 0001
LINE ADDR OBJECT E SOURCE
0001          * P0760094
0002          *
0003          * *****
0004          *
0005          * **** DATA STRING OUTPUT ROUTINE ****
0006          *
0007          * THIS PROGRAM TRANSFERS THE CONTENTS OF A MEMORY TABLE IN BIT BY
0008          * BIT SERIAL FORM TO THE FLAG OUTPUT OF THE 2650.
0009          *
0010          * THE TABLE LENGTH AND THE NUMBER OF BITS ARE SOFTWARE PROGRAMMED.
0011          *
0012          * A DATA STROBE OUTPUT IS GENERATED ON THE ADDRESS 14 OUTPUT
0013          *
0014          * *****
0015          *
0016          * DEFINITIONS OF SYMBOLS:
0017          *
0018          * R0 EQU 0      PROCESSOR REGISTERS
0019          * R1 EQU 1
0020          * R2 EQU 2
0021          * R3 EQU 3
0022          * S EQU H'80'  FSU: SENSE
0023          * F EQU H'40'  FLAG
0024          * WC EQU H'08'  FSL: 1=WITH, 0=WITHOUT CARRY
0025          * N EQU 2      BRANCH COND: NEGATIVE
0026          * UN EQU 3      UNCONDITIONAL
0027          *
0028          * TLEN EQU H'07'  TABLE LENGTH
0029          * BFN EQU H'08'  NUMBER OF BITS PER BYTE
0030          *
0031          * ORG H'0600'
0032          *
0033          * *****
0034          *
0035          * ORG H'0500'
0036          * STRT LODI,R3 TLEN
0037          * CPFL WC
0038          * STR1 LODR,R0 TABL,R3  LOAD R0 WITH TABLE DATA
0039          * LODI,R1 BFN        SET R1 TO NUMBER OF BITS PER BYTE
0040          * ROTR R0,R0
0041          * BCTR,N ONE        TEST BIT
0042          * ZERO CPFL F        IF ZERO, RESET FLAG
0043          * BCTR,UN STRB
0044          *
0045          * ADR DATA H'40,00'
0046          *
0047          * ONE PPSU F        IF ONE, SET FLAG
0048          * STRB STRR,R0 *ADR  GENERATE STROBE SIGNAL ON RL4
0049          * B0RR,R1 ROTA      TEST FOR NUMBER OF BITS
0050          * B0RR,R3 STR1      TEST FOR NUMBER OF BYTES
0051          * EXIT RETC,UN
0052          * END 0
TOTAL ASSEMBLY ERRORS = 0000
    
```

Figure 14

PARALLEL INPUT/OUTPUT

The 2650 instruction set contains the following six input/output instructions:

		NO. BYTES
WRTC, RX	Write Control	1
REDC, RX	Read Control	1
WRD, RX	Write Data	1
REDD, RX	Read Data	1
WRTÉ, RX DEVA	Write Extended	2
REDE, RX DEVA	Read Extended	2

The control signals generated by each I/O instruction simplify the interface circuitry required to generate I/O selection and timing signals. A low-cost control signal interface with related timing is shown in Figures 15 and 16.

When using standard TTL and 8T series I/O ports, the I/O operations can be done without slowing down the system. In this case the OPACK input could be controlled directly for all I/O operations.

Non-Extended I/O

The single-byte I/O instructions of the 2650 are referred to as non-extended I/O. In small systems with only two 8-bit input ports and two 8-bit output ports, this I/O facility requires a minimum of hardware interfacing between the CPU and I/O ports. The signals WRTC, WRD, REDC, and REDD generated by the control logic decoder in Figure 15 can be used directly as output port clock pulses and input port enable signals, respectively.

Sequential I/O With Non-Extended I/O Instructions

In systems where a larger number of devices must be serviced in sequence, the use of a simple 8-bit output port can offer considerable savings in software. Normally the devices could be serviced with extended I/O instructions. However, since the device address is the second byte in this type of instruction, a series of data fetch and I/O instructions would be required to service the devices in sequence.

With an 8-bit output port functioning as a device address register, the device address can be modified under software control. In this way, a simple program loop can serve up to 8 I/O ports by rotating a single '1' through a CPU register that is output as a device address. This I/O addressing technique may also be used advantageously in systems where I/O operation requests are detected by software polling. A functional block diagram of this technique is shown in Figure 17.

Extended I/O

There are 2 extended I/O instructions in the 2650 instruction set. In these 2-byte instructions, the first byte specifies the operation code and the data source or destination register in the CPU. The second byte provides an 8-bit device address code that is output on the 8 least-significant bits of the address bus, ADR0 through ADR7.

The control signal decoding diagram (Figure 15) can be simplified for systems using only extended I/O, as shown in Figure 18. The timing diagram of Figure 16 also applies to this decoding technique.

Device Address Decoding Schemes

For extended I/O it is necessary to decode the address lines ADR0 through ADR7 in order to generate appropriate port selection signals. The choice of an address decoding scheme depends on factors such as total

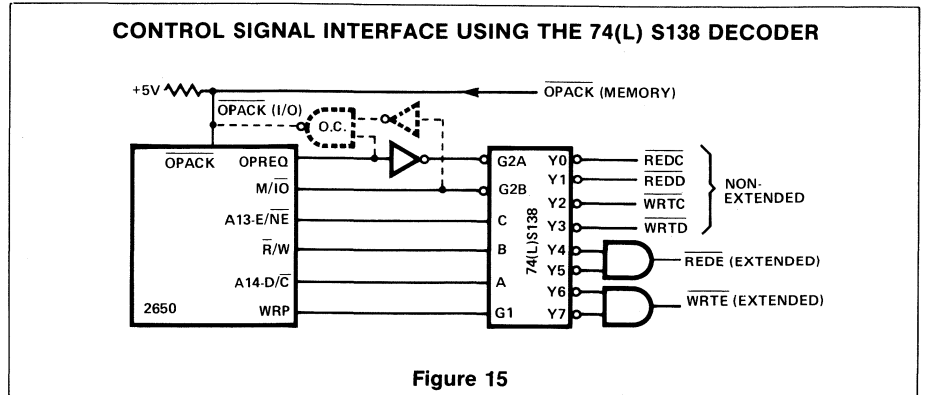


Figure 15

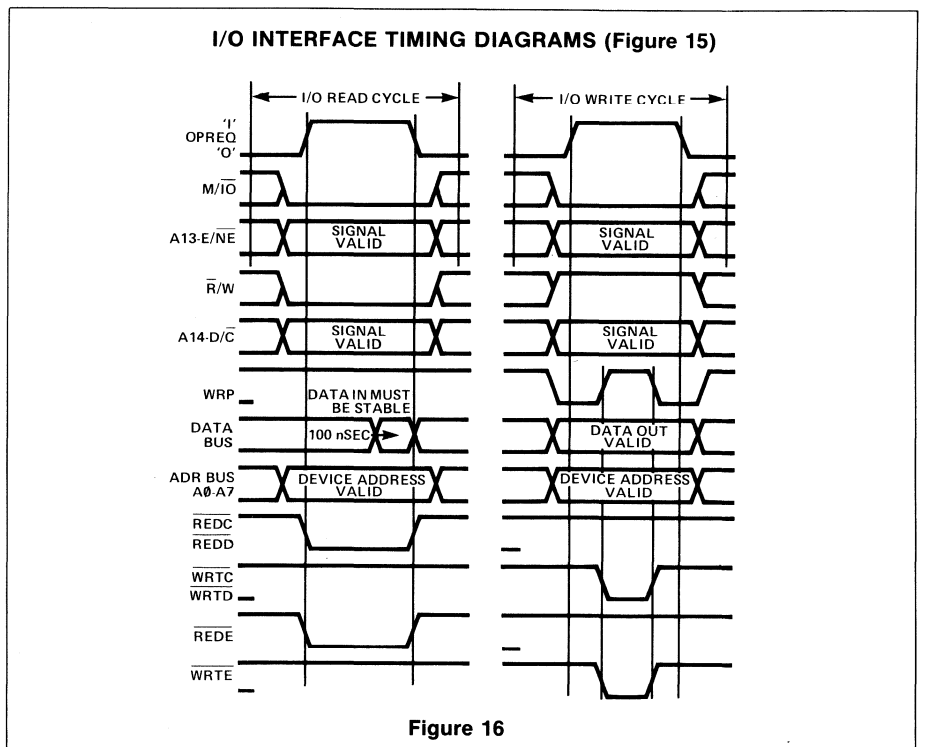


Figure 16

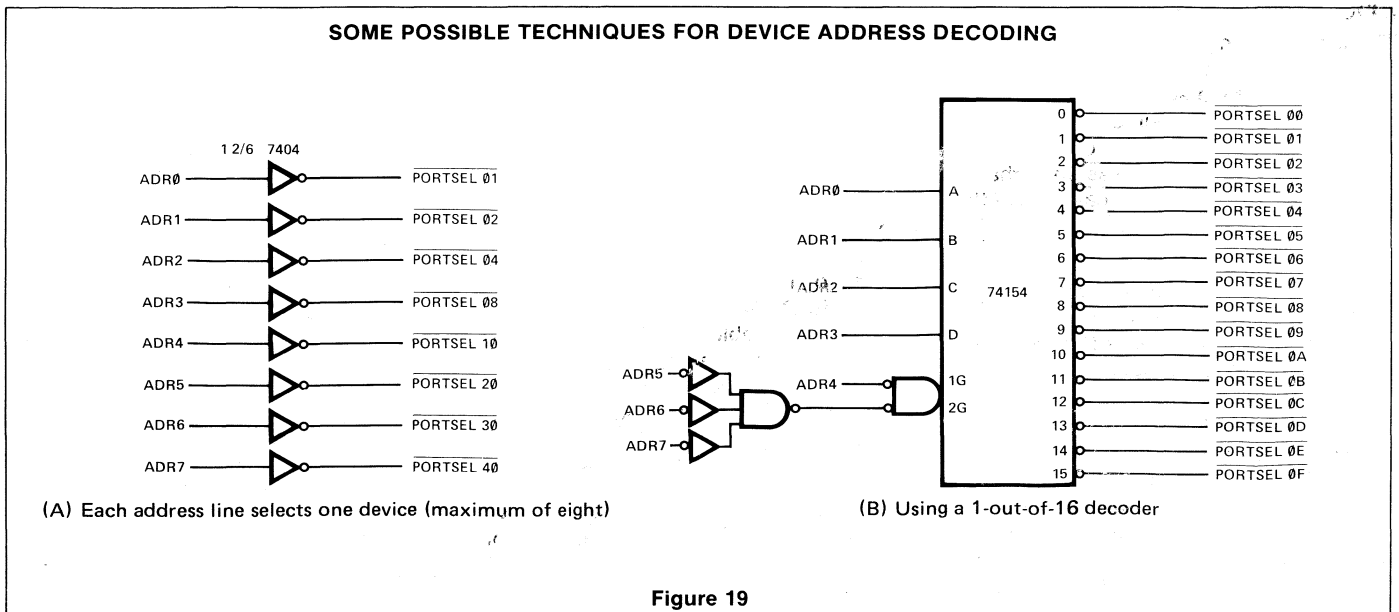
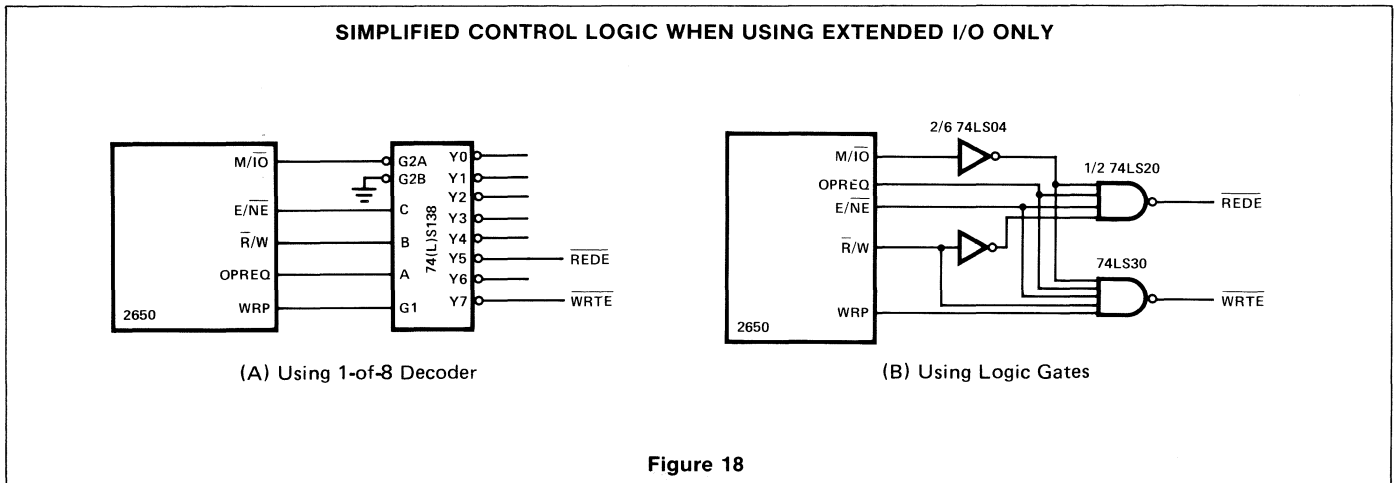
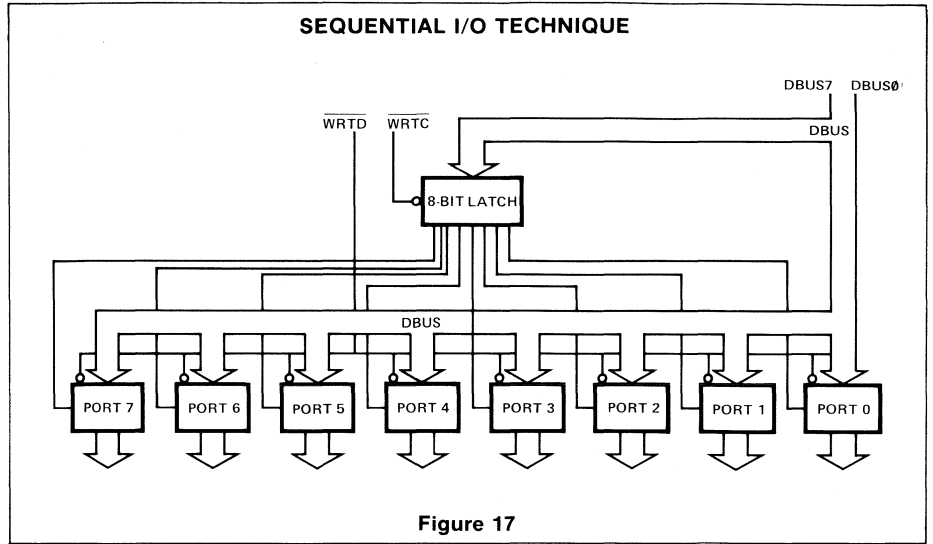
I/O requirements, the type of I/O ports used, and the total system configuration.

In principle, there are 2 basic methods of device address decoding. One method is the use of hardwired logic in which the device address is fixed; the other is a hardware programmable method in which the device addresses are individually set with jumpers or switches. Some examples of these methods are given in Figures 19 and 20.

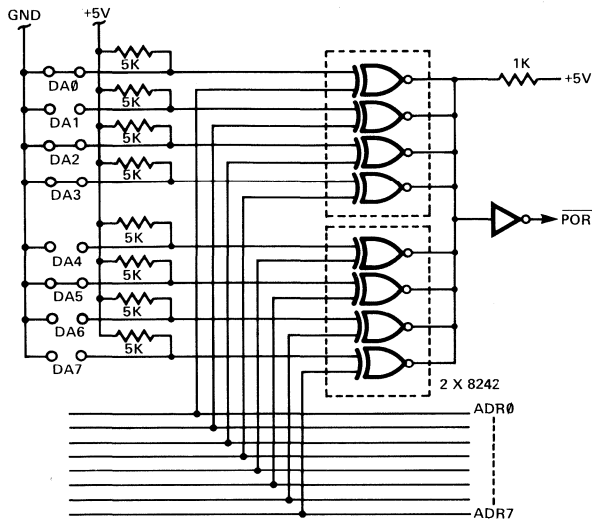
In many applications a combination of these 2 methods is used. In addition, the control logic can be implemented as an integral part of the device address decoding. An example is shown in Figure 21.

Memory Mapped I/O

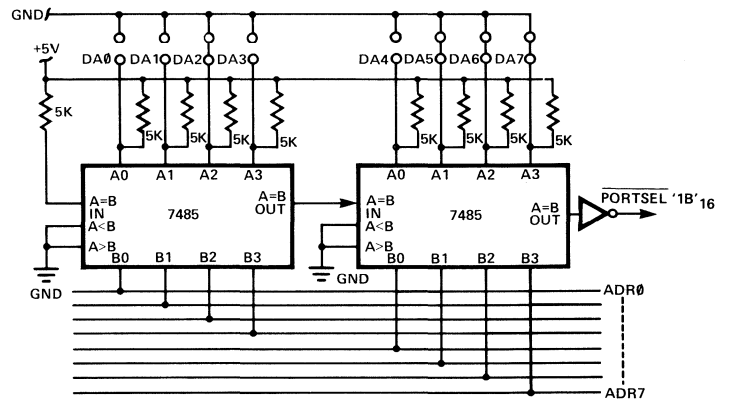
In memory mapped I/O, the I/O devices are treated as memory locations. An advantage of this technique is that all memory referencing instruction types (store, load, arithmetic, logical, etc.) can be used directly for I/O data. Device address decoding is not necessarily more complex than for normal extended I/O, since all I/O addresses could be located in a specific address block. Of course, this technique can only be used in systems which do not use the full memory address space for programs. A diagram of the I/O control logic, using the ADR14 output to discriminate between memory and I/O operations, is given in Figure 22. The device address decoding methods described earlier can also be applied to memory mapped I/O.



HARDWARE PROGRAMMABLE DEVICE ADDRESS DECODERS



(A) Using Exclusive-OR Gates



(B) Using Comparators

Figure 20

COMBINED CONTROL LOGIC AND ADDRESS DECODING

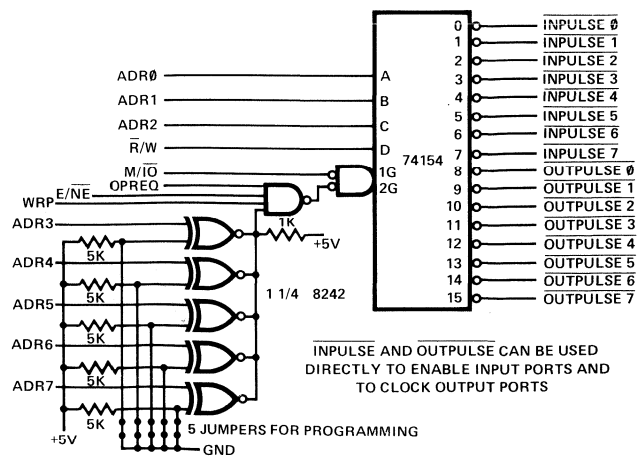


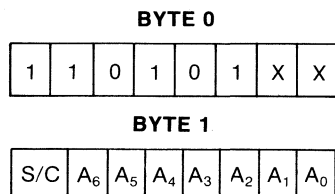
Figure 21

SINGLE POINT CONTROL

In many applications, the capability to set, clear, or test a single output point selected from a large number of output points is required. Designs of this type can be implemented using the 2650 I/O instructions. When used as described below, the WRTE, WRTC, and WRTD instructions become "set/clear single-bit" instructions, while the REDE instruction becomes a "test single-bit" instruction.

Single Bit Output—Direct Address

The write extended instruction can be used to select and set or clear a single output bit. The 2 bytes of the instruction can be interpreted as follows:



A₀ through A₆ of the second byte specify the output selected. The S/C bit specifies whether the bit is set or cleared. A typical hardware configuration controlling 64 points is shown in Figure 23. Here, the control line decoding and partial address decoding is done by the 74LS138, which selects one of the eight 9334s. One of the 8 latches in the selected 9334 is enabled by ADR₀, ADR₁, and ADR₂ and is either cleared or set, as determined by the value of ADR₇.

The XX field in the first byte selects 1 of the 4 available registers and outputs its contents on the data bus. Since this information is not used in this application, the value of XX is not important. However, it could be used to output an 8-bit control or status word in conjunction with the set/clear operation.

Single Bit Output—Indirect Address

If the address of the output to be set or cleared must be determined at program run time, the WRTD and WRTC instructions can be used. The address of the output bit is first loaded into one of the 2650 registers. A WRTD, Rx instruction is then issued if the bit is to be set, and a WRTC, Rx instruction is issued if the bit is to be cleared. The bit select is output on the data bus, and the D/C output carries the set/clear information. The hardware implementation can be the same as shown in Figure 23, except that ADR₀-ADR₅ are replaced by DBUS₀-DBUS₅, and ADR₇ is replaced by D/C.

I/O CONTROL SIGNAL GENERATION FOR MEMORY MAPPED I/O

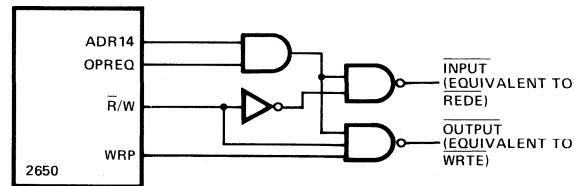


Figure 22

SIXTY-FOUR SINGLE BIT OUTPUTS USING THE 9334

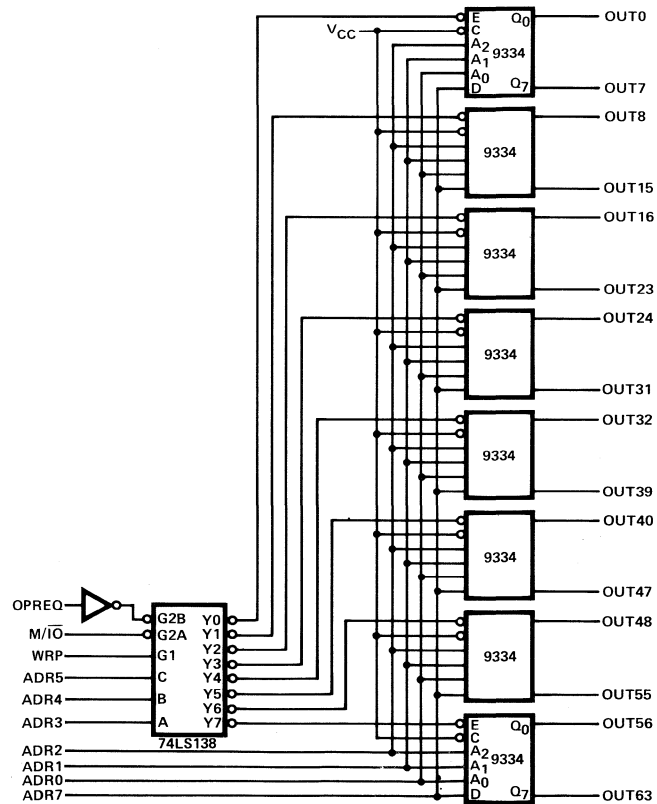


Figure 23

Single Bit Input

One way of doing single bit input uses the techniques described earlier. The address of the bit that is to be tested is loaded into one of the 2650 registers and output to an 8-bit latch using an extended or non-extended write instruction. The latch output is decoded to select the desired bit, which is then applied to the Sense input pin. The 2650 Program Status Word instructions can then be used to test the state of the Sense input and to take appropriate program action.

The technique described above must be used if "indirect" bit addressing is required. If this is not a requirement, a more efficient implementation can be accomplished using the extended read instruction. This technique makes use of the fact that the 2650 automatically tests the contents of a register every time it is used as the destination of an operation. Thus, when the read extended operation reads data from an input port, the condition code bits in the program status word are set to reflect whether the new

register contents is positive, negative, or zero.

For the single bit input application, the second byte of the RETE, Rx instruction contains the address of the input bit to be tested. This data is applied to a bank of data selectors to select the addressed bit, which is then applied to the most-significant bit of the data bus, DBUS7. Since this is interpreted as the sign bit, the condition code bits in PSL will be set to reflect whether the bit being tested is a one or a zero. A conditional branch instruction can then be used to affect the desired program action. A hardware implementation for 64 inputs is shown in Figure 24. Note that an address latch is not required for this method.

INPUT PORT DEVICES

Gated Input Ports

The simplest form of an input port is the tri-state gate. Figure 25 illustrates the use of the 8T97 high-speed hex tri-state buffer for gated input ports. The 8T97 is non-inverting, and the tri-state control signals enable the buffers in groups of 4 and groups of 2, so that 8-bit ports can be implemented efficiently.

An effective circuit for systems using 8-gated input ports is the 74251 8-to-1 multiplexer, which has tri-state outputs that can interface directly with the data bus. The advantage of this circuit is that no external address decoding logic is needed. A configuration using gated input ports with the 74251 multiplexer is illustrated in Figure 26.

In addition to these 2 configurations, many other input port configurations are possible using standard TTL or Signetics 8T series logic circuits.

Latching Input Ports

Latching input ports may be required to store data from an external device, which is available only momentarily, before the actual input operation to the microprocessor takes place. This type of input port can be realized by connecting TTL-latch or D-type flip-flop circuits, such as the 7475, 74100, or 74175, to the inputs of a gated input port. As illustrated in Figure 27, by using the Signetics 8T10 Quad D-type flip-flop with tri-state outputs, an 8-bit latching input port can be implemented with only 2 packages. The 8T10 is functionally identical to the 74173.

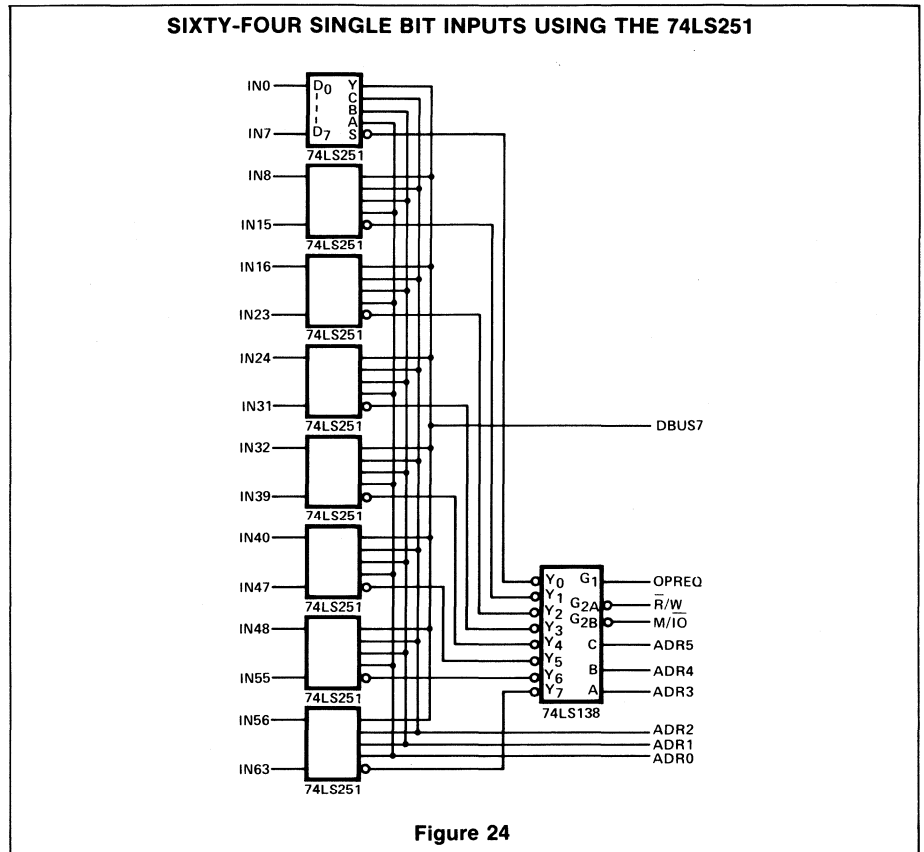


Figure 24

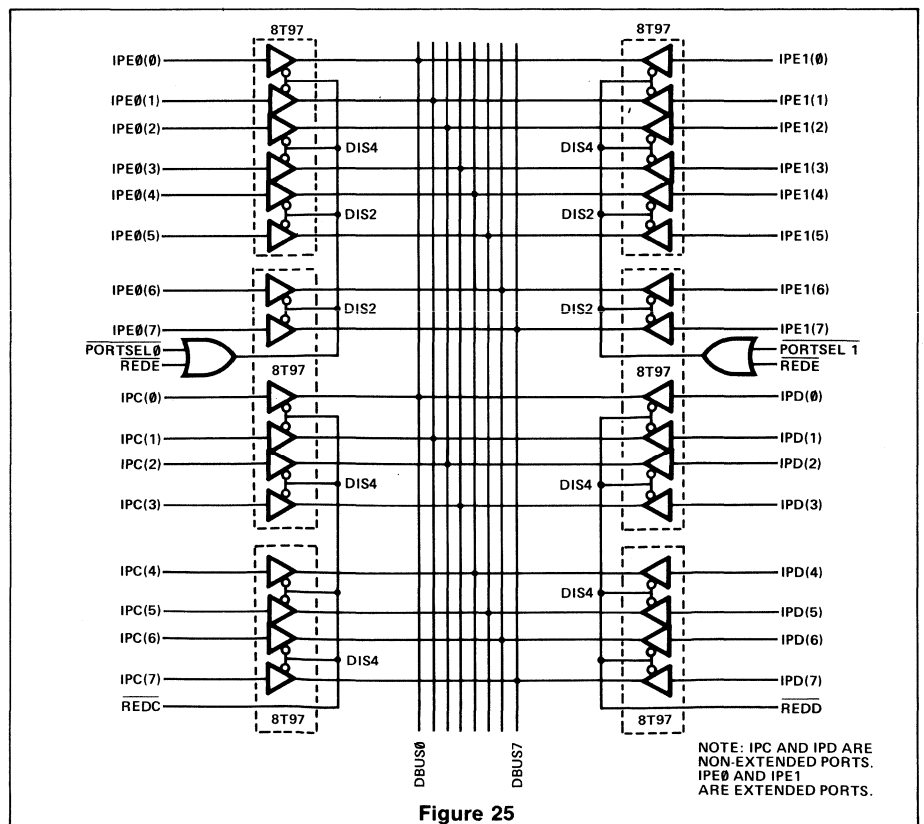
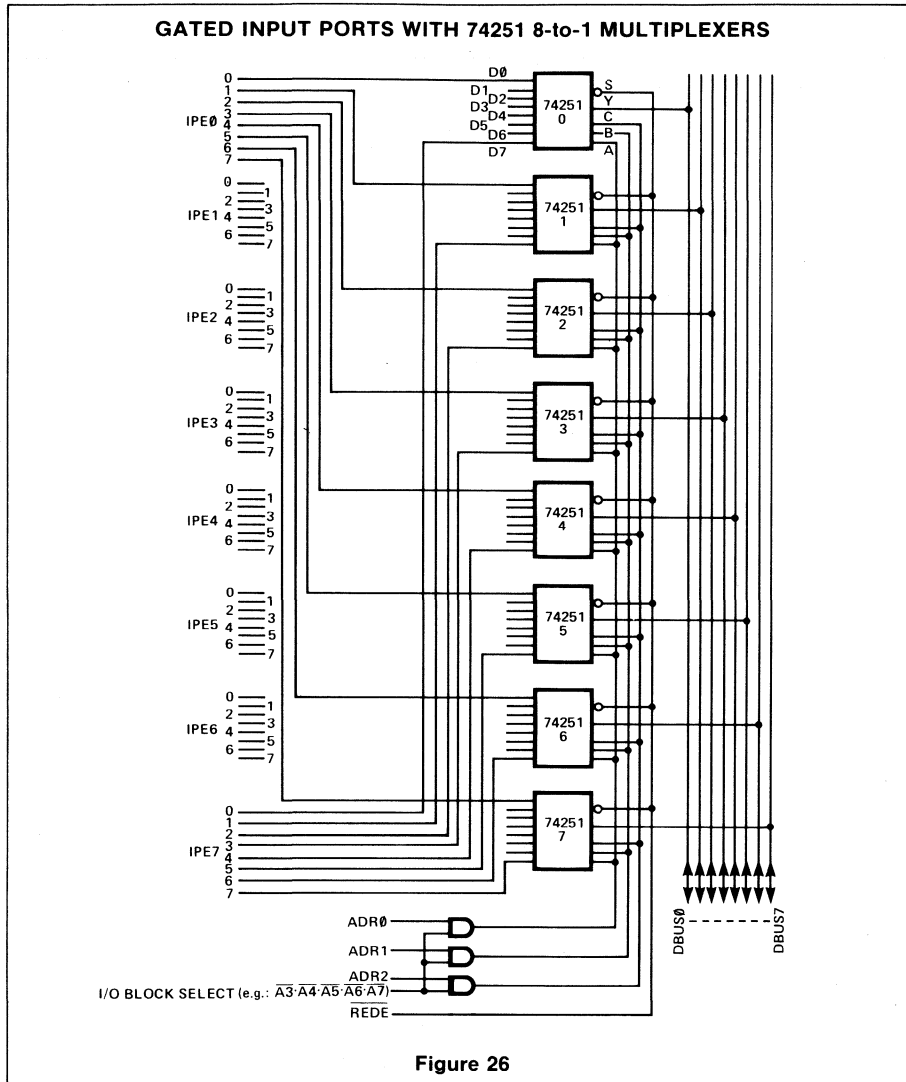


Figure 25



OUTPUT PORT DEVICES

Output ports can be configured with a variety of standard TTL and 8T series flip-flops and registers. Typical circuits include:

- 9334 Addressable 8-bit latch
- 7475 Quadruple latch
- 74100 8-bit latch
- 74175 Quadruple D-type flip-flop
- 8T10 Quadruple D-type flip-flop with tri-state outputs

The 7475 and 74175 both have true and complement outputs. One special feature of the 8T10 is that the outputs may be disabled (placed in a high-impedance output mode) by the device that is connected to this output port. A logic diagram using these circuits for output ports appears in Figure 28.

The 9334 is useful in systems requiring a large number of latched outputs, since a portion of the decoding can be done using the on-chip 3-input decoder. A typical application of this was shown in Figure 23. It is also an efficient circuit for implementing eight 8-bit output ports.

I/O CONFIGURATIONS USING THE 8T31 BIDIRECTIONAL PORT

Functional Description

The 8T31 is an 8-bit bidirectional I/O port consisting of 8 clocked latches with 2 bidirectional I/O buses, each of which has its own control logic. Each bus (A and B) has a read and a write control input, and there is a master enable input for bus B only. The outputs of the latches follow the inputs when the clock is high, and latching will occur when the clock returns low.

The 8T31 is also equipped with a "power-on clear" circuit. If the clock input is held low until the power supply reaches 3.5 volts, the latches will be cleared. There is a logic inversion between bus A and bus B. As a result, when the 8T31 is cleared, bus A will have all logic "1" outputs and bus B all logic "0" outputs.

The control functions of the 8T31 are listed in Table III. A functional block diagram of the 8T31 are illustrated in Figures 29 and 30, respectively.

LATCHING INPUT PORTS USING THE 8T10 (74173)

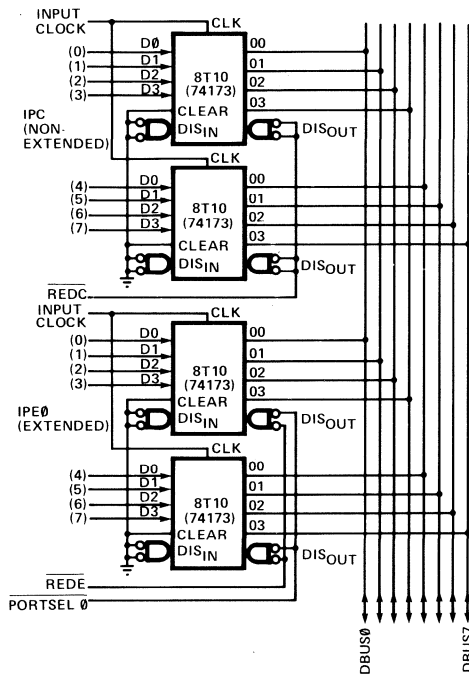


Figure 27

OUTPUT PORTS BUILT WITH STANDARD TTL AND 8T SERIES

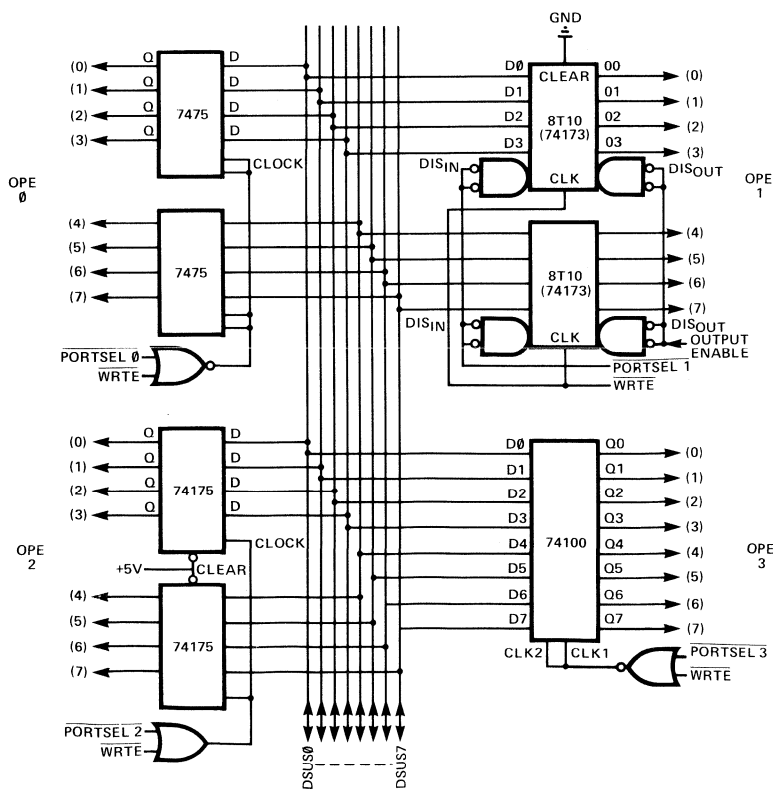


Figure 28

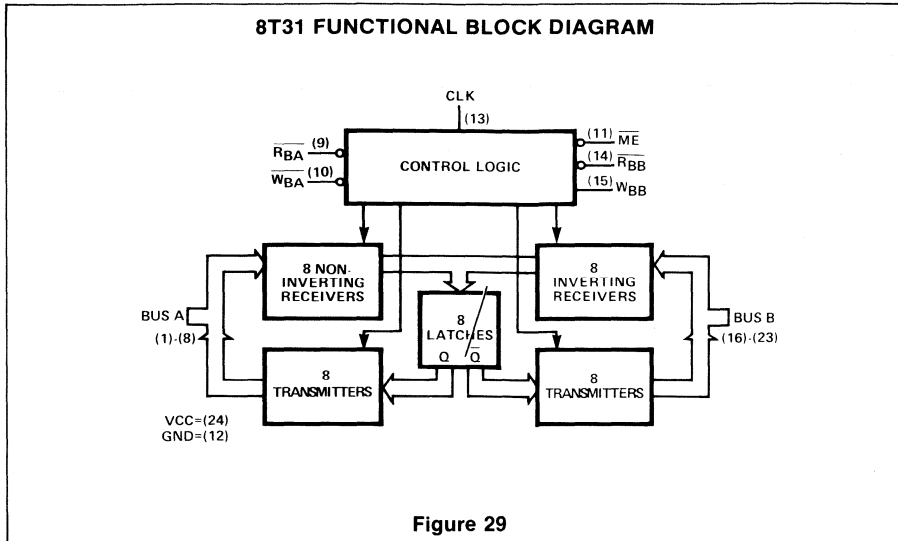


Figure 29

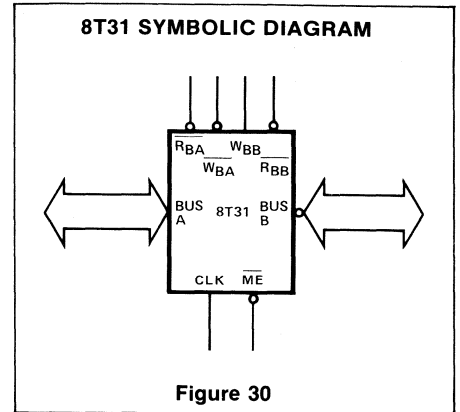


Figure 30

BUS A					
RBA	WBA	CLK	BUS A		
X	0	1	WRITE (A → latch)		
0	1	X	READ (latch → A)		
1	1	X	HI-Z (Tri-state)		
BUS B					
RBB	WBB	WBA	CLK	ME	BUS B
X	X	X	X	1	HI-Z
1	0	X	X	0	HI-Z
X	1	0	X	0	HI-Z
0	0	X	X	0	READ (latch → B)
X	1	1	1	0	WRITE (B → latch)

Table 3 8T31 CONTROL FUNCTIONS

As shown in Table III, each bus can operate independently except for the case of writing from both bus A and B. In this case writing from bus A will override any attempt to write from bus B.

8T31 Applications

The control functions of the 8T31 allow it to be used in various microcomputer input/output applications. In the I/O system diagram of Figure 31, the 8T31 is used to implement gated input ports, latching input ports, output ports, and a bidirectional data bus driver. All I/O ports can be controlled directly with the device select and REDE and WRTE lines coming from device decoders and I/O control logic.

In applications where interfacing is necessary with peripheral devices that need data transfers in two directions, like digital cassettes and data link communication circuits, the 8T31 can be used as a bidirectional I/O port. In this application, the I/O opera-

tion should be requested by interrupt or polling to prevent simultaneous write operations from peripheral and CPU. The bidirectional I/O port concept is illustrated in Figure 32.

Implementing an Eight-Bit Flag Register with the 8T31

In many industrial applications, such as process control, single bit inputs and outputs are used to monitor switches and detectors or to drive relays and lamps. A possible solution for such a flag register would be an eight-bit output port and a memory byte reserved as a flag register in the system's RAM. The setting, resetting, or testing of individual bits with this method of implementing a flag register requires many bytes of program memory. The output port and the memory location reserved as a flag register image must be updated after each bit operation.

The 8T31 can be used to implement a flag register without the use of a memory byte in the system's RAM. No additional hardware is required, and the saving in program memory bytes for flag operations is considerable. A logic diagram of this application is given in Figure 33. Listings of basic software to set, reset, and test individual flags for both positive and negative true outputs are given in Figures 34 and 35.

THE 8T31 USED AS A GATED INPUT PORT,
LATCHING INPUT PORT, OUTPUT PORT, AND DATA BUS DRIVER

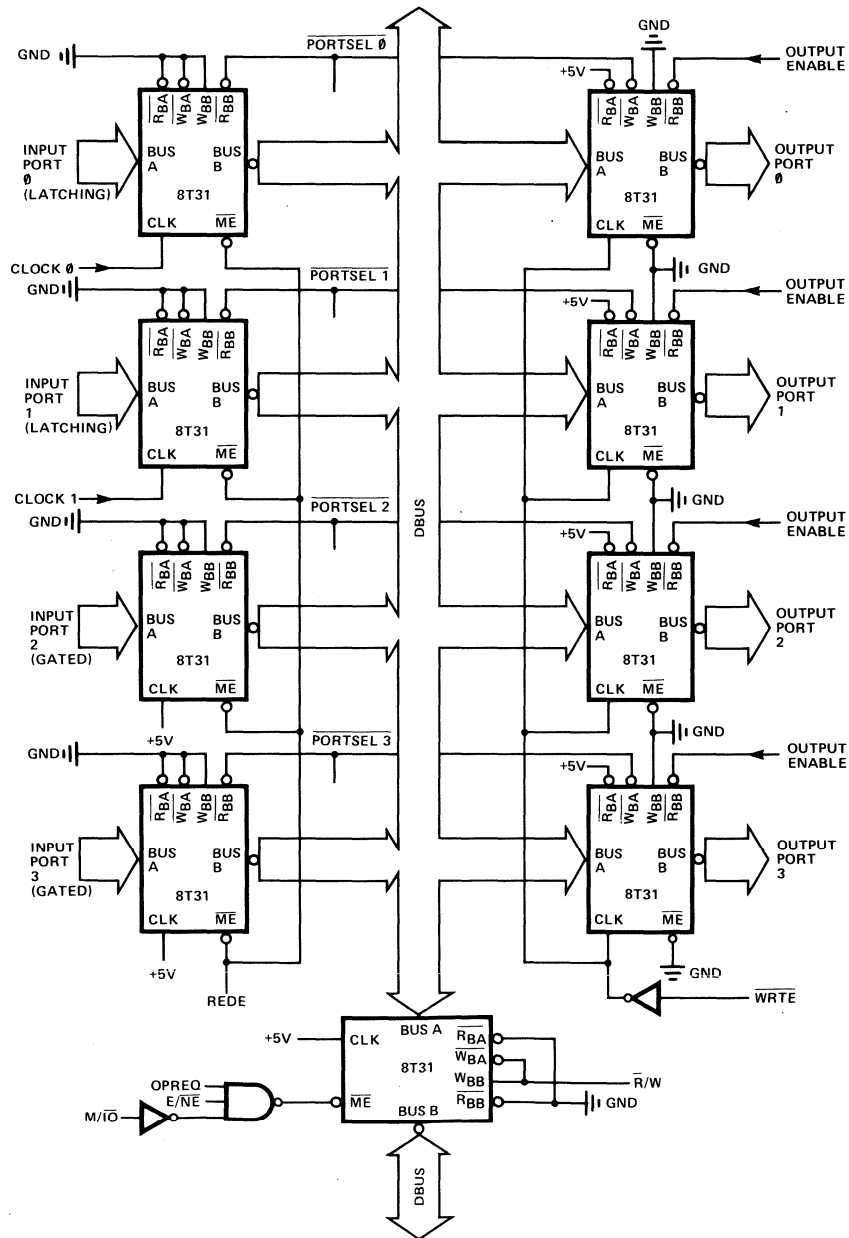


Figure 31

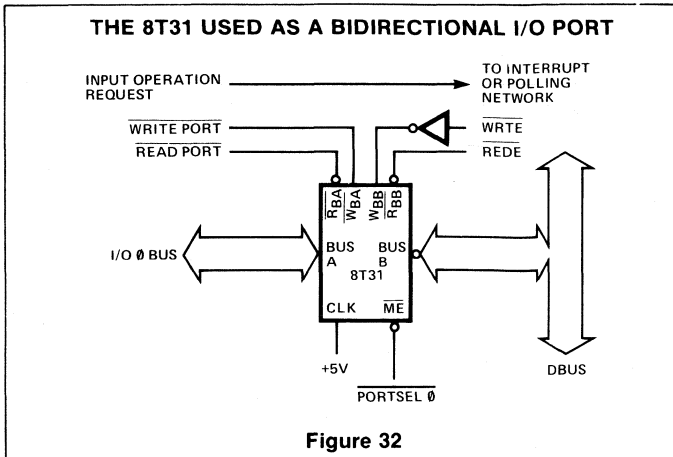


Figure 32

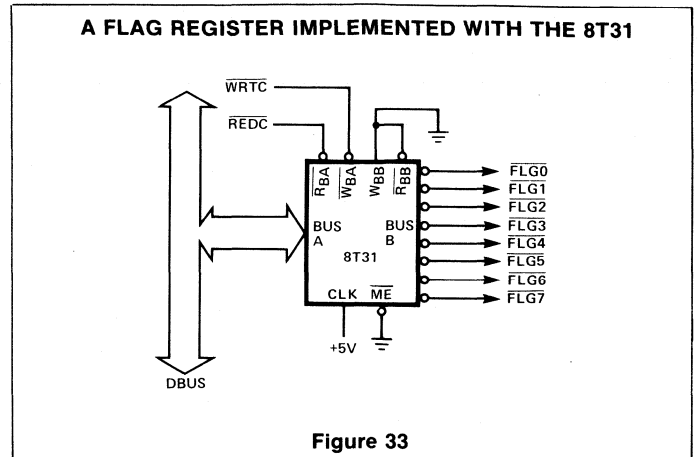


Figure 33

BASIC SOFTWARE FOR FLAG REGISTER OPERATIONS

```

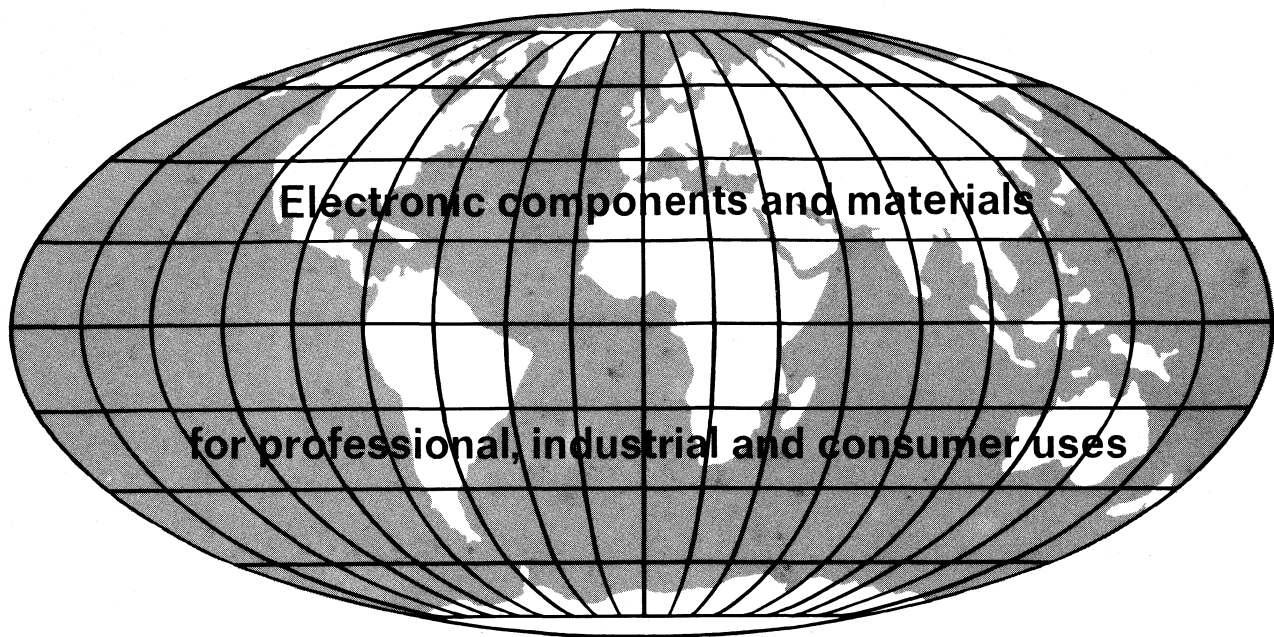
TWIN ASSEMBLER VER 1.0                                PAGE 0001
LINE ADDR  OBJECT  E SOURCE
0001          * PD760090
0002          *****
0003          *
0004          * **** FLAG MANIPULATION EXAMPLES ****
0005          *
0006          * THIS LISTING GIVES SOME EXAMPLES HOW TO SET, RESET
0007          * AND TEST INDIVIDUAL BITS OF AN EXTERNAL FLAG REGISTER *
0008          * BUILT WITH THE 8T31 BIDIRECTIONAL I/O PORT.
0009          * INSTRUCTIONS ARE GIVEN FOR BOTH ACTIVE 'HIGH' AND
0010          * ACTIVE 'LOW' OUTPUTS.
0011          *
0012          *****
0013          *
0014          * DEFINITIONS OF SYMBOLS:
0015          *
0016 0000      R0 EQU 0          PROCESSOR REGISTERS
0017 0001      R1 EQU 1
0018 0002      R2 EQU 2
0019 0003      R3 EQU 3
0020 0000      Z EQU 0          BRANCH COND. ZERO UNCONDITIONAL
0021 0003      UN EQU 3
0022 0000      AL EQU 0          ALL BITS ARE 1
0023          *
0024 0001      FLAG EQU 'H'01'   FLAG 0
0025 0002      FLG1 EQU 'H'02'   FLAG 1
0026 0004      FLG2 EQU 'H'04'   FLAG 2
0027 0008      FLG3 EQU 'H'08'   FLAG 3
0028 0010      FLG4 EQU 'H'10'   FLAG 4
0029 0020      FLG5 EQU 'H'20'   FLAG 5
0030 0040      FLG6 EQU 'H'40'   FLAG 6
0031 0080      FLG7 EQU 'H'80'   FLAG 7
0032          *
0033 0600      ONE EQU 'H'0600'   DUMMY ADDRESS OF ROUTINE 'ONE'
0034 0650      ONES EQU 'H'0650'  DUMMY ADDRESS OF ROUTINE 'ONES'
0035          *
0036          *****
0037          *
0038          * ***INSTRUCTIONS FOR ACTIVE 'LOW' OUTPUTS***
0039          *
0040 0000      * ORG 'H'0500'
0041          *
0042          * SET FLAG(S)
0043          *
0044 0500 30   SNFG REDC,R0      LOAD FLAG REGISTER IN R0
0045 0501 6404 IORI,R0 FLG2     SET FLAG 2
0046 0503 B0   WRTC,R0        RESTORE FLAG REGISTER
0047          *
0048 0504 30   SNFS REDC,R0
0049 0505 6460 IORI,R0 FLG5+FLG6 SET FLAGS 5 AND 6
0050 0507 B0   WRTC,R0        RESTORE
0051          *
0052          * RESET FLAG(S)
0053          *
0054 0508 30   RNFG REDC,R0
0055 0509 44FB ANDI,R0 H'FF'-FLG2 RESET FLAG 2
0056 050E B0   WRTC,R0        RESTORE
    
```

Figure 34

```

TWIN ASSEMBLER VER 1.0                                PAGE 0002
LINE ADDR  OBJECT  E SOURCE
0058 050C 30   RNFS REDC,R0
0059 050D 449F ANDI,R0 H'FF'-FLG5-FLG6 RESET FLAGS 5 AND 6
0060 050F B0   WRTC,R0        RESTORE
0061          *
0062          * TEST FLAG(S)
0063          *
0064 0510 30   TNFG REDC,R0
0065 0511 F404 TMI,R0 FLG2     TEST FLAG 2
0066 0513 1C0600 BCFR,AL ONES   BRANCH IF ONE
0067          *
0068 0516 30   TNFS REDC,R0
0069 0517 F460 TMI,R0 FLG5+FLG6 TEST FLAGS 5 AND 6
0070 0519 1C0650 BCFR,AL ONES   BRANCH IF BOTH ARE ONE
0071          *
0072          *****
0073          *
0074          * ***INSTRUCTIONS FOR ACTIVE 'HIGH' OUTPUTS***
0075          *
0076 051C      ORG 'H'0550'
0077          *
0078          * SET FLAG(S)
0079          *
0080 0550 30   SPFG REDC,R0
0081 0551 44FB ANDI,R0 H'FF'-FLG2 SET FLAG 2
0082 0553 B0   WRTC,R0        RESTORE
0083          *
0084 0554 30   SPFS REDC,R0
0085 0555 44ED ANDI,R0 H'FF'-FLG1-FLG4 SET FLAGS 1 AND 4
0086 0557 B0   WRTC,R0        RESTORE
0087          *
0088          * RESET FLAG(S)
0089          *
0090 0558 30   RPFG REDC,R0
0091 0559 6404 IORI,R0 FLG2     RESET FLAG 2
0092 055B B0   WRTC,R0        RESTORE
0093          *
0094 055C 30   RPPFS REDC,R0
0095 055D 6412 IORI,R0 FLG1+FLG4 SET FLAGS 1 AND 4
0096 055F B0   WRTC,R0        RESTORE
0097          *
0098          * TEST FLAG(S)
0099          *
0100 0560 30   TPFG REDC,R0
0101 0561 F404 TMI,R0 FLG2     TEST FLAG 2
0102 0563 9C0600 BCFR,AL ONES   BRANCH IF ONE
0103          *
0104 0566 30   TPFS REDC,R0
0105 0567 F412 TMI,R0 FLG1+FLG4 TEST FLAGS 1 AND 4
0106 0569 9C0650 BCFR,AL ONES   BRANCH IF BOTH ARE ONE
0107          *
0108 0000      END          9
TOTAL ASSEMBLY ERRORS = 0000
    
```

Figure 35



from the world-wide Philips Group of Companies

- Argentina:** FAPESA I.y.C., Av. Crovara 2550, Tablada, Prov. de BUENOS AIRES, Tel. 652-7438/7478.
- Australia:** PHILIPS INDUSTRIES HOLDINGS LTD., Elcoma Division, 67 Mars Road, LANE COVE, 2066, N.S.W., Tel. 42 1261.
- Austria:** ÖSTERREICHISCHE PHILIPS BAUELEMENTE Industrie G.m.b.H., Triester Str. 64, A-1101 WIEN, Tel. 62 91 11.
- Belgium:** M.B.L.E., 80, rue des Deux Gares, B-1070 BRUXELLES, Tel. 523 00 00.
- Brazil:** IBRAPE, Caixa Postal 7383, Av. Paulista 2073-S/Loja, SAO PAULO, SP, Tel. 287-7144.
- Canada:** PHILIPS ELECTRONICS LTD., Electron Devices Div., 601 Milner Ave., SCARBOROUGH, Ontario, M1B 1M8, Tel. 292-5161.
- Chile:** PHILIPS CHILENA S.A., Av. Santa Maria 0760, SANTIAGO, Tel. 39-40 01.
- Colombia:** SADAPE S.A., P.O. Box 9805, Calle 13, No. 51 + 39, BOGOTA D.E. 1., Tel. 600 600.
- Denmark:** MINIWATT A/S, Emdrupvej 115A, DK-2400 KØBENHAVN NV., Tel. (01) 69 16 22.
- Finland:** OY PHILIPS AB, Elcoma Division, Kaivokatu 8, SF-00100 HELSINKI 10, Tel. 1 72 71.
- France:** R.T.C. LA RADIOTECHNIQUE-COMPELEC, 130 Avenue Ledru Rollin, F-75540 PARIS 11, Tel. 355-44-99.
- Germany:** VALVO, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, D-2 HAMBURG 1, Tel. (040) 3296-1.
- Greece:** PHILIPS S.A. HELLENIQUE, Elcoma Division, 52, Av. Syngrou, ATHENS, Tel. 915 311.
- Hong Kong:** PHILIPS HONG KONG LTD., Comp. Dept., Philips Ind. Bldg., Kung Yip St., K.C.T.L. 289, KWAI CHUNG, N.T. Tel. 12-24 51 21.
- India:** PHILIPS INDIA LTD., Elcoma Div., Band Box House, 254-D, Dr. Annie Besant Rd., Prabhadevi, BOMBAY-25-DD, Tel. 457 311-5.
- Indonesia:** P.T. PHILIPS-RALIN ELECTRONICS, Elcoma Division, 'Timah' Building, Jl. Jen. Gatot Subroto, JAKARTA, Tel. 44 163.
- Ireland:** PHILIPS ELECTRICAL (IRELAND) LTD., Newstead, Clonskeagh, DUBLIN 14, Tel. 69 33 55.
- Italy:** PHILIPS S.P.A., Sezione Elcoma, Piazza IV Novembre 3, I-20124 MILANO, Tel. 2-6994.
- Japan:** NIHON PHILIPS CORP., Shuwa Shinagawa Bldg., 26-33 Takanawa 3-chome, Minato-ku, TOKYO (108), Tel. 448-5611.
(IC Products) SIGNETICS JAPAN, LTD., TOKYO, Tel. (03) 230-1521.
- Korea:** PHILIPS ELECTRONICS (KOREA) LTD., Philips House, 260-199 Itaewon-dong, Yongsan-ku, C.P.O. Box 3680, SEOUL, Tel. 44-4202.
- Mexico:** ELECTRONICA S.A. de C.V., Varsovia No. 36, MEXICO 6, D.F., Tel. 5-33-11-80.
- Netherlands:** PHILIPS NEDERLAND B.V., Afd. Elonco, Boschdijk 525, NL-4510 EINDHOVEN, Tel. (040) 79 33 33.
- New Zealand:** Philips Electrical Ind. Ltd., Elcoma Division, 2 Wagener Place, St. Lukes, AUCKLAND, Tel. 867 119.
- Norway:** ELECTRONICA A/S., Vitaminveien 11, P.O. Box 29, Grefsen, OSLO 4, Tel. (02) 15 05 90.
- Peru:** CADESA, Jr. Ilo, No. 216, Apartado 10132, LIMA, Tel. 27 73 17.
- Philippines:** ELDAC, Philips Industrial Dev. Inc., 2246 Pasong Tamo, MAKATI-RIZAL, Tel. 86-89-51 to 59.
- Portugal:** PHILIPS PORTUGESA S.A.R.L., Av. Eng. Duharte Pacheco 6, LISBOA 1, Tel. 68 31 21.
- Singapore:** PHILIPS SINGAPORE PTE LTD., Elcoma Div., POB 340, Toa Payoh CPO, Lorong 1, Toa Payoh, SINGAPORE 12, Tel. 53 88 11.
- South Africa:** EDAC (Pty.) Ltd., South Park Lane, New Doornfontein, JOHANNESBURG 2001, Tel. 24/6701.
- Spain:** COPRESA S.A., Balmes 22, BARCELONA 7, Tel. 301 63 12.
- Sweden:** A.B. ELCOMA, Lidingsvägen 50, S-10 250 STOCKHOLM 27, Tel. 08/67 97 80.
- Switzerland:** PHILIPS A.G., Elcoma Dept., Edenstrasse 20, CH-8027 ZÜRICH, Tel. 01/44 22 11.
- Taiwan:** PHILIPS TAIWAN LTD., 3rd Fl., San Min Building, 57-1, Chung Shan N. Rd, Section 2, P.O. Box 22978, TAIPEI, Tel. 5513101-5.
- Turkey:** TÜRK PHILIPS TICARET A.S., EMET Department, Inonu Cad. No. 78-80, ISTANBUL, Tel. 43 59 10.
- United Kingdom:** MULLARD LTD., Mullard House, Torrington Place, LONDON WC1E 7HD, Tel. 01-580 6633.
- United States:** (Active devices & Materials) AMPEREX SALES CORP., 230, Duffy Avenue, HICKSVILLE, N.Y. 11802, Tel. (516) 931-6200.
(Passive devices) MEPCO/ELECTRA INC., Columbia Rd., MORRISTOWN, N.J. 07960, Tel. (201) 539-2000.
(IC Products) SIGNETICS CORPORATION, 811 East Arques Avenue, SUNNYVALE, California 94086, Tel. (408) 739-7700.
- Uruguay:** LUZILECTRON S.A., Rondeau 1567, piso 5, MONTEVIDEO, Tel. 9 43 21.
- Venezuela:** IND. VENEZOLANAS PHILIPS S.A., Elcoma Dept., A. Ppal de los Ruices, Edif. Centro Colgate, Apdo 1167, CARACAS, Tel. 36 05 11.