intel

THE GREATEST MICROCOMPUTER FAIR EVER

4th MAY 1976

intel

WELCOMES YOU TO THE MICRO COMPUTER FAIR

The Greatest Ever

The Microcomputer Fair

The Intel Microcomputer Fair consists of an exhibition and four simultaneous seminars. The seminars have been arranged so that you need only attend those papers which are of direct interest to you. Most of the papers start and finish at the same time so that you can change theatres in between papers if you wish.

The Exhibition

In the exhibition area you will be able to see the latest microcomputer equipment demonstrated and you will be able to discuss your own microcomputer hardware and software requirements with leading experts.

The Seminars

The seminars cover all topics associated with microcomputing. Papers will be read that will be of interest to the newcomer to the subject and to the engineer who already has experience of microcomputer systems. A complete seminar programme and a summary of all the papers is included in this book.

The Book

The book contains a timetable for the four seminars, a plan of the Hilton showing the location of the conference rooms and exhibition, a summary of the seminar papers, a selection of useful data sheets and technical articles, some blank paper for your notes and a form for requesting further information.

The Object

The object of the Microcomputer Fair, which is the first event of its kind, is to enable you to obtain any microcomputer information you may need in convivial surroundings. We very much hope you will have an interesting and enjoyable day.

intel Microcomputer Seminar Programme

Suite A

1)	9.30 - 10.15	Programme Management, R.A. Perrin, Computer, Analysts and Programmers Ltd.
2)	10.35 - 11.15	4040 Peripheral ICs, Tom Cubitt, GEC Semiconductors Ltd.
3)	11.30 - 12.30	OEM Computers and Microcomputer Kits, Bob Robinson, Rapid Recall Ltd.
4)	14.00 - 14.45	8080A Peripheral ICs, Tom Cubitt, GEC Semiconductors Ltd.
5)	15.00 - 15.45	Microcomputer Development Aids, Tom Cubitt, GEC Semiconductors Ltd.
6)	16.00 - 16.45	

Suite B

1)	9.30 - 10.15	Cross Products Advantages and Use, Howard Kornstein, Intel Corporation (UK) Ltd.
2)	10.35 - 11.15	Basic Programming Techniques, Phil Pittman, Rapid Recall Ltd.
3)	11.30 - 12.30	Advanced Software Techniques, Bill Betts, GEC Semiconductors Ltd.
4)	14.00 - 14.45	Handling Interrupts, Bill Betts, GEC Semiconductors Ltd.
5)	15.00 - 15.45	The Coral Compiler, Bill Betts, GEC Semiconductors Ltd.
6)	16.00 - 16.45	PL/M v Assembly Language, Phil Pittman, Rapid Recall Ltd.

Suite C

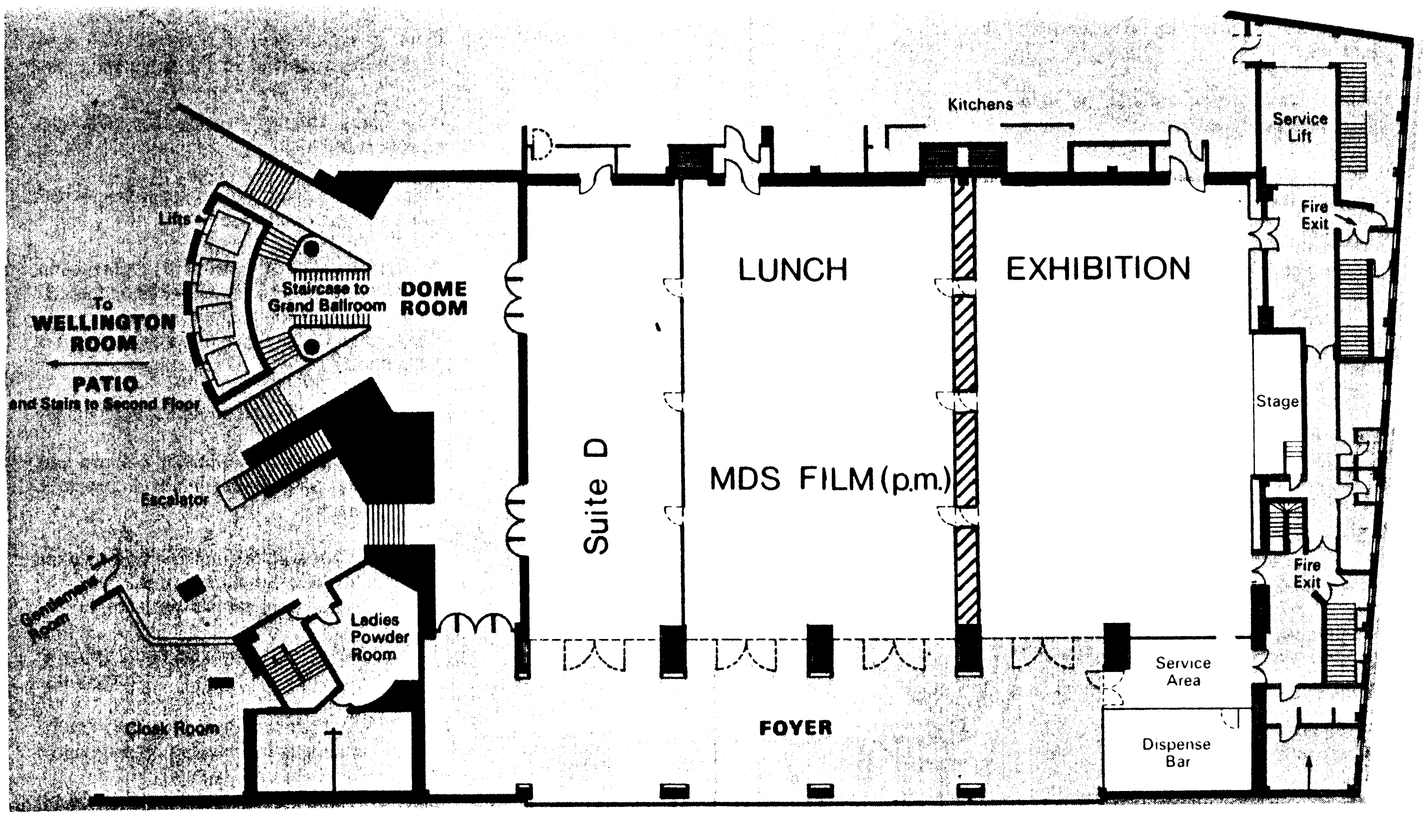
1)	9.30 - 10.15	3000 Series Overview, Phil Pittman, Rapid Recall Ltd.
2)	10.35 - 11.15	
3)	11.30 - 12.30	Programme Debugging Techniques, John Payne, Computer, Analysts and Programmers Ltd.
4)	14.00 - 14.45	Microprogramming the 3000 Series, Stan Mazor, Intel International
5)	15.00 - 15.45	Introduction to PL/M, Mike McCullough, Intel International
6)	16.00 - 16.45	

Suite D

1)	9.30 - 10.15	Introduction to the Commercial Aspects of Microcomputing, Keith Chapple, Intel Corporation (UK) Ltd.
2)	10.35 - 11.15	8080A Overview, Howard Kornstein, Intel Corporation (UK) Ltd.
3)	11.30 - 12.30	8080A Overview (continued)
4)	14.00 - 14.45	4040 Overview, Howard Kornstein, Intel Corporation (UK) Ltd.
5)	15.00 - 15.45	4040 Overview (continued)
6)	16.00 - 16.45	Discussion. Future Trends in Microprocessing, Keith Chapple (Intel), Bob Robinson (Rapid Recall), Edgar Valentine (GEC Semiconductors)

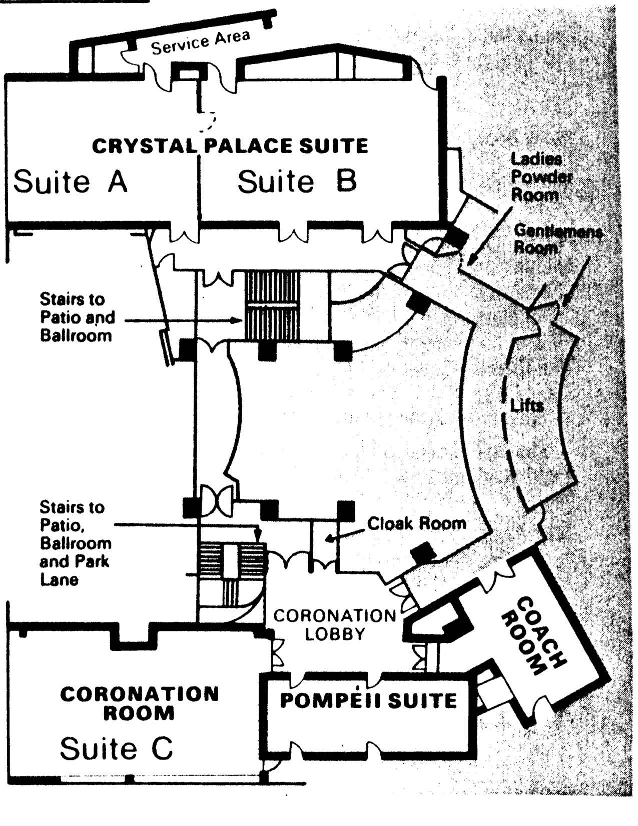
PLAN OF THE HILTON

Ground Floor



NOTE: If response to any particular Seminar exceeds the expected numbers it may be necessary for us to change the lettering of the Seminar suites. Notices to this effect will be clearly posted.

First Floor



intel Microcomputer Fair Seminar Suite A

PROGRAMME MANAGEMENT

by R.A. Perrin, Senior Consultant, Computer, Analysts and Programmers Ltd
CAP House, 14-15 Great James Street, London WC1 N3DY
Tel: 01-242 0021

This paper describes well-proven management techniques used in the development of software products. The techniques and project organization described are suitable for a small project of up to five people, this being typical of the current size of microprocessor projects.

Project Manager Responsibilities

The motivator for the project will have laid down the objectives in terms of its basic functions, characteristics, the operating environment and the planned volume of the production run. He will also know the maximum development costs and time which can be tolerated.

The responsibilities of the project manager can be stated as follows:

* Produce a plan for the development of the project.
* Monitor development against the agreed plan and report deviations.
* Ensure the product performs as required.

Development Plan

The purposes of the development plan are to measure the cost and duration of the project to determine if it is worthwhile continuing and to make sure that resources are available, that activities are started on time and that progress is accurately monitored.

The project manager in the production of the development plan must decide what activities have to be done, work out their interdependence and thus the sequence in which they will be carried out, and calculate the time and resources required.

The most obvious, and perhaps the most difficult of these, is the first: deciding what has to be done. Probably the worst error that can be made is to miss out an activity altogether. It is thus necessary to have a clear idea of the activities which are normally present in a microprocessor project.

Feasibility Study

The Feasibility Study lays the foundation for the remainder of the project and should include the following topics:

Project Objectives: These are basically the 'terms of reference' for the project and the project manager must ensure that he is given all the relevant requirements of the product in terms of its function, its interfaces, its required performance and reliability, and its operating environment.

Outline Design: The outline design will identify the functions to be performed by discrete logic and programmed logic. The hardware/software tradeoffs can be decided by examining the project objectives. Once this has been decided, initial estimates are required on the components needed (I/0 ports, memory chips, support chips, DMA, interrupts etc.). It is not essential at this stage to have selected the processor, but it is important to have identified the type (e.g. 4 bit, 8 bit, p-mos, n-mos).

Preliminary Project Plan: The project plan will in general contain the following information:

* Major activities; estimates of cost and time
* Critical activities
* Problems, unknowns, and uncertainties

At this stage, a rough estimate may be made on the length of an activity, based perhaps on previous experience or on the knowledge that a particular technical difficulty exists.

The completion of the Feasibility Study is an important checkpoint in the project and two parallel activities can be initiated after it has been agreed: programme development and hardware development. This paper will specifically pursue the programme development, although parallels with hardware development can be readily seen.

Programme Development

This consists of the following major activities:

Programme Analysis: This activity produces the Functional Specification which is a user-orientated document which specifies in detail the action to be taken by the programme for every event whether it is initiated by an external signal or by elapsed time.

Programme Design: This activity produces the Programme Specification which describes how the programme is to be constructed. This is a crucial stage in the development of a programme. A badly designed programme will result in extra time and costs during programme testing and may result in a programme going 'live' with undetected bugs. The programme is divided into logical and functional units (modules) which, in terms of logical complexity and volume of code, are capable of being understood quickly and easily. Another rule for a module is that it must have only one entry and exit point, and should normally return control to the module which invoked it.

Flowcharting: Flowcharting can be a useful step prior to coding. It is a useful discipline in that it can assist in ordering the logic within a module so that the logic can proceed in a step-by-step fashion. The correct ordering of the logic within a module is important in order to avoid the 'tangled ball of wool' construction with its consequential problems of testing, documentation and maintenance.

Much of a module's complexity arises from a module containing many jumps to other parts of the module - these jumps being both forward and backward. These sudden transfers of control tend to make it difficult to follow the logic of the code, and difficult to know at any given point in the code what the present conditions are. The complexity of the code and its jumps may also increase as a programme is maintained and modified.

The flowchart should be used to build a module structure so that control flows from top to bottom or from beginning to end. There is no back-tracking except for the 'repetition' or 'DO WHILE' structure.

Coding: By this stage, coding should be a mechanical translation from the flowcharts. The project manager should have ensured that programme standards such as linkage between modules and passing of parameters have already been defined.

There are certain golden rules which the project manager must ensure are obeyed:

* An instruction must not alter another instruction at run-time.

* 'Tricky' code resulting from a programmer's obsession to save a few bits of memory or a few microseconds of processor time must be actively discouraged.

* Comments should be extensive and meaningful in terms of the application function rather than a description of the machine instruction.

Preparing a Unit-Test Plan: Unit testing is defined as the testing of a collection of modules in isolation. The objective of the test plan is to identify the test data required to exercise all logical paths of the test unit, and to predict the test results. The test data should test not only normal conditions but also error conditions, such as invalid data and table overflow.

Unit-Testing: During unit testing, the project manager should ensure that proper records of test runs are being kept and that the test runs are cross-referenced against the test plan.

Integration Testing: Integration testing is the testing of the complete programme as a functional unit. Even if the unit of the previous stage was the complete programme, integration testing is a progression from unit testing since we are now concerned with testing the ability of the system to handle combinations of test data and conditions arriving in realistic circumstances.

System Testing

System testing is the testing of both discrete and programme logic as an entity or system. It is important that some programme test aids should be available during these later test stages in the anticipation that things will go wrong.

Documentation

During the development of the project, documentation has been building up as the natural completion of a stage, e.g. study report, functional specification, programme specification, programme listing. The project manager should ensure that user manuals are scheduled as early as possible.

Estimating

Estimating and scheduling a project are important responsibilities for the project manager. There are three methods of estimating with which the manager should be familiar: experience, quantitative, and constraint.

Experience Method: This approach takes advantage of experience on a similar job and simply assumes that like tasks take like resources.

Quantitative Method: This method depends upon ability to estimate programme size. The programme is broken down into its constituent modules and at least one module which is considered to be of average size is studied in detail.

$$\text{Deliverable instructions} = \text{number of modules} \times \text{average module size}$$

$$\text{Implementation effort} = \frac{\text{deliverable instructions}}{\text{programmer productivity}}$$

Implementation is defined as the flowcharting through to unit testing stages of the project. The programmer productivity depends on the manager's judgement as to the difficulty of the programme. Productivity rates of 10, 20 or 30 instructions per day are selected according to whether the programme is 'easy', 'medium' or 'difficult'.

$$\text{Finally, project effort} = \frac{\text{implementation effort}}{0.6}$$

The magic figure of 0.6 is obtained from statistics which show that 10% of project effort is absorbed by the design stages, 60% by the implementation stage, and 30% by the integration and system test stages.

Constraint Method: Based on constraints, time or costs, the manager agrees to do the job within the constraints.

Scheduling

Scheduling is necessary at two levels: activities within the project, and activities to people. The activity chart (or PERT chart) is used to show which activities can be conducted in parallel, the dependencies between activities, and the critical path of the project. It is important to identify the critical path since the manager is then in a position to know which activities can slip without jeopardising the overall timescale of the project.

Bar charts are useful to show who is assigned to each task, their progress, and when significant events (checkpoints or mistakes) occur.

Important statistics to remember in the scheduling of a project are that on average 30% of the elapsed time of a project is consumed by the design stages, 40% by the implementation stages, and 30% by the integration and system test stages. Whilst these are only average percentages, a project manager would be wise to consider his reasons if his percentages differ significantly.

Monitoring

The monitoring functions of a project manager are continuous in the sense that he should know at all times the state of the project and its difficulties. It is useful, however, to have regular meetings at discrete points, normally at weekly and monthly intervals.

In Conclusion

It has not been possible within this paper to discuss all the functions of a project manager. Indeed some of the subjects mentioned are worthy of papers in their own right. However, it is worth emphasising that the techniques described in this paper are all well-proven and have been built up over years of successful and, unfortunately, less successful project implementations. Since these experiences are well documented, the onus is upon the engineer entering the software field to learn from these past mistakes and successes. To err once is human, to repeat the mistakes of yesteryear is difficult to forgive.

Bipolar PROM Numerical Cross Reference

Part Number	Prefix and Manufacturer	Organization	Intel Part Number	
			Direct Replacement	For New Designs[1]
1024-2	HPROM-Harris	256 x 4	M3621	
1024-5	HPROM-Harris	256 x 4	3621	
1024A-2	HPROM-Harris	256 x 4	M3601	
1024A-5	HPROM-Harris	256 x 4	3601	
27S10C	AMD	256 x 4	3601	
27S10M	AMD	256 x 4	M3601	
27S11C	AMD	256 x 4	3621	
27S11M	AMD	256 x 4	M3621	
5300	MMI	256 x 4	M3601	
5300-1	MMI	256 x 4	M3601	
5301	MMI	256 x 4	M3621	
5301-1	MMI	256 x 4	M3621	
5305	MMI	512 x 4	M3602	
5305-1	MMI	512 x 4	M3602	
5306	MMI	512 x 4	M3622	
5306-1	MMI	512 x 4	M3622	
5340	MMI	512 x 8	M3604	
5341-1	MMI	512 x 8	M3624	
54S287	SN-TI	256 x 4	M3621	
54S287	DM-National	256 x 4	M3621	
54S387	SN-TI	256 x 4	M3601	
54S387	DM-National	256 x 4	M3601	
5603AC	IM-Intersil	256 x 4	3601	
5603AM	IM-Intersil	256 x 4	M3601	
5604C	IM-Intersil	512 x 4	3602	
5604M	IM-Intersil	512 x 4	M3602	
5605C	IM-Intersil	512 x 8	3604	
5605M	IM-Intersil	512 x 8	M3604	
5623C	IM-Intersil	256 x 4	3621	
5623M	IM-Intersil	256 x 4	M3621	
5624C	IM-Intersil	512 x 4	3622	
5624M	IM-Intersil	512 x 4	M3622	
5625C	IM-Intersil	512 x 8	3624	
5625M	IM-Intersil	512 x 8	M3624	
6300	MMI	256 x 4	3601	
6300-1	MMI	256 x 4	3601-1	
6301	MMI	256 x 4	3621	
6301-1	MMI	256 x 4	3621	
6305	MMI	512 x 4	3602	
6305-1	MMI	512 x 4	3602	
6306	MMI	512 x 4	3622	
6306-1	MMI	512 x 4	3622	
6340	MMI	512 x 8	3604	
6341-1	MMI	512 x 8	3624	

Part Number	Prefix and Manufacturer	Organization	Intel Part Number	
			Direct Replacement	For New Designs[1]
74S287	SN-TI	256 x 4	3621-1	
74S287	DM-National	256 x 4	3621-1	
74S387	SN-TI	256 x 4	3601-1	
74S387	DM-National	256 x 4	3601-1	
7573	DM-National	256 x 4	M3601	
7574	DM-National	256 x 4	M3621	
7610-2	HM-Harris	256 x 4		M3601
7610-5	HM-Harris	256 x 4	3601-1	
7611-2	HM-Harris	256 x 4		M3621
7611-5	HM-Harris	256 x 4	3621-1	
7620-2	HM-Harris	512 x 4		M3602
7620-5	HM-Harris	512 x 4	3602	
7621-2	HM-Harris	512 x 4		M3622
7621-5	HM-Harris	512 x 4	3622	
7643-2	HM-Harris	512 x 8		M3604
7643-5	HM-Harris	512 x 8	3604	
7644-2	HM-Harris	512 x 8		M3624
7644-5	HM-Harris	512 x 8	3624	
82S115	N-Signetics	512 x 8		3624
82S115	S-Signetics	512 x 8	M3624	
82S126	N-Signetics	256 x 4	3601-1	
82S126	S-Signetics	256 x 4		M3601
82S129	N-Signetics	256 x 4	3621-1	
82S129	S-Signetics	256 x 4		M3621
82S130	N-Signetics	512 x 4		3602
82S130	S-Signetics	512 x 4		M3602
82S131	N-Signetcis	512 x 4		3622
82S131	S-Signetics	512 x 4		M3622
8573	DM-National	256 x 4	3601	
8574	DM-National	256 x 4	3621	
93416C	Fairchild	256 x 4	3601	
93416M	Fairchild	256 x 4		M3601
93426C	Fairchild	256 x 4	3621	
93426M	Fairchild	256 x 4		M3621
93436C	Fairchild	512 x 4		3602
93436M	Fairchild	512 x 4		M3602
93438C	Fairchild	512 x 8		3604
93438M	Fairchild	512 x 8		M3604
93446C	Fairchild	512 x 4		3622
93446M	Fairchild	512 x 4		M3622
93448C	Fairchild	512 x 8		3624
93448M	Fairchild	512 x 8		M3624

Note 1. The Intel® PROMs have the same pin configuration and differ only in access time from the PROMs in the first column. The exception is the 82S115 which has a different pin configuration than the 3624.

INTERFACE TO THE 4040 MADE EASY

by Tom Cubitt, GEC Semiconductors Ltd

Two very powerful interface chips have recently been added to Intel's 4-bit microcomputer family. These are the 4269, a display and keyboard interface unit, and the 4255, a parallel I/O device.

Interfacing to Keyboards and Displays

The 4269 enables the 4040 cpu to be easily interfaced with a wide variety of input devices such as encoded and non-encoded keyboards, switches and push-buttons, and will also drive alpha-numeric, numeric, or on/off displays. These displays can be of seven-segment LEDs or Burroughs Self-scan* or Panaplex* displays. If required, a single 4269 will drive, and individually control, an array of up to 128 LED indicators.

There are several advantages that accrue from the use of the 4269. For example, reliability is increased and overall system cost is reduced because of the lower component count. Many of the routine operations involved in handling keyboards and displays are transferred from the cpu to the interface, leaving more processing time available for other system tasks, so that throughput is increased.

During system initialization, the 4040 cpu outputs a control word to 4269 which selects one of three possible input modes (sensor scan, keyboard scan and encoded keyboard input) and one of two output modes (individually scanned display device or self scan display). The 4269 is connected to the CM-RAM line and, therefore, has a fixed RAM address which allows up to four 4269s to be used with each CM-RAM in the system without any additional logic.

A keyboard with up to 64 keys in an 8 x 8 matrix can be connected to the 4269. A shift input enables the number of keys handled to be increased to 128. Alternatively, using a control and shift input, 256 switches can be connected to the device. An eight character first-in-first-out (FIFO) RAM on the chip functions as a keyboard buffer.

When used with a non-encoded keyboard (keyboard scan mode), pressing a key will result in a code being placed in the buffer which describes the position in

* Self-scan and Panaplex are trademarks of the Burroughs Corporation

the matrix of the key which has been pressed. The user's software assigns a value to the key and may interpret it as a simple numeric value or as an instruction to start execution of a sophisticated programme sequence. Automatic switch de-bounce and two-key rollover are provided in this mode.

When used with an encoded keyboard the output code from the keyboard is loaded directly into the FIFO buffer. Keyboards of this nature are likely to incorporate their own rollover and bounce protection.

In the sensor mode, the 4269 will monitor the condition of up to 256 switches. The de-bounce and rollover logic is inhibited so that simultaneous contact closures can be monitored.

A status buffer indicates the number of characters in the FIFO and if character over-entry has occurred. Another output indicates when a character has been entered into the FIFO and is, therefore, very suitable for generating a system interrupt.

On the output side, two 16 x 4 display registers continuously circulate information to be displayed in synchronism with the keyboard scan lines. These registers are directly accessible by the cpu and can be treated as normal read/write memory. All of the 4040 memory reference instructions can be applied to the display registers. It is even possible to implement a system in which the 4269 display registers provide all the RAM storage needed by a system, eliminating the need for any RAM chips at all!

Under programme control, either 4 x 32 or dual 4 x 16 individual displays can be accommodated. If desired, Self-scan displays of 16, 18 or 20 characters can be driven or an array of up to 128 individually controlled LEDs can be used.

The 4269 resides on the 4040 system's timing bus and derives its basic timing from the $\emptyset_1$ and $\emptyset_2$ clock signals. Synchronization and chip select information are provided by the sync and CM-RAM lines respectively.

Programmable Parallel Interface

To cater for parallel I/0 requirements, the 4265 general purpose programmable 16-line input/output (I/0) i.c. was introduced; this, under software control, provides 14 separate modes of operation.

The 4265 enables the 4040 cpu to be interfaced to a wide variety of peripheral units with little or no extra logic. These peripherals may be those normally associated with computers, such as keyboards, tape punches and readers, printers, etc., or the external devices often found in microprocessing systems, including such devices as D/A or A/D converters, indicators, relays, control valves, actuators, transducers and the like.

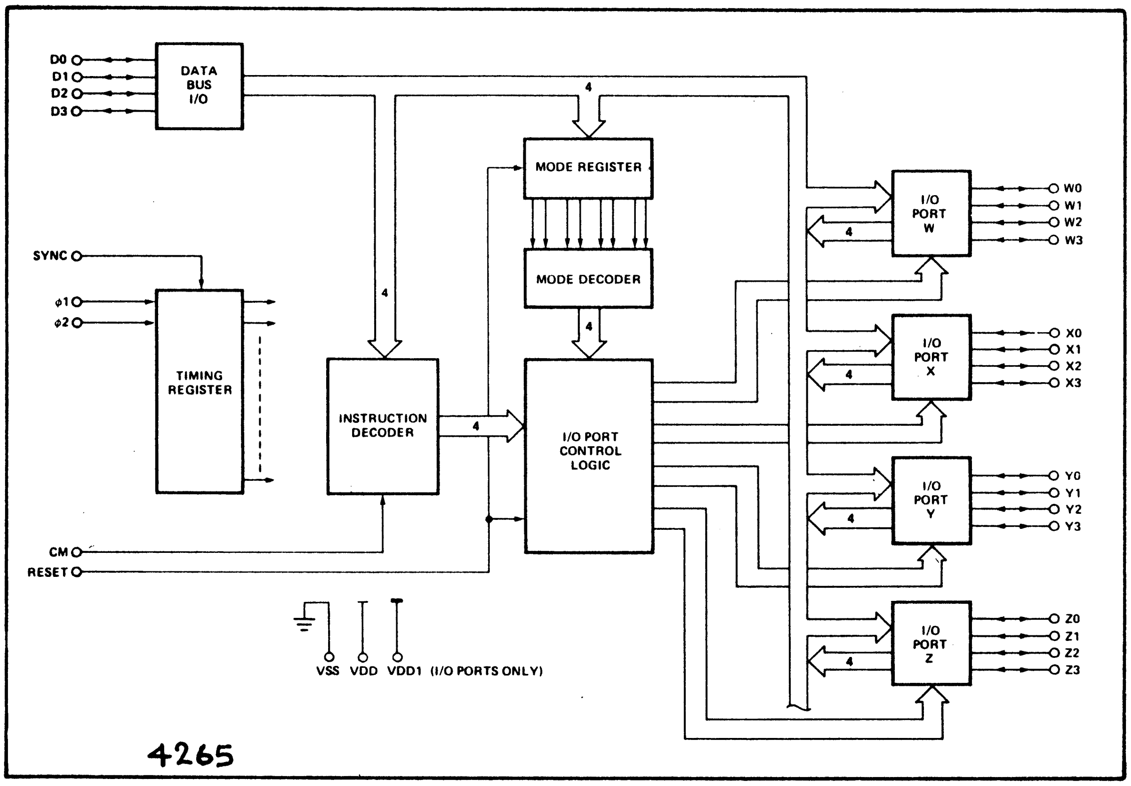
This versatility is due to the programmability of the 4265. The function carried out by the device is determined by control information fed to it by the cpu as dictated by the programme during the initialization or 'power up' sequence of the microprocessor.

Basically, the I/0 lines of the 4265 are arranged in four groups of four bits. The characteristics of each group of four lines, or ports, are determined by instructions from the cpu. There are 14 such instructions which can be divided into three groups. One group allows any of the four ports to be individually set up as input or output ports. This is particularly useful in a process control environment where the 4040 is being used to control machinery, on/off devices or other logic. The second group of instructions is concerned with multiple-bit data transfers. These can be either synchronous or asynchronous and can be eight bits at a time with full hand-shaking (data requested/data acknowledged) capability. The third group allows interface with standard off-the-shelf memory components.

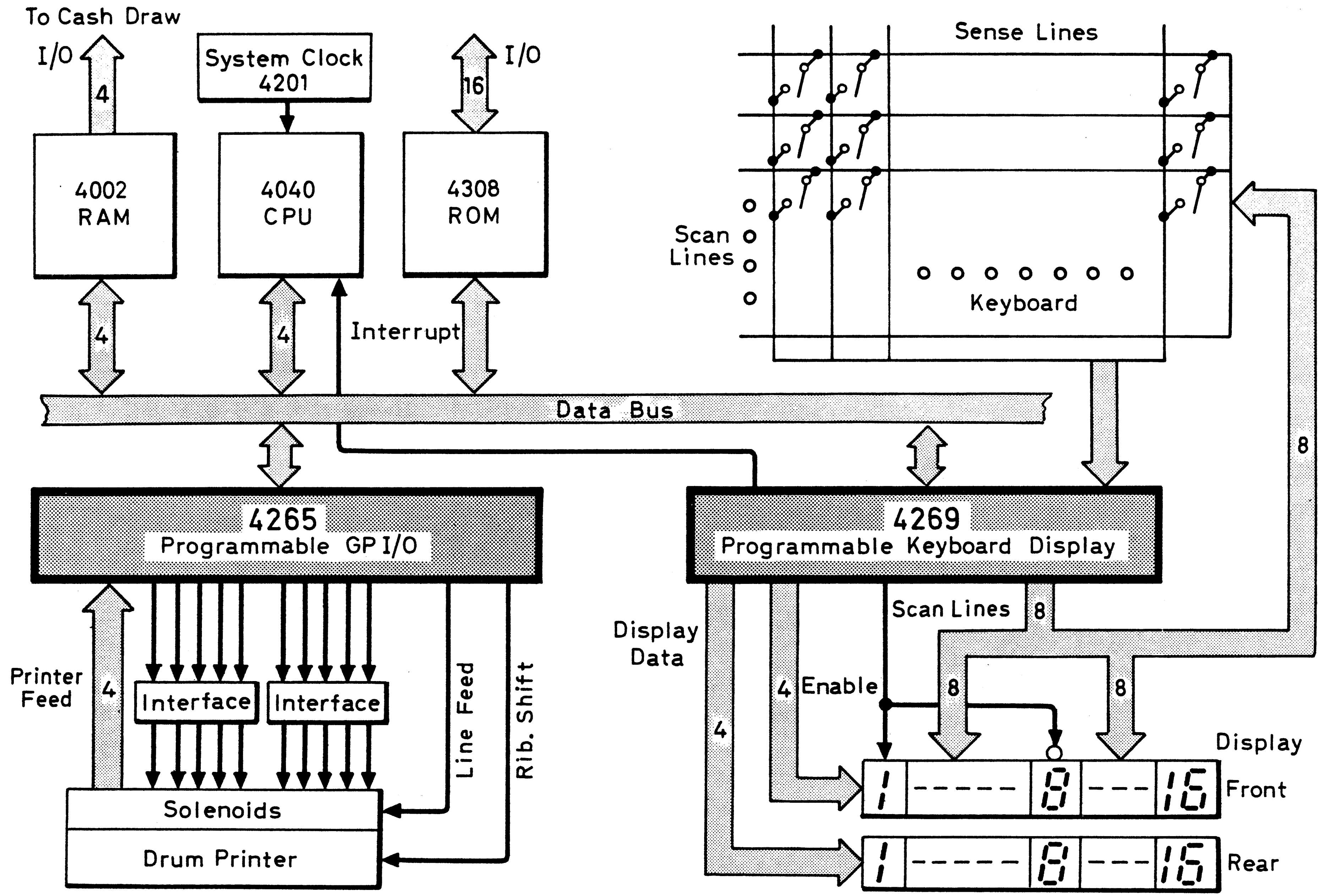
The 4265 is compatible with the other members of the 4040 family as well as TTL and provides the designer with buffered strobed inputs and outputs. The 8-bit data transfer capability means that peripheral devices designed for use with the 4040's big brother - the 8080 - can now be interfaced with the 4040 with the minimum of trouble.

Each 4040 cpu can be connected to up to eight 4265s using the data and control busses to provide 128 I/O lines.

The Intel 4265 is housed in a 28-pin plastic dual-in-line package. Operating speed is compatible with other 4040 system components.



PRINTER DISPLAY TERMINAL USING INTEL PROGRAMMABLE I/O DEVICES



OEM COMPUTERS AND A DIY KIT

by Bob Robinson, Rapid Recall Ltd

Two recent introductions by Intel are a complete OEM computer system and a comprehensive kit of parts for engineers who wish to obtain hands-on experience by building their own microcomputer. While both of these systems are based on the 8080A microcomputer and are very similar in architecture, they were designed with entirely different objects in mind.

An 8-bit OEM computer

The OEM computer is on a single card measuring 305 x 171 mm (12 x 6.75 inches) and is complete with serial and programmable I/0 ports, 1K bytes of RAM, sockets for 4K bytes of ROM or PROM and a single-level interrupt system. The memory size and I/0 arrangements can be expanded as required up to the maximum allowable in an 8080A system using standard components and sub-assemblies.

The card, which has been designated the SBC-80/10, is the only complete single-card microcomputer system that is available today, and has been designed to fulfil all the general purpose computing requirements of a diverse range of OEM applications at low cost. For this reason the interface control chips are configured by software, and sockets are included on the board for user designated line termination and driver circuits to provide the signal level translations that may be necessary. In addition, three serial interfaces - RS 232, current loop TTY and TTL - are included on the board as standard.

A single 8251 is used to implement the serial data port, which can be linked to the three serial interfaces. The 8251 is programmable and will provide serial data transmission and reception to virtually any known standard including IBM bi-sync. The 8251 provides the following options: asynchronous or synchronous operation; odd, even or no parity; one, one and a half or two stop bits; and word lengths of 5 to 8 bits. Parity, overrun and framing error detection are incorporated. Baud rate can be selected to one of 15 rates from 75 to 19,200 in the asynchronous mode or to any of 5 rates from 3,490 to 56,000 in the synchronous mode.

Parallel and individual bit outputs are provided by two 8255 programmable peripheral interface chips which together give 48 lines which can be configured as inputs or outputs in a wide variety of different ways under software control, as shown in the table.

A whole range of OEM support cards have been announced for the SBC-80/10. These will enable memory expansion up to 64K bytes and I/0 expansion up to 504 input lines and 504 output lines. Also available are card cages, back planes, connectors, and a variety of cables. In addition, a proto-typing kit is announced which will enable a complete special-to-purpose OEM computer to be de-signed and built.

Possible parallel I/0 configurations

Port	Unidirectional Input or Output	Bidirectional	Control
1	8U. 8LS	8	–
2	8U. 8LS	–	–
3	8U. –	–	8*
4	8U. –	–	–
5	8U. –	–	–
6	8U. –	–	–

* Port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input/output or port 1 is used in its bidirectional mode.

U = Unlatched LS = Latched and Strobed

The SBC-80/10 is a proven design which is intended for large-scale production. If required, a licence can be supplied that will allow the SBC-80/10 to be manufactured by users.

All of our support aids, such as the MDS and ICE-80, are fully compa-tible with the SBC-80/10 and will greatly speed prototype development. Programmes can be written in either assembly language, using a resident assembler, or one of the many cross assemblers that are now available, or they can be written in PL/M and compiled using a time sharing service or an in-house computer.

DIY Development Kit

For newcomers to the world of microcomputing and for small-scale production, there is a comprehensive set of components known as the SDK-80 kit, which enables first-hand experience of using both microcomputer hardware and software to be gained at low cost. The kit, a proven design by Intel, includes a monitor and in-corporates all the lsi chips, crystal, i.c. sockets, printed circuit board, board connectors and other components necessary to construct an advanced 8080A 8-bit microcomputer system.

Supplied with the kit is a set of detailed step-by-step building instruc-tions, in addition to the latest 8080A System, Software and PL/M manuals.

The programmable nature of the interface devices and the way in which the board is designed allows the microcomputer to be configured in a wide variety of different ways and a generous amount of space is left on the board for the user's own circuitry.

Communication with the microcomputer is carried out with a standard teletype or visual display unit which is interfaced to the microcomputer via a serial I/0 port formed by an 8251 universal synchronous/asynchronous receiver/ transmitter (USART). Links on the board allow this communications port to be RS 232, TTY current loop or TTL compatible. The transmission rate can be set with wire links or switch selected to any of seven values from 75 to 4,800 bauds.

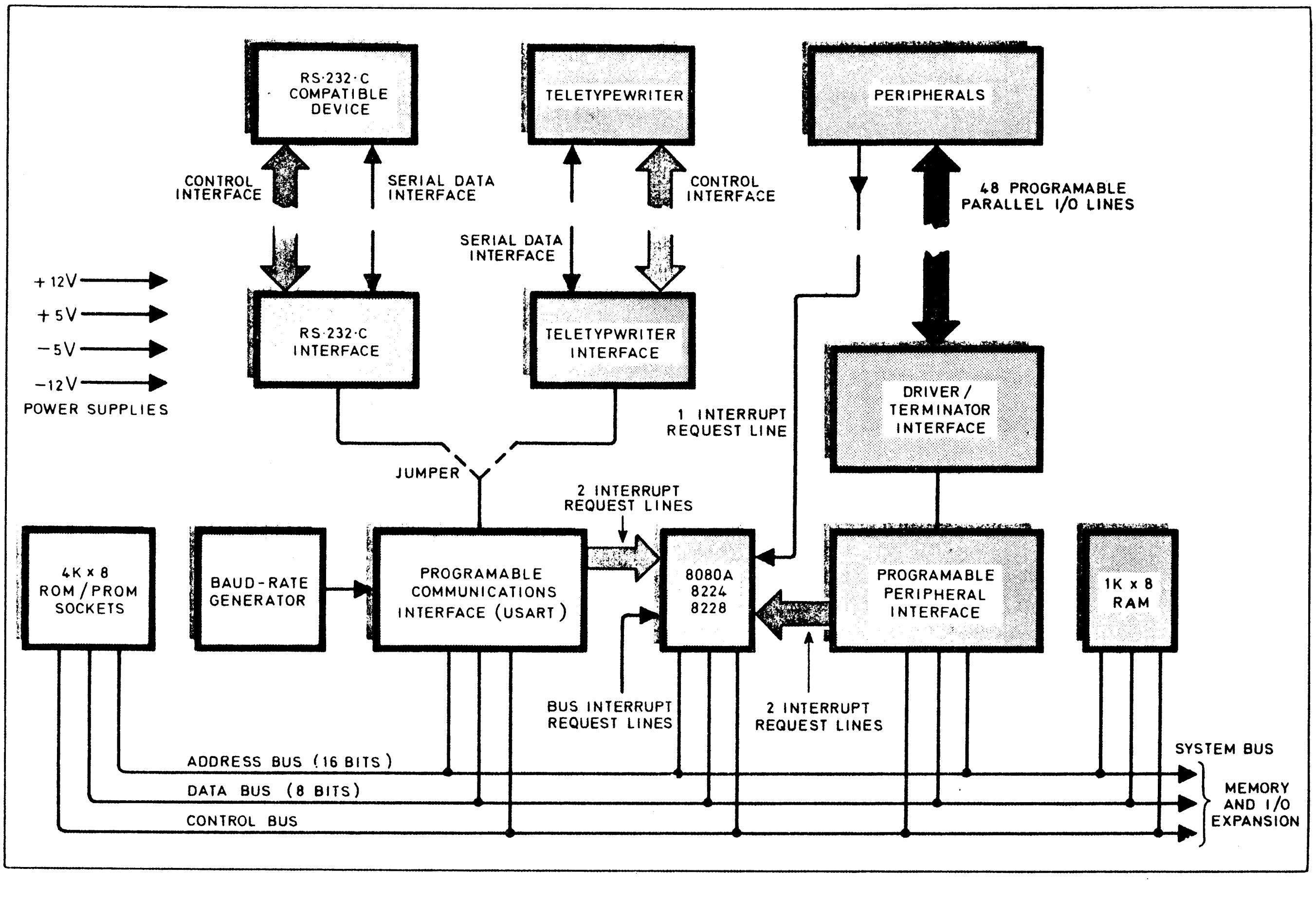
The kit is supplied with 2K bytes of PROM and the board will accommodate two further 8708 1K PROMs to bring the total up to 4K bytes - sufficient to hold some extremely sophisticated programmes. The PROM supplied with the kit for the lower 1K bytes of programme memory is already loaded with the System Monitor, which enables the user to immediately communicate with the system using the teletype or vdu, and to perform such actions as entering and running a programme, programme checkout, altering the contents of read/write memory locations, examining cpu register contents and so on. The second 1K PROM supplied is uncommitted and can be used to hold the user's own programmes.

Read/write memory work space and a stack area are formed by two 8111 256 x 4 RAMs, which together provide 256 bytes. The board supplied with the kit will accommodate a total of eight 8111 devices giving a total of 1K bytes of RAM.

A single 8255 programmable peripheral interface chip supplied with the kit provides 24 uncommitted input/output lines, which can be configured as required by the user. Under software control any of these lines can be inputs or outputs and can form 8-bit wide synchronous or asynchronous ports, bi-directional busses, individually controlled inputs and outputs, etc. Space is available on the board for fitting a second 8255 to double the number of I/0 lines.

The monitor communicates with the user via the teletype using the monitor's command language which consists of a series of single characters. Most of these commands consist of a character followed by one or more hexadecimal numbers.

Block diagram of the SBC-80/10 OEM computer.

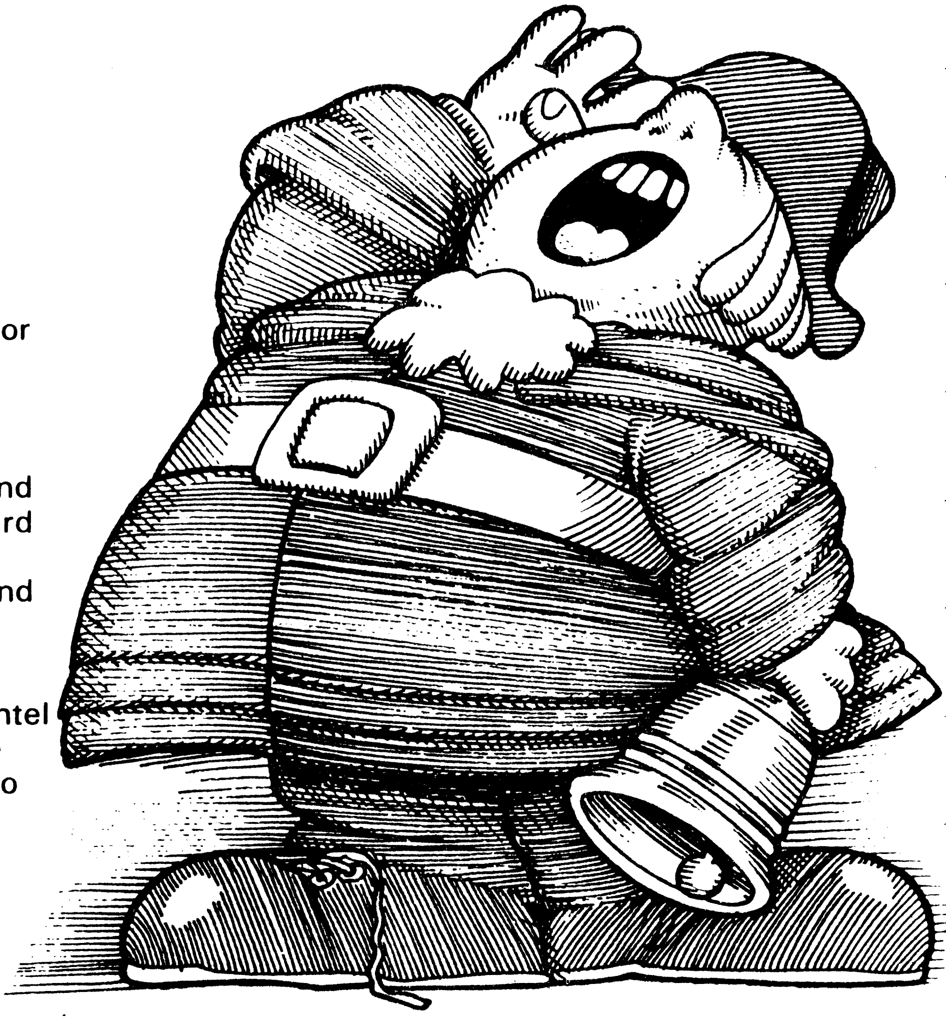


COME TO THE MICROPROCESSOR COURSES BY INTEL

With the continuing invention and discovery of new techniques and applications in microprocessing, the necessity for intensifying Intel's already highly successful training programme for the design engineer is evident.

The new season of Intel courses will be held at our well equipped, purpose designed training workshop at Oxford and run by our applications manager, Howard Kornstein who will be explaining both practically and theoretically his first-hand knowledge and experience.

Apart from the weekly Microprocessor Courses which are already planned by Intel and its distributors, Howard will also be pleased to arrange courses on request to meet customers specific requirements.



8080A PROGRAMMABLE SUPPORT DEVICES

by Tom Cubitt, GEC Semiconductors Ltd

Intel have introduced a number of programmable integrated circuits, which are all designed to relieve the cpu in a microcomputer system from the routine tasks involved in 'talking to' a peripheral device. In doing so, the peripheral chips leave more time available for more important system tasks and thereby increase the cost effectiveness of a system by enabling it to achieve a higher throughput. Additionally, the peripheral and other programmable support chips reduce cost and increase reliability by drastically reducing package count.

This paper will briefly describe the latest five devices that have been recently introduced by Intel and which fall into this category. All these devices share certain common features in that they are 'told' what role they are to perform by the cpu during the system's initialization or power-up sequence, and they can all be made to respond to the normal cpu I/O instructions, or can be connected to appear as memory locations to the cpu. In this latter mode, all the instructions that normally reference memory can be applied to the interface. In some applications this is an advantage.

8255 - Programmable Parallel Interface

The 8255 has three major modes of operation which, between them, cover most microprocessor interface requirements. Under programme control, the twenty-four I/O lines can be divided into two eight-bit ports and two four-bit ports and any of these ports can function as either an input or an output. For unidirectional-bus handshaking interfaces, the twenty-four lines are divided into two groups of twelve. Each group of twelve lines provides eight data lines, which can either be inputs or outputs, and four control lines for handshaking and interrupt. In the final mode of operation, eight of the lines become a bi-directional data bus and five lines are available for handshaking.

As mentioned earlier, the way in which the device functions is determined by a control word which is sent to the interface during the initialization of the system. This control word is stored in registers internal to the interface chip. It is possible to programme half of the device to function in one of the previously mentioned modes and the other half in another. For example, one half could provide an 8-bit synchronous input port with full handshaking and interrupt capability; four more lines could be used as inputs to monitor peripheral status and the remaining eight lines might be employed as outputs to control external equipment such as relays, motors, etc. Obviously, a very large number of other configurations are possible.

8251 - Programmable Serial Interface

Serial microcomputer interfaces are very similar in function to the normal serial data receiver/transmitter circuits one finds in any data communications system. The difference being, of course, that the precise function to be carried out by the interface is determined by a control word output by the cpu.

The 8251 provides synchronous or asynchronous reception and transmission of data words between 5 and 8 bits long. It provides facilities for one, one and a half or two stop bits; error detection for parity, overrun and framing, and will function in accordance with all known data transmission standards including IBM bi-sync. Speed of operation can be set anywhere from d.c. to 9.6k Bauds.

8253 - Programmable Interval Timer

Microcomputers are frequently used to control mechanical devices such as printers, relays and motors which have a response time that is orders of magnitude slower than the microcomputer itself. Because of this, the microcomputer often has to wait for the mechanical unit to complete its current task before it can demand that the next task be carried out.

One way of creating the required delay in microcomputer operation is to cause a programme loop to be executed a given number of times. This loop does not normally perform any useful function other than keeping the microcomputer occupied while the mechanical peripheral completes its task. This technique is extremely wasteful because while the microcomputer is executing the programme loop it cannot be doing anything else.

To overcome this problem, Intel have just introduced the 8253, a programmable interval timer which is designed for use in either 8008 or 8080 microcomputer systems for peripheral timing purposes. The new device allows the microcomputer to carry out other work instead of waiting for a slow peripheral.

The 8253 is housed in a single 24-pin dual-in-line package and is designed to be connected to the system's address, control and data busses. It is of n-mos construction and requires only a single 5V supply.

In essence, the device comprises three independent 16-bit counters which are driven as I/0 peripheral ports. Instead of setting up a software loop to provide a time delay, the programmer causes the cpu to output control information to the 8253 which causes it to start counting clock pulses and then to interrupt the microcomputer when the required delay has elapsed. Between time delay initialization and the interrupt the microcomputer can carry out other tasks.

Each of the counters can count in either binary or BCD from d.c. to 3MHz to provide an extremely wide range of different delays ranging from microseconds to hours.

The 8253 can also be used as a programmable baud rate generator, an event counter, a binary rate multiplier and a real time clock.

8259 - Programmable Interrupt Controller

The efficient management of interrupts has long been accepted as being the key to the effective use of microcomputers in those applications that require a prompt response to external events coupled with large processing throughputs. The rate at which external peripheral devices can be serviced determines how many other tasks can be assigned to the microcomputer. The more of these tasks that can be handled by the microcomputer the more cost-effective the system will be.

The 8259 is a priority interrupt controller which allows the entire system's interrupt structure to be configured by the software and to be changed at any time during the course of a programme. The new device will handle eight different priority levels and up to eight 8259s can be used in a system to provide up to 64 levels of interrupt without additional circuitry.

The 8259 is programmed in the same way as the programmable I/0 devices in the 8080A family. Control words sent to it by the cpu under the control of the programme determine how the 8259 will respond to interrupts and allow the programmer to select from a number of priority control algorithms which are provided by the 8259. Amongst the facilities offered is the capability of individually masking specific interrupt requests.

Interface with the microcomputer itself is standard and follows the same rules as for the other 8080A family members. A special input is provided to enable it to perform the role of either slave or master for use when more than one 8259 is being used in a system. Basically, the master accepts INT inputs from the slave or slaves and issues a composite request to the 8080A; when it receives the INTA signal from the system controller (8228) the first byte of the CALL is put on the bus. On subsequent $\overline{\text{INTA}}$ signals the interrupting slave puts out the address of the vector (the first instruction of the required interrupt routine).

8257 - Programmable DMA Controller

The transfer of data between a mass storage device such as a floppy disc or magnetic cassette and system RAM memory is often limited by the speed of the microcomputer. Removing the microcomputer during such a transfer and letting an auxiliary device manage the transfer in a more efficient manner greatly improves the speed and makes mass storage devices more attractive, even to the small system designer.

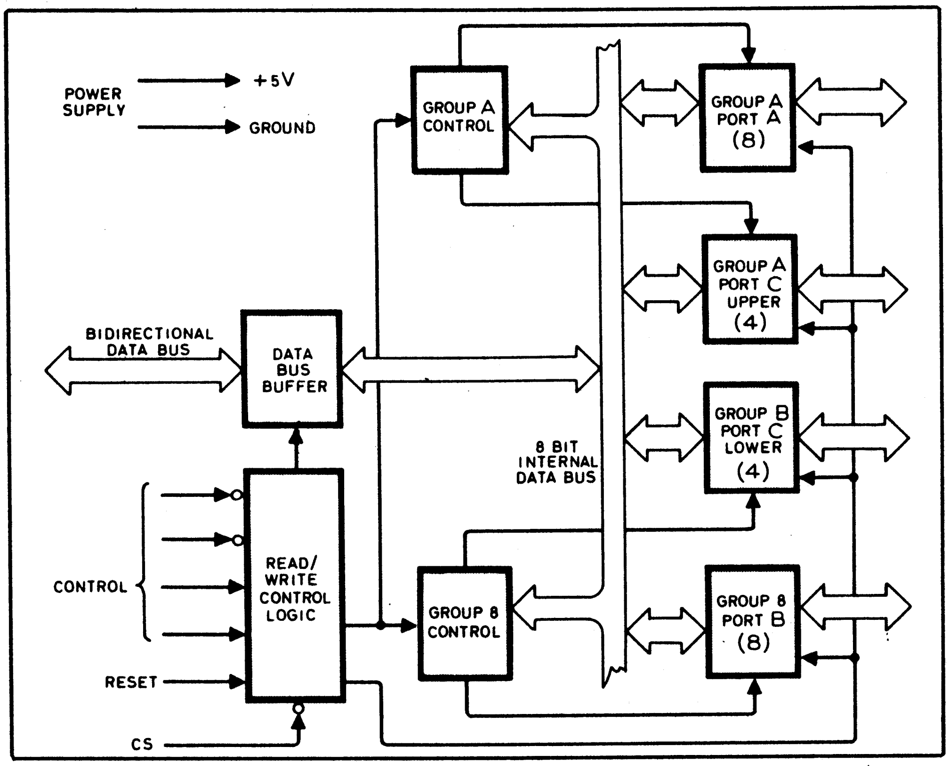
The 8257 programmable DMA controller is a single chip, four channel device for this purpose. Each channel is assigned a priority level so that if multi-DMA activities are required each mass storage device can be serviced, based on its importance in the system. In operation, a request is made from a peripheral device for access to the system bus. After its priority is accepted a HOLD command is issued to the cpu, the cpu acknowledges with a HLDA output which signals that the DMA channel has complete control of the system bus. Transfers can be made in blocks, suspending the processor's operation during the entire transfer, or the transfer can be made a few bytes at a time, hidden in the execution states of each instruction cycle (cycle stealing).

The modes and priority resolving are maintained by the system software as well as initializing each channel as to the starting address and length of transfer.

The system interface is similar to the other peripherals of the MCS-80, but an additional 8212 is necessary to control the entire address bus. A special control signal BUSEN is connected directly to the 8228 so that the data bus and control bus will be released at the proper time.

The 8257 generates, upon a peripheral request, a sequential memory address which will allow the peripheral to access or deposit data directly from or to memory. It also keeps count of the number of DMA cycles for each channel and notifies the peripheral when a programmable terminal count has been reached. Other features included are two-mode priority logic to resolve the request among the four channels, programmable channel inhibit logic, an early write pulse option, a module 256/128 Mark output for sectored data transfers, an automatic load mode, a terminal count status register, and control signal timing generation during DMA cycles. There are three types of DMA cycles: Read DMA Cycle, Write DMA Cycle and Verify DMA Cycle.

More than one 8257 can be used in a system to increase the number of DMA channels available.



DEVELOPMENT AIDS

by Tom Cubitt, GEC Semiconductors Ltd

There are three distinct types of microcomputer development system, each of which is intended to serve a different purpose.

First Stage

These systems are often referred to as evaluation or prototyping cards and normally comprise a microcomputer complete with RAM, EPROM, a teletype interface and a basic system monitor, which enables the user to carry out basic machine code programming and a limited amount of debugging. The recently announced SDK kit, which is the subject of a separate paper, is a first stage system. The main value of the prototype card is as a low-cost means of obtaining 'hands-on' practical experience with a particular microprocessor.

Second Stage

The Intel Intellec is a 2nd stage development system. This is housed in a metal cabinet complete with front panel switches and looks very much like a minicomputer. It enables programmes to be automatically assembled into machine code from symbolic statements; it enables programmes to be edited and provides some fairly sophisticated programme development and debugging aids .

Third Stage

There is only one third generation system at present; this is the Intel MDS and ICE-80 combination. MDS has been designed so that it is suitable for use with both 8080A and 3000 Series microcomputers. In fact, it will also be suitable for use with any future microcomputer we may introduce. In this paper we will discuss the MDS configuration used for the 8080A.

The MDS for the 8080A microcomputer contains an 8080A microcomputer with 16K (expandable to 64K) of RAM, a system monitor and I/O ports with interfaces for a wide variety of peripheral devices. The monitor is comprehensive and provides a large number of facilities for both hardware and software development and debugging.

ICE-80 comprises a hardware section which fits into the cabinet of the MDS and a software package which resides in the MDS RAM.

With the aid of the MDS, the systems designer will develop the prototype hardware and software for his system. In the user's prototype circuit card there will be a socket which would normally contain an 8080A cpu. The ICE module terminates in a 40-pin connector that electrically and functionally 'looks like' an 8080A cpu. This connector is plugged into the cpu socket on the prototype board so that there is direct communication between the prototype system and the MDS. It is just as if the facilities available in the MDS had been built into the prototype system. Sections of the microcomputer system within the MDS can be used to replace sections of the prototype circuitry so that the whole process of hardware/software integration and system diagnosis becomes a controlled step-by-step process.

The key feature of ICE is that the developed software is checked out in Real Time in the designer's own hardware. This provides a complete and totally real test of the operation of the microcomputer design. Testing is done with an easily understood and conversational command language. Software operations can be monitored by setting emulation breaks on specific operating conditions. Events leading up to the break can be examined.

If there is a hardware fault - for example, an unwanted voltage spike is appearing at an output port - then the spike can be used to automatically halt programme execution and cause the MDS to print out the 44 programme steps that were carried out immediately before the spike occurred, so that the cause can be determined.

The MDS has recently been enhanced by the availability of an Intel disc system. This can be supplied with either a single or a dual disc drive. It employs standard diskettes (about the size of a 45 rpm record) which have a storage capacity of a quarter of a million bytes each. Data can be transferred to and fro between the MDS and the disc at very high speed. For example, it takes only two or three seconds to load the assembler and editor into the MDS from the disc as against two to three minutes with a high-speed tape reader or tens of minutes with a teletype tape reader.

Assembly listings, intermediate files and object code can be stored directly on the disc. There is no need to wait while the terminal punches intermediate paper tapes. The time saving during a complete project is significant.

The disc system is controlled by an Intel 3000 bipolar microcomputer. A disc resident software package called ISIS (Intel Systems Implementation Supervisor) is supplied with the disc system and enables the user to manipulate his disc files. There can be up to 200 files on the disc which are allocated names by the user, and the file name is the only information that needs to be given to the system in order to store or retrieve the file. The actual placing and retrieval of the file or the disc is handled by the software without the user needing to know where the file or disc is located.

An attractive feature of the operating system is its 'random access' nature. Any data location in a file can be accessed immediately without carrying out a sequential search.

Earlier development systems required the user to load and read the paper tape once for each 'function' of the assembler. Very often two or three such 'passes' were required. With ISIS and the disc all passes take place automatically and in seconds. This feature alone can save days of development time.

The disc system is designed for use with an Intellec MDS which has 32K bytes of RAM.

To demonstrate how things are done with ISIS, imagine that you have a paper tape copy of the source code for an assembly language programme. Assume this programme was written for use on a memory-based MDS using Monitor I/0 calls, and you want to convert it to a diskette-based programme that uses the more powerful I/0 facilities of ISIS. You load the paper tape into your high-speed paper tape reader and initialize the Intellec MDS to run under ISIS. You tell ISIS to copy from the reader to diskette, giving the programme the name PROGA.SRC on the diskette:

```
-COPY  :HR:  TO PROGA.SRC
```

Then you call the ISIS Text Editor:

```
-EDIT PROGA.SRC
```

and make the text changes required to convert Monitor I/0 calls to ISIS I/0 calls. Now you can call the assembler to create hexadecimal object code and a listing file:

```
-ASM80 PROGA.SRC TO PROGA.HEX LIST PROGA.LST
```

Programmes must be in absolute binary format to execute with ISIS, so tell ISIS to convert the hexadecimal code to absolute binary:

```
-HEXBIN PROGA.HEX TO PROGA.BIN
```

Now you are ready to debug your programme by executing it under the Monitor:

```
-DEBUG PROGA.BIN
#3100H
.G
```

If errors occur, correct the source code using the editor and repeat ASM80, HEXBIN, and DEBUG.

When you are satisfied with the state of the programme, it can become part of your permanent library and executed by simply entering the file name:

```
-PROGA.BIN
```

If you prefer a shorter name, you can rename the file:

```
-RENAME PROGA.BIN TO PROGA
```

To check out a programme in real time

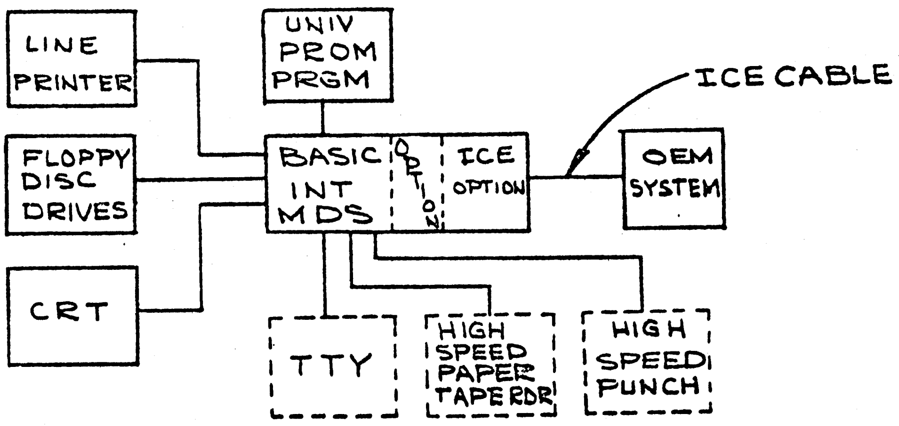
```
-ICE 80
```

loads the in-circuit emulator software. The command

```
.LOAD PROGA.HEX
```

loads the machine code for execution under ICE 80.

Figure 1: Intellec MDS System Block Diagram



Cross Product Software

ADVANTAGES AND USE

by Howard Kornstein, Intel Corporation (UK) Ltd

A large part of microcomputer system development can be carried out using a mainframe computer for programme development and debugging, and system simulation. If you have a suitable in-house computer all that is required is the Intel cross software relevant to your application. This software is also available from several time sharing bureaux in the U.K. All you need in this case is to rent a terminal and modem.

All the cross software is written in ANSI standard Fortran IV and requires a computer with a 32-bit word length (such as the IBM 360 or PDP-10).

Using the cross software it is possible to write assembly language programmes for the 4004, 4040, 8008 and 8080A microcomputers. The high-level language PL/M (a sub-set of the IBM language PL/1) is also available for the 8008 or 8080A microcomputers (it is interesting to note that a programme written in PL/M can be compiled for use on either the 8008 or the 8080A or any other microcomputer which we may introduce for the future for which we prepare a compiler). Also available is CROMIS, for preparing microprogrammes for the 3000 family. Simulators are available which enable the mainframe to 'look like' a microcomputer so that you can try out your programmes. The use of simulators is recommended for those programmes which are not greatly dependent on real time events. The cross product software which is currently available from us is as follows:

MCS-40 Cross Assembler

The MCS-40 Cross Assembler, MAC40, can be used for both the 4004 and the 4040. MAC40 provides capabilities which reduce the time and effort involved in programme development, debug and documentation. The cross assembler allows usage of the high speed I/O and text editing capability of a large computer system to further shorten the programming task.

MAC40 translates 4004/4040 machine assembly language instructions into the appropriate machine operation codes. In addition to eliminating the errors of hand translation, the ability to refer to programme addresses with symbolic names makes it easy to modify programmes by adding or deleting instructions, or to move the programme to another memory location. Full macro capability eliminates the need to rewrite similar sections of code repeatedly and simplifies programme documentation. Conditional assembly permits MAC40 to include or delete sections of code which may vary from system to system, such as the code required to handle optional external devices.

Output from MAC40 may be punched to paper tape in hex format for loading into an Intellec 4 Development System or may be punched in BNPF format to programme ROMs.

MCS-8 Cross Assembler

Known as MAC-8 this programme provides the same facilities as MAC40, but for the 8008. Output from MAC8 may be loaded directly to the 8008 Simulator (INTERP/8) for interactive, symbolic debugging or may be punched to paper tape in hex format for loading into an Intellec 8/Mod 8 Development System. It may also be punched in BNPF format to programme ROMs.

4004/4040 Simulator

The 4004/4040 Simulator, INTERP/40, is a complete simulation and debug programme for the Intel 4004 and 4040 microcomputers. Programmes can be run, displayed, stopped, and altered allowing step by step refinement without continuous reassembly of the source programme. INTERP/40 provides commands to control the execution of 4004 and 4040 programmes. Debug features are built in to help reduce the time and cost involved in programme checkout. The programme also provides symbolic reference to storage locations and operation codes as well as numeric reference in various number bases.

8008 Simulator

INTERP/8 is similar to INTERP/4 but is for use with the 8008.

MCS-80 Cross Assembler

Known as MAC80 this programme translates symbolic 8080 assembly language instructions into the appropriate machine operation codes. In addition to eliminating the errors of hand translation, the ability to refer to programme addresses with symbolic names makes it easy to modify programmes by adding or deleting instructions, or to move the programme to another memory location. Full macro capability eliminates the need to rewrite similar sections of code repeatedly and simplifies programme documentation. Conditional assembly permits MAC80 to include or delete sections of code which may vary from system to system, such as the code required to handle optional external devices.

Output from MAC80 may be loaded directly to the 8080 Simulator

(INTERP/80) for interactive, symbolic debugging or may be punched to paper tape in hex format for loading into an Intellec MDS Microcomputer Development System. It may also be punched in BNPF format to programme ROMs.

8080 Simulator

The 8080 Simulator, INTERP/80, is a complete simulation and debug programme for the Intel 8080 microcomputer. INTERP/80 provides commands to control the execution of 8080 programmes. Extensive debug features are built in to help reduce the time and cost involved in programme checkout.

INTERP/80 simulates execution of all 8080 machine instructions. Programmes either compiled on the PL/M compiler or assembled on the MAC80 Cross Assembler may be loaded directly into INTERP/80 for simulation and checkout.

INTERP/80 provides commands to:

: Set Breakpoints
: Trace Programme Execution
: Dump and Modify Memory
: Examine and Modify Registers
: Measure Programme Timing
: Examine and Set I/0 Ports
: Perform Interrupts and Stack Manipulations
: Perform Address Arithmetic

INTERP/80 also provides symbolic debugging capability. Memory locations may be referenced by their symbolic names, either through labels or variable names. This eliminates the need to know the specific absolute address of each variable or label.

PL/M High Level Programming Language

PL/M is a high-level programming language, specifically designed to ease the programming task for Intel's 8-bit microcomputers, the 8008 and the 8080. PL/M is a powerful tool, well suited to the requirements of the microcomputer system designed. The language has been designed to facilitate the use of modern techniques in structured programming. These techniques can lead to rapid system development and checkout, straightforward maintenance and modifications, and high product reliability.

The PL/M compilers convert a free-form symbolic PL/M programme into an equivalent 8008 or 8080 object programme. The compilers themselves take care of all the details of machine or assembly language programming, which permits the programmer to concentrate entirely on effective software design, and the logical requirements of his system.

Output from the PL/M compiler may be loaded directly into the 8008 or 8080 simulator programmes for interactive, symbolic debugging or may be punched to paper tape in hex format for loading into an Intellec Microcomputer Development System. It may also be punched in BNPF format to programme ROMs.

3000 Cross Microprogramming System

The Intel Series 3000 Cross Microprogramming System, CROMIS, is a software system that supports the generation of microprogrammes for Series 3000 processor and controller micro-architectures. It provides extensive programming facilities that greatly reduce the time and effort required to develop, debug, and document a microprogramme.

CROMIS consists of two major software subsystems, XMAS and XMAP. XMAS is a symbolic microassembler which is dynamically user extensible in the size and structure of the target microinstruction format. XMAP is a complementary subsystem which maps the microinstruction bit patterns produced by XMAS into the desired physical microprogramme memory locations.

In addition to providing four built-in microinstruction fields and corresponding mnemonic sets for the basic 3001 MCU and 3002 CPE functions, XMAS accepts user definitions for extended microinstruction fields and their associated mnemonics. Graphic debugging aids, string macro capability, definable defaults, and extended address generation further simplify the microprogramming of Series 3000 computing elements.

XMAP accepts the microinstruction file produced by XMAS and generates under user specifications one or more programming files for use with standard memory components. It enables the user to specify the mapping of the field into the physical bit positions of the microprogramme memory components.

CROMIS is designed for use on almost any modern computing system with high speed I/0 and on-line file facilities. It is available in ANSI (standard) FORTRAN IV source form for user installation or may be immediately accessed on any of several major timesharing services throughout the world. To insure the long term reliability and maintainability of CROMIS, all component programmes are written in a highly modular, structured programming style with extensive operational documentation.

User's Library

In addition to the Intel software just described, we also have an extensive library of programmes submitted by microcomputer users. The library currently contains a number of cross products for use on a variety of mainframes in addition to a useful selection of utilities.

INTRODUCTION TO MICROCOMPUTER PROGRAMMING

by Phil Pittman, Rapid Recall Ltd

Once the decision is made to use a microcomputer in a particular system, design and development usually follows a fairly standard pattern. From a study of the system requirements, the hardware designer can select from a family of microcomputer parts those components which match his needs in terms of processing power and the number and characteristics of inputs and outputs.

At the same time, the programmer will map out flow charts for the system and will estimate how many instructions will be required to cause the microcomputer to carry out the desired task. Since these instructions will be stored in semiconductor memory, it is now possible to calculate how many memory packages will be required.

The hardware designer sets about designing the printed circuit boards and the various input/output interfaces, or orders a ready-made OEM board, while the programmer writes his programmes. To check them he will use a microcomputer development system such as the Intellec 80 or the MDS. These systems are used to develop programmes and have several facilities which make the task easier.

At some stage the hardware and the programmes (software) are combined and the whole system is checked out. It is here that the MDS comes into its own, as it is designed to aid with both hardware and software development. Using an in-circuit emulator (ICE) the MDS system is substituted for the central processor in the user's system. Without going into detail, since this is the subject of another paper, fault diagnosis and system development become much more straightforward than they have been previously.

Programming

A programme comprises a set of instructions that are followed by the microcomputer to produce the relationship between inputs and outputs required by the application. There are many different instructions; basically, however, they fall into six groups: arithmetic, logical, input/output, data movement, programme control and data manipulation.

Each instruction is represented by a unique binary bit pattern, or series of binary bit patterns, within the microcomputer's memory. The whole object of writing a programme is to produce the binary instructions which will cause the microcomputer to carry out the desired function. There are three basic ways in which this can be done, which all start in the same way - the flow chart.

The flow chart performs a dual role. It graphically illustrates the way in which a task is to be accomplished step-by-step, and provides the programmer with an overview of the whole system. Flow charts have the property of almost infinite expansion.

The programmer divides the job to be carried out into a series of interconnected tasks. Each task is progressively divided into smaller tasks until each one can be represented by a single microcomputer instruction. At each stage of this process a flow chart will have been produced, showing how each task is to be implemented and how it fits into the overall scheme.

It may be found that a particular task has to be carried out several times during a programme. Common tasks such as these need only be written once; they can then be called up as and when required. With some microcomputers, such as the Intel 8080, these 'subroutine' calls can be made conditional.

When a microcomputer is called upon to execute a subroutine, it stores away enough information to enable it to return to where it left off when the subroutine is completed. Returns from subroutine can also be conditional. On a production line, for instance, an example of a conditional return could be: go to the subroutine 'put components in box'. Is the box full yet? If no, repeat subroutine load. If yes, place full box on conveyor and get new empty box.

When the flow charts are planned to the programmer's satisfaction, he begins the process of writing programmes. He usually starts with the minor subroutines and continues through the more important tasks, which link the subroutines together, until he reaches the overall controlling programme which links all the main tasks together.

Programmes are usually written using a suitable computer terminal, such as a teletype, coupled to a microcomputer development system such as the MDS mentioned earlier.

The object is to produce a series of binary instructions to be stored in the microcomputer memory. This series of binary coded instructions is called the object code: the sequence 10110111, 11110000, 10010101, 01111000, could well be a series of instructions. When it is considered that a programme may comprise thousands of such apparently meaningless collections of 1s and 0s, it is not surprising that easier ways of programming were developed.

The first step is to make the binary words easier to handle by using a form of shorthand called hexadecimal notation. Hexadecimal is, in fact, more than just shorthand - it is a number system in its own right with the base, or radix, 16. An eight bit binary word (or byte) can be represented by two hexadecimal characters as shown in Table 1.

It is possible to write microcomputer instructions in hexadecimal notation but the process is tedious, fairly cumbersome and, therefore, expensive.

Table 1

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

A much better approach is to use an assembler language. Each instruction in a microcomputer's repertoire can be represented by a simple mnemonic code which is constructed in such a way as to describe the operation performed by the instruction. For the Intel 8080, typical mnemonics are:

Mnemonic	Explanation
MOV	Move data
ADD	Add
SUB	Subtract
CMP	Compare

When a programme is written in mnemonics, a computer is used to convert the mnemonic code, or user's 'Source Programme' as it is called, to an Object code that can be followed by the microcomputer. The computer employed for the conversion can be a microcomputer development system (Intellec) or a mainframe machine. Obviously, the development system, or mainframe, has to be programmed for this purpose. This is done by loading a programme called an 'assembler'.

A resident assembler is run on the microprocessor development system. A cross assembler is run on a mainframe computer.

Assembly language gives the programmer a great deal of freedom. For example, subroutines can be given names or Labels to enable them to be called when necessary without the programmer having to know where the subroutine is situated within the memory.

Assembly languages provide many other facilities to make programming easier.

The next, and last, step up the programming ladder, is the use of high level languages. For Intel machines, the language used is called PL/M and is written in Fortran.

Like the assembler, it is the function of PL/M to produce the binary coded object tape that can be loaded into a microcomputer. At the present time the translation between source and object code must be carried out on a mainframe computer.

The use of PL/M provides the programmer with advantages that can greatly reduce the time taken to write a programme. A disadvantage is that the object code output of a compiler is often longer than assembler produced object code for a given programme and therefore more memory is required in the micro-computer system to store it. This last statement assumes that the programmer is 100% efficient in his use of assembly language. For very large programmes, the use of a compiler language is likely to result in a better structuring of the programme since fewer instructions are involved (see later) and, therefore, could result in a shorter object programme. The choice between the use of an assembler and a compiler must be determined by the ability of the programmer, the precise application, and the size and complexity of the programme to be written.

A programme written in assembly language to move the larger of two numbers, A and B to location C, might be as follows:

```
TEST    SHL B              ;LOAD ADDRESS OF B
        LAM                ;LOAD B INTO ACCUMULATOR
        SHL A              ;LOAD ADDRESS OF A
        CMP M              ;COMPARE B WITH A
        JFC L1             ;JUMP TO L1 IF B≤A
        LAM                ;LOAD A INTO ACCUMULATOR
L1      SHL C              ;LOAD ADDRESS OF C
        LMA                ;STORE ACCUMULATOR IN C
        END
```

Nine assembly language instructions are needed. In PL/M the same programme could be written as a single statement:

```
IF A>B, THEN C = A; ELSE C = B
```

Both of these programmes will produce the object code that causes the microcomputer to carry out the desired function. However, the object codes from the two programmes will probably not be identical.

Another advantage of PL/M over assembly language is that, providing one understands PL/M, there is no need to have any knowledge of the architecture of the microcomputer. A PL/M programme written for one microprocessor could be re-compiled to run on another without revising the programme. In addition, PL/M enables such tasks as multiplication, division, etc. to be implemented without having to write all the individual subroutines.

ADVANCED SOFTWARE TECHNIQUES

by Bill Betts, GEC Semiconductors Ltd

Most engineers are introduced to the microprocessor as a component with the built-in ability to perform a dazzling number of tasks dictated by a sequence of commands or programme. In the beginning programming is taught or learned at the assembly language level, where each bit pattern to be interpreted as a command is represented by an easy-to-remember mnemonic. Programming at this level involves the programmer in the disciplines of allocating, maintaining and saving all registers and their contents and offers great scope for trivial, but hard to find, errors.

Switching to a high level language avoids this type of problem since the detail of the register and memory usage can be left to the compiler. High level languages are available to the 8080 microprocessor user and are the subject of other papers at the seminar. These are block structured languages whereby all the programme is segmented into self contained blocks of code. Parameters used within blocks can be common to all the programme or unique to the current block level. In a nested block situation, local names have limited validity in the next lower block level. This ability to nest blocks and in fact to call blocks within themselves can be very powerful, provided one is careful not to create an infinite loop.

When using high level languages one often requires to use previously written and proven routines. To use these sections of programme, one can copy them out at source code into one's programme before compilation. If one final programme is very large or uses a large number of names or labels, then the size of the compiler memory needed to run the compiler may be excessive compared even with the final object code requirements. For this reason it is convenient or even essential to compile the programme in parts and to link the resulting code blocks together. This linking requires not only that each new block is placed into addresses contiguous with the proceeding block, but also that this is correct for both read only and read/write memory. As an exercise in arithmetic this is not beyond the wit of most engineers, but the problem is compounded by labels and procedure calls. Whilst it is possible or even mandatory to define every procedure before it is called, labels can be, and often are, forward referencing and all these addresses relevant to the names and labels must be available to every section of the programme at compile time. An alternative to this is to format the output code from the compiler so that the addresses requiring external information for their formation are identified. By carefully defining the format and by using an intelligent loader programme the segments of the programme can be stitched together in any required order.

To expand the apparent size of a procession memory during programme execution one can resort to various techniques. The technique of overlaying part of the programme area with new programme data from a buck store is well known in computer systems where all memory is read/write. A similar trick can be played by writing one programme as an executive or supervisory programme and a set of operations. By configuring the operating parts of the programme carefully one can stack them one to a PROM chip. If the chip selects of these PROMs are driven from the bits of an output part then the sequence of operations can be controlled by the executive by outputting a defined bit pattern to this part and calling a routine which is always the same address. The selected PROM will respond and operate until the return instruction.

An extension of this technique is to interpret data as instructions. Obviously this 'data' needs to be carefully organized to make sense as instructions, but by running part of the programme from read/write memory the actual instruction sequence obeyed is a function of the data at run time. Such 'live' code is very powerful and if used with flair can be the trick that buys the memory space required, but any scheme whereby the programme itself is modified by the events taking place can be very difficult to debug since it may never be the same two 'looks' running.

HANDLING INTERRUPTS

by Bill Betts, GEC Semiconductors Ltd

In many microprocessor systems, the rate at which a peripheral device or devices can be serviced determines the total number of system tasks that can be assigned to the control of the microprocessor. The higher the throughput the more jobs the microprocessor can do and the more cost effective it becomes. Efficient management of peripheral servicing will have a significant effect on the overall cost performance of the microprocessor system.

A common method of servicing such devices is polling. Here the microprocessor must test each device in sequence and in effect "ask" each one if it needs servicing. It is apparent that such an approach could occupy a large portion of the main programme looping through a continuous polling cycle, thereby having a detrimental effect on system throughput and limiting the number of tasks that can be assumed by the processor.

Furthermore, the frequency of polling each device must be related to the response time required by that device. One must ensure that any input is acknowledged and acted upon before a second input appears on the same peripheral. This may mean that the polling routine must visit certain peripherals more frequently than others to satisfy their various needs.

A more desirable method is one in which the microprocessor can be executing its main programme and only stops to service a peripheral device when told to do so by the device itself. Effectively, any peripheral requiring attention would activate an asynchronous input to the processor that commands it to finish the instruction it is executing and fetch a routine to service the peripheral. Once the peripheral servicing is complete the processor returns to the main programme exactly where it left off and continues as if nothing had happened. The method is called 'interrupting' and it can drastically improve the throughput of a microprocessor system by limiting the time spent on peripheral operations to a barest minimum.

In a simple interrupt system every peripheral device has equal access to the asynchronous interrupt input to the processor. Once the processor has acknowledged the interrupt and has made records of all the current values of the internal registers so that it can recreate the pre-interrupt status, it must inspect the peripherals to determine which device caused the interrupt. By a simple one pass poll of the peripheral status or by reading a special interrupt status port the requesting device can be identified.

Inevitably, the time will come when two or more devices will simultaneously demand attention and such situations can only be resolved by allocating some priority to the inputs, either by the order in which the devices are scanned or by using a Priority Interrupt Control Unit (PICU) such as the Intel 8214. The PICU functions as an overall manager in an Interrupt driven system. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest priority, ascertains if the new interrupt has a higher priority than the interrupt currently being executed and issues a further interrupt on the basis of these decisions.

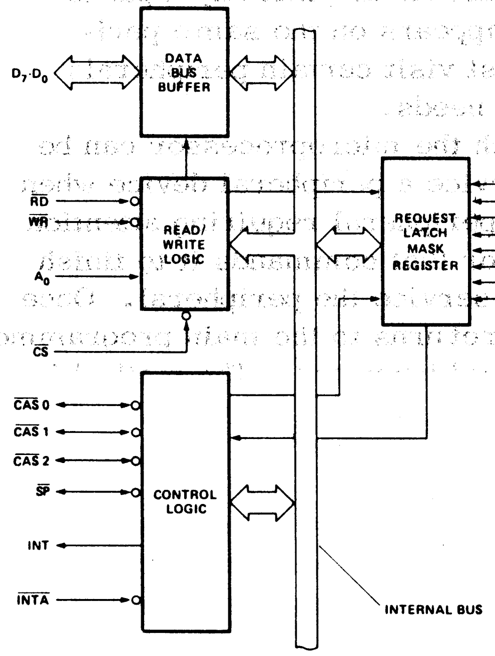
Each peripheral device or group of devices will have in memory a special section of programme, or routine, associated with its particular requirements. The PICU must indicate to the processor which service routine is required to execute the currently requested interrupt. This indication is given by issuing one of eight requests for the interrupt to "vector" to the correct Interrupt Service Routine.

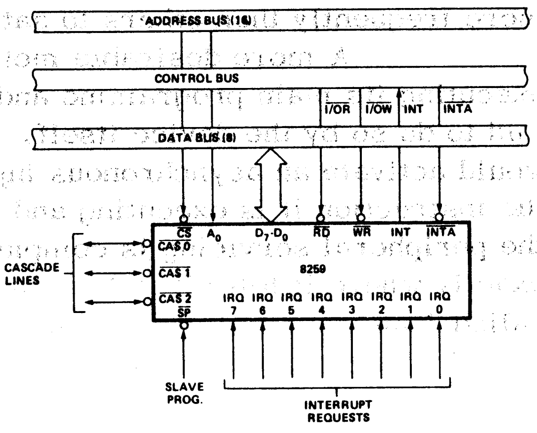
Additional possibilities include the masking of particular levels of interrupt either by additional hardware or via the Intel programmable interrupt controller 8259.

intel 8259

PROGRAMMABLE INTERRUPT CONTROLLER

BLOCK DIAGRAM





8259 System Interface.

A CORAL COMPILER FOR THE INTEL 8080 MICROPROCESSOR

by Bill Betts, GEC Semiconductors Ltd

With the coming of age of the 8080 microprocessor and its wide acceptability across the industry as a standard device, people are becoming increasingly aware of the advantages to be gained from not only standardizing on hardware but also on software.

Coral 66 is a general purpose programming language whose definition has been formalized and issued by the Ministry of Defence as an inter-services standard for military programming. It has also been accepted and adopted by many sectors of the control and automation industry.

The concept of transportability of programmes between various computers/processors is usually accepted as being a desirable feature, allowing, as it does, programmes developed on one system to be used directly on a different or improved machine. This concept allows considerable time and effort to be put into a sophisticated software system without the worry of future redundancy or obsolescence caused by hardware improvements. However, it is recognised that no standard language can be equally ideally suited to a number of processors and Coral 66 includes, for example, the ability to write in the main programme code statements in 8080 assembly language to make specific use of particular hardware features of one's design.

Coral 66 is a high level block structured procedure orientated language based upon Algol 60 but taking features from Coral 64, Jovial and Fortran. It is designed to be as general-purpose as is practical and is for use by skilled programmers. It has no inbuilt assumptions about specialized applications except, in its application to the 8080, that decisions are taken at compile time as to the allocation of storage between read only and read/write memory.

The use of any high level language in a real time as opposed to a problem solving or data movement situation is dependent upon an external presence of a set of routines handling the communications between the programme and the real time world. These procedures that provide this interface are not included in the Coral 66 language definition but can be easily created by reference to many standard handling routines mentioned in other presentations at this fair.

Some of the advantages of Coral 66 when applied to an 8080 microprocessor stem from the availability from other work previously carried out of proven routines for defined tasks and also a source of trained programmers. Other advantages of using Coral 66 over other high level languages must include its standardization by published "blue book" specification; full floating point arithmetic capability and the correct handling of signed quantities (both bytes and integers) in such things as loop counts, array efforts, multiply and divide. Other notable features could include conventional argument passing, two-dimensional arrays, macro facility, common declarations and the use of code to enhance performance.

The compiler is written in Coral 66 and so by compiling itself on itself arrives at a compiler written in 8080 assembly language. This enables the compiler to run on an 8080 system such as the Intel MDS. Intended primarily to run on a floppy disc based system the compiler will operate from paper tape in any 8080 system having at least 48K of read/write memory.

In the first releases of the compiler the output will be in 8080 assembly language. Whilst this requires the use of the MAC80 assembler to create the object code required to run the programme, it simplifies the linking of segments from various sources of code. Later editions will output directly in the Intel relocatable code format to be compatible with the relocating linker for load time allocation of addresses.

Which is the most cost-effective?

COMPILER OR ASSEMBLY LANGUAGE?

by Phil Pittman, Rapid Recall Ltd

When taking the decision to use either assembly language or the PL/M compiler for a microcomputer system design project, there are three main factors that have to be taken into account. These are the cost of writing the programme, the amount of memory needed to house the programme and the number of systems that are to be built.

If a large production run is envisaged the programme length is the most important factor since, if this can be reduced, a smaller programme store can be used and the resulting saving is multiplied by the number of systems that are to be built. For small production runs the cost of programme preparation (writing, debugging and documenting) is the dominant factor.

The PL/M compiler enables programmes to be written very quickly but, generally speaking, may make less efficient use of the programme store than programmes written in assembly language.

It is most useful to be able to calculate how many systems have to be built before the overall costs of programming in both PL/M and assembly language become equal, since this enables the most effective language to be chosen at the beginning of a project. The factors that have to be taken into account to perform this calculation are explained below:

P - Cost of programming per line. This will be the same for both PL/M and assembly language. Since a programmer will write about 25 lines of programme a day, it is realistic to set the cost per line at £5.

B_a, B_c	-	Number of bits generated by a line of assembly code (B_a) and a line of compiler code (B_c). Average values are $B_a = 16$, $B_c = 80$.
M	-	Cost of memory per bit (see Table)
E_c	-	Compiler efficiency (see Table). This factor compensates for the difference between the amount of memory for assembler code and the amount of memory for compiler code needed to implement a particular task, and will vary from programmer to programmer. In general, however, for systems below about 1K bytes E_c will be about 2 and will fall

towards 1 as the system increases in size to 4K bytes.

Table

Memory cost per bit (pence)	Compiler expansion factor (E_c)				
	1	1.1	1.25	1.5	2
EPROM 0.73	∞	334	128	60	26
PROM 0.47	∞	518	200	93	40
ROM 0.12	∞	2031	781	365	156

N - Number of systems to be produced.

L - Number of lines of programme.

From the foregoing it can be seen that:

Cost of memory = N x L x B x M
System cost = cost of memory + cost of programming (P x L)

When the system cost is the same in both assembler and compiler code the following equality applies:

$$N \times L_a \times B_a \times M + (P \times L_a) = N \times L_c \times B_c \times M + (P \times L_c)$$

Since the compiler requires a different number of lines to implement a system when compared to the assembler language, it is necessary to introduce the compiler expansion factor. Solving for N at the same time gives:

$$N = P(1/BA - E_c/B_c) \Big/ (E_c - 1)M$$

which is the number of systems at which the costs of programming in PL/M and assembler language are equal. If you are producing more than this number of systems it is less expensive to use assembly language.

If we assume that the compiler requires 25% more lines of programme than the assembler language and low volume production is being employed, it costs less to use PL/M for 780 systems in masked ROM or 128 systems in EPROM. For a very small system in which the worst case value of 2 is taken for E_c, it is cheaper to use PL/M for less than 150 systems in ROM or 26 systems in EPROM.

A general rule of thumb emerges for an average system. For a production run of less than 100 systems with EPROM, or 1,000 systems with ROM, it is better to use PL/M.

For very complex programmes it is usually better to use PL/M regardless of the production run, since inefficiencies are likely to be introduced by the programmer writing a long complex programme in assembly language.

3000 BIPOLAR MICROCOMPUTER OVERVIEW

by Phil Pittman, Rapid Recall Ltd

General purpose bipolar computer components, such as the Intel 3000 series, enable digital computers to be constructed with an architecture and an instruction set that is defined entirely by the designer. This means that the characteristics of the computer can be made to exactly match the requirements of the application.

The central processor unit of a general purpose digital computer can be divided into two sections: the arithmetic section and the control section. The control section fetches an instruction from the main memory and executes it by carrying out a whole series of sub-operations. In effect, the task is broken down into a number of small steps, each step being executed in turn by a sub-operation until the task is completed.

The instructions which define the sequence of steps necessary to carry out a main memory instruction are called micro-instructions, and the sequence of steps is called a micro-programme. The instruction which is fetched from the main mem ory is called a macro-instruction.

: The user of the computer writes a macro-programme that causes the machine to fulfil its intended function

: A macro-programme is made up from a sequence of macro-instructions

: Each macro-instruction is implemented in the computer by running a micro-programme

: The micro-programmes are written by the designer of the machine using micro-instructions

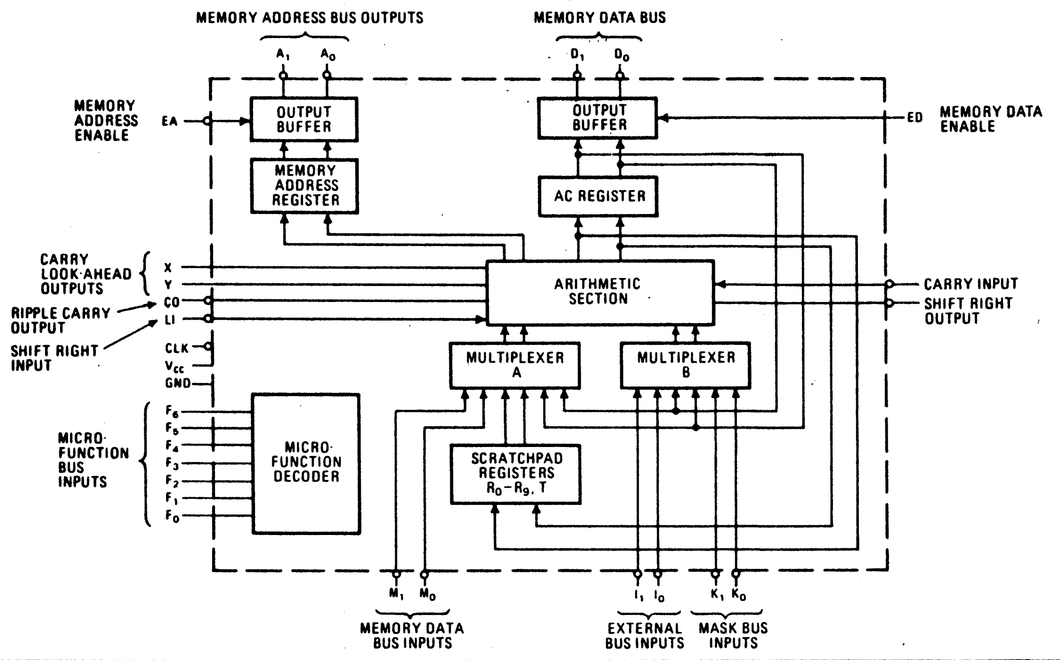
The user of the machine is not concerned with the micro-programmes which provide the instruction set that is available to him.

One family of bipolar microcomputer components is the Intel 3000 series, a set of components realized with Schottky TTL technology. The two most important circuits in this family are the 3001 micro-programme control unit (MCU) and the 3002 central processing element (CPE). The MCU determines the sequence of execution of micro-instructions from the control memory, and provides carry logic. The CPE represents a two-bit wide slice through the arithmetic, logic, register and data bus portions of a computer central processing unit. Several CPEs may be wired together to produce a central processing unit with arbitrary data bus width. For example, to produce a 16-bit wide data path, eight CPEs would be used.

Other members of the family include the 3003 fast carry chip, the 3212 input/output register chip, and the 3214 interrupt control chip. The control memory portion of a central processor or controller built with this family may be realized with standard field-programmable ROMs, mask-programmable ROMs or read/write memory (RAMs).

The micro-programmed approach is useful for bipolar microcomputers because complex macro-instruction sets can be realized as sequences of relatively primitive micro-instructions. The logic of the final macromachine remains relatively simple, with most of the complexity being represented by the contents of the control memory.

When using the Intel bipolar microcomputer family, the 3001 MCU implements most of the functions of a micro-programme control unit. When used with the 3002 CPE slice, the basic micro-instruction functions are established, although additional logical elements drawn from standard TTL families may be added which will alter or enhance the micro-instruction set.



Central processing element. This element contains all the circuits representing a two-bit-wide slice through a small computer's central processor. To build a processor of word width N, all that's necessary is to connect an array of N/2 CPEs together.

WHY USE A MICROCOMPUTER?

by Richard Forster, Jermyn Distribution

There are five basic reasons why the designer should evaluate microcomputers:

* They save development time and costs
* They reduce the number of different components that have to be bought, tested, stored and accounted for
* They will do the same job as a random logic system in a smaller space
* They reduce production costs
* They give better reliability

Most of these advantages stem entirely from the fact that the gates and maze of interconnections used in random logic systems is replaced by an ordered set of instructions which are stored in semiconductor read only memory as binary patterns. It is realistic to think in terms of a single 2K bit memory replacing about 28 mixed s.s.i., m.s.i. and l.s.i. packages in a typical system. The saving in circuit board area is immediately apparent. What is often forgotten is the fact that the number of interconnections between the board and i.c. package is reduced by about 60%. There is also an equivalent reduction in the number of boards inside the packages from the chips to the leadouts. The increase in reliability that results from just this factor alone is significant.

The systems designer used to thinking in terms of TTL has to exchange his knowledge and experience of Boolean algebra, Karnaugh maps and minimization for expertise in software. He has to learn about assemblers, simulators and PL/M. He has to acquaint himself with floppy discs and the MDS and has to delve into the mysteries of structured programming.

Engineers who have made the transition have found it enjoyable and interesting and many consider that the exchange of a soldering iron and a pair of snips for a keyboard for building a prototype system is a change for the better.

While programming can be learned from books it is far better to attend a proper course of instruction and to have the benefit of practical work whilst learning.

Once the necessary software expertise has been obtained, the designer will find that the time taken to develop a system will become shorter. He will also find, with careful board layout, that one microcomputer card will be suitable for many different projects. In most cases the changes will only concern the memory size, the type and configuration of the I/0 ports and the complexity of the interrupt structure. Some of these changes can be brought about with software alterations only.

With a microcomputer late changes in the specification of the product being designed can often be implemented in the software, and there may be no need to rework the board design at all. With random logic these last minute changes can be exceedingly expensive in both engineering effort and delayed product introduction.

MCS-80 MICROCOMPUTER SYSTEM

Type	Group	Description
8080A	CPU	Central Processor
8080A-1	CPU	Central Processor (1.3μs)
8080A-2	CPU	Central Processor (1.5μs)
M8080A	CPU	Central Processor (−55° to +125°C)
8224	CPU	Clock Generator
8228/8238	CPU	System Controller
8008, 8008-1	CPU	Eight-Bit Microprocessor
8702A	ROMs	Erasable PROM (256 x 8)
8708	ROMs	Erasable 1K x 8 PROM
8302	ROMs	Mask ROM (256 x 8)
8308	ROMs	Mask ROM (1K x 8)
8316A	ROMs	Mask ROM (2K x 8)
8101-2	RAMs	Static RAM (256 x 4)
8101A-4	RAMs	Static RAM (256 x 4) 450 ns
8111-2	RAMs	Static RAM (256 x 4)
8111A-4	RAMs	Static RAM (256 x 4) 450 ns
8102A-4	RAMs	Static RAM (1K x 1) 450 ns
8107B-4	RAMs	Dynamic RAM (4K x 1)
8222	RAMs	Dynamic RAM Refresh Controller
8212	I/O	8-Bit I/O Port
8255	I/O	Programmable Peripheral Interface
8251	I/O	Programmable Communication Interface
8205	Peripherals	One of Eight Decoder
8214	Peripherals	Priority Interrupt Control Unit
8216/8226	Peripherals	4-Bit Bi-Directional Bus Driver
8253	Peripherals	Programmable Interval Timer
8257	Peripherals	Programmable DMA Controller
8259	Peripherals	Programmable Interrupt Controller

THE CONSTRUCTION AND TESTING OF PROGRAMMES

by J.L. Payne, CAP Microprocessor Group, Computer, Analysts and Programmers Ltd
CAP House, 14-15 Great James Street, London, WC1 N3DY
Tel: 01-242 0021

There is a difference between programme testing and programme debugging. Programme testing is the process of checking that the programme works in all cases in which it is supposed to work; programme debugging is the process of locating and correcting programme bugs indicated during and after testing. The best way to debug is to minimise the need for it, and this is achieved by proper programme design.

Programme Design and Construction

The design of a programme is based on a Functional Specification. This defines in an unambiguous way actions to be taken by the programme for each event. In other words it defines what the programme has to do. The Programme Specification is a document which describes how the programme is constructed. These two documents, together with the programme listing, form the major part of the programme documentation.

The most important part of programme design is the structuring of the programme and data. Programmes should be broken down into logical or functional units. Such units or modules will often be subroutines and communicate, where possible, by parameter passing or in some cases via global data (data common to all subroutines).

Subroutines should be constructed with only one entry and exit point, and should always return control to the point following their call. They should be designed independent from a given set of data. This generalised approach often enables them to be used elsewhere in the programme, and it helps anticipate future modifications and adaptations. Such generalised routines can contribute to a library to assist future projects.

Simplicity of coding is another aspect of programme design to be stressed. Unnecessarily complex logic or tricky coding taking advantage of some obscure processor dependent feature is to be avoided as this can impede programme checkout. Future maintenance is almost certain to involve programmers who are unfamiliar with the coding.

The correct structuring of data is also important. Fixed parameters and variable data must be separated as the former will be stored in PROM and the latter will reside in RAM in most microprocessor systems.

The goal of structured programming is to organize and discipline the programme design and coding process in order to prevent more logic errors and to detect those remaining errors. The two most important features of this approach are (1) top-down design and (2) modular design. Top-down programme design is similar to top-down report writing.

Reports are structured hierarchically and written from the top of the hierarchy, that is, starting with a brief synopsis. Much of the work on structured programming has suggested that programme correctness is more likely simply because of the way the programme is developed.

The use of fairly detailed flowcharts is also to be encouraged. They assist in the design of code segments with a minimum number of internal paths. Ideally such segments should be one page long.

With microprocessor systems the choice of languages will often be restricted. There are, however, three language levels possible; machine (in hexadecimal, octal or binary), assembler (mnemonics) or high level (e.g. PL/M).

Machine level programming is always to be discounted. It has no place in a professional approach to programming. So the choice will be between, say, PL/M or 8080 assembler. Three important points to remember here are:

* High level languages are not a panacea - programmes must be properly structured.

* High level languages reduce programme coding time.

* Efficiency of compiled code is often unimportant.

Testing and Debugging

When designing a programme the programmer should pay constant attention to how he will test it. The ultimate result of a programme which is difficult to test is that it is never completely tested.

"Bloody instructions which, being learned,
return to plague the inventor." - Macbeth.

It is often helpful to include diagnostic features in a programme, particularly in areas where programmes interact with hardware. A typical facility might be to provide a count of the total number of parity errors or a count of time spent waiting for some I/0 operation to complete. The intelligent use of flags to mark access to subroutines or data areas can often pay off. Saving interface status information before initiating a command sequence to that interface can assist in debugging hardware/software interactive problems.

There are two major groups of tests to perform when adopting a progressional approach to testing and debugging. The first group is known as unit testing. This is testing individual or groups of subroutines or modules in isolation. Data are selected to test all possible combinations of events that can occur. The outputs are monitored and compared with the pre-determined results. The tests must be methodical and repeated after errors have been found and corrected. The process is clearly iterative.

The second group is known as integration testing. Hopefully at this stage most of the bugs associated with isolated modules will have been found. Integration testing is the bringing together and testing of the total programme (all modules). For most microprocessor programmes this stage will include the full hardware configuration. Unit testing, however, will take place in an isolated environment using either a test harness (a control programme) or a simulator. The additional unknown of new hardware increases the argument for diagnostic features built into the programme.

When a bug is found during integration testing and the faulty module corrected, it is necessary to go back and unit test that module again before proceeding with integration testing.

The use of up-to-date listings and symbol tables is crucial for integration testing. When used in conjunction with an Intellec system an interactive on-line approach to testing and debugging is possible.

Such features as memory inspection and insertion, programme tracing and breakpoint insertion enable the Intellec user to interact with the running programme directly.

Conclusion

The paper has attempted to show the major steps in professional programme testing and debugging. It has emphasised strongly the initial design and construction as major factors in programme debugging and laid guidelines for the testing methodology. Such a short paper cannot deal with the mechanics of testing and debugging in more detail.

THE INTELLEC® MDS IS THE WORLD'S FIRST IN-CIRCUIT MICROCOMPUTER DEVELOPMENT SYSTEM. IT HELPS SAVE CONSIDERABLE TIME AND MONEY IN THE DESIGN, DEVELOPMENT, DEBUGGING AND MARKETING OF MICROCOMPUTER-BASED PRODUCTS.

- Use of the Intellec® MDS can help manufacturers get to the marketplace with their products months ahead of the time required with other development methods. Larger shares of market can be captured earlier, and held, and the effects of price erosion can be minimized because of the very fast product development enabled by the MDS system.
- A user can tailor his Intellec® MDS development system to his own unique requirements. If he wishes, he can begin with the basic MDS system and expand to the ultimate design configuration in a series of modest capital investments.
- The sophisticated MDS system can eliminate literally thousands of man hours normally required in research, product development, engineering and programming. The cost and time savings provided by the use of this new development system can be most significant.
- The Intellec® MDS system can be used immediately to design, develop and debug products based on either the Intel® 8080 CPU or the Series 3000 Schottky bipolar microcomputer family.
- The new MDS system can be used for any number of new, user-developed products or systems, because its support is not limited to any one Intel® microcomputer family.
- In a single, self-contained Intellec® MDS system, all the hardware and software required for efficient, cost-effective product development is supplied.
- The MDS supports all program development, prototype debugging and production and field testing of microcomputer-based products. This single-system support eliminates the need of separate systems to perform these functions.
- The internal capabilities of the Intellec® MDS eliminate the need to use dial-it-up time-sharing computer services and their associated costs. In addition, the delays often caused by in-house computer service are eliminated by the self-contained MDS system.
- Product development projects are fully supported by Intel® in the form of the most extensive hardware and software field applications assistance in the industry, international field service, and comprehensive documentation.
- The Intellec® MDS Microcomputer Development System is available now, enabling users to design, develop and debug their products much faster, at considerably less expense, than was ever before possible.

MICROPROGRAMMING THE 3000 SERIES

For some time, microprogramming has been recognized as a powerful technique for the design of a complex processing system such as the central processing unit of a general purpose computer. Recent advances in high-speed LSI circuit technology have made it possible and economically practical to apply a microprogramming solution to a wide range of design problems.

A microprogramming approach to a design problem requires two distinct steps to achieve the design objective:

1) Design a microprogrammable processor capable of meeting the design objective.

2) Design a microprogramme that will direct the activity of the processor to satisfy the design objective.

The first of these steps is a logic design problem; the second step is a programming problem. The Intel Series 3000 Bipolar Microcomputer Set supports the first aspect. CROMIS, the Cross Microprogramming System, supports the second.

Series 3000 Bipolar Microcomputer Set

The Intel Series 3000 Bipolar Microcomputer Set is a complete, compatible family of high performance LSI circuit components that serve as the basic building blocks for custom microprogrammed processors and controllers. The set's two major components, the 3001 Microprogramme Control Unit (MCU) and 3002 Central Processing Element (CPE) establish the foundation of a flexible microprogrammable architecture. The user can build on this foundation to meet the requirements of a wide variety of special applications.

Although a user's configuration is designed to meet the requirements of his particular application, it is worthwhile to consider how the Series 3000 elements might be applied in a typical microprogrammable processor-controller implementation. The configuration illustrated in Figure 1-1 may differ in some ways from the one a user is microprogramming. However, a discussion of Figure 1-1 identifies the roles played by the Microprogramme Memory, the Microprogramme Control Unit (MCU) and the Central Processing Element (CPE), which are fundamental to the architecture of any Series 3000 configuration.

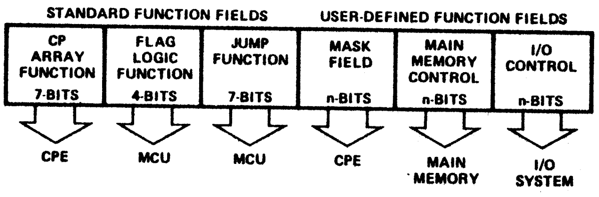
The microprogramme memory may be viewed (see Figure 1-2) as an array of locations, each providing storage for one unit of control information: a microinstruction word. Each location is identified by a unique address. An address applied to the address inputs of the microprogramme memory selects a location. The bit pattern stored in the selected location (i.e. the microinstruction word) appears in the form of a pattern of binary signal levels on the control lines connected to the data outputs of the microprogramme memory. Thus, each bit in the selected microinstruction word determines the state of one control line.

A group of related control lines forms a function bus. Each function bus controls the operation of, or supplies data to, a functional unit in the system. The grouping of the control lines into function busses imposes natural functional divisions on the microinstruction word. These divisions are called 'fields'. Each field of the microinstruction word drives a single function bus and thereby governs the behaviour of one functional unit. Therefore, each field of a microinstruction word can be viewed as providing an instruction to be executed by one functional unit in the system. In other words, the microinstruction word consists of a fixed group of instructions that are executed in parallel by corresponding functional units when the microinstruction word is selected from the microprogramme memory.

During a basic operating cycle, called a microinstruction cycle, an address is applied to the microprogramme memory, selecting one microinstruction word for execution. The fields of the selected microinstruction word dictate the functions to be performed or initiated by the functional units in the configuration during that cycle.

One of the microinstruction word fields controls the operation that determines which microinstruction word will be executed during the next microinstruction cycle. Another field specifies the operation to be performed by the data processing section (the CPE array). Still other fields may control an external main memory and I/O devices.

The design flexibility of the Series 3000 computing elements makes it impossible to describe every format exactly. However, the microinstruction word format below reflects the general requirements of the typical configuration illustrated in Figure 1-1:



The CPE is a complete 2-bit data processing module. It includes a number of general and special purpose registers, arithmetic and logic circuits, and several busses for data input and data output, as illustrated in Figure 1-3. Virtually any number of CPE's may be connected, as shown in Figure 1-4, to implement a data processing section of any desired word width. For example, 8 CPE's may be arrayed to produce a 16-bit processing section.

As CPE's are wired together, all registers, arithmetic and logic circuits, and data paths expand accordingly. However, the seven function inputs (F0-F6) to each CPE are connected in parallel to form a single 7-bit CPE Function Bus for the entire array. This arrangement allows the microprogrammer to view the CPE array as a single functional unit capable of executing a variety of data processing functions on N-bit operands.

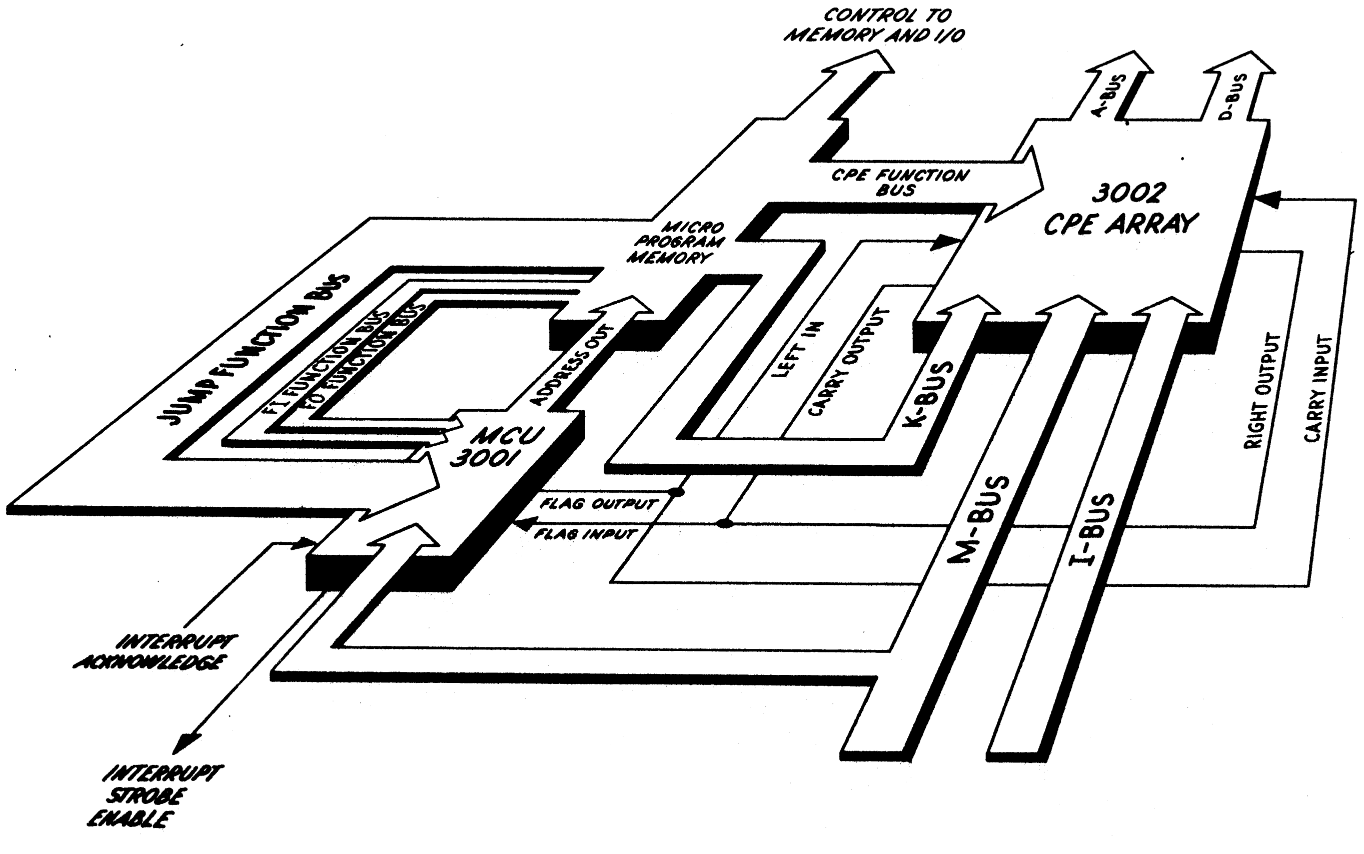


Fig. 1-1. Typical Series 3000 Configuration

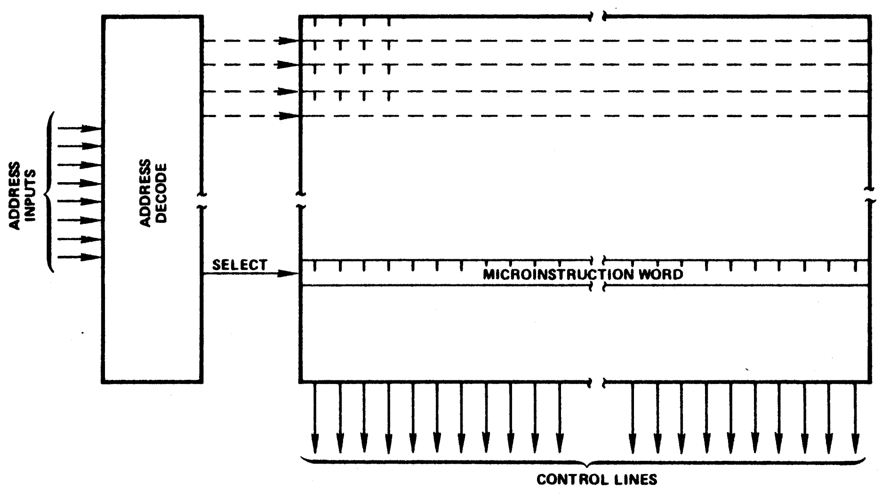
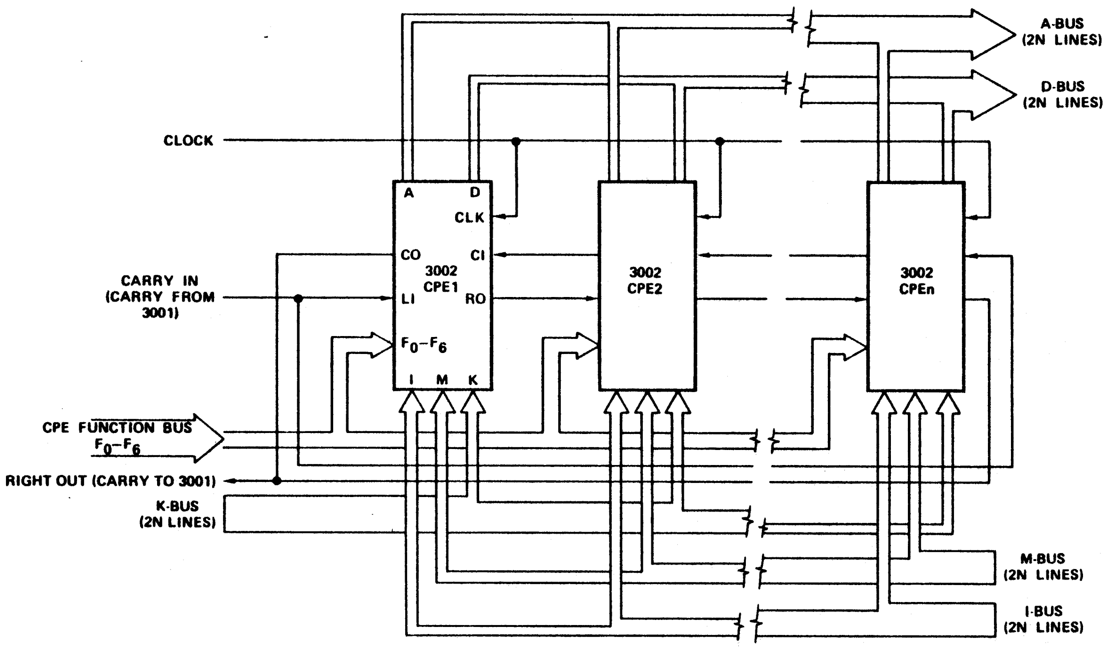


Fig. 1-2. Microprogram Memory



AN INTRODUCTION TO PL/M

There are two main approaches that can be taken to writing a programme for a microcomputer. The programme can be written either in assembly language or in a high-level language like PL/M. In some applications it may be beneficial to write some sections of the programme in assembly language and the remainder in PL/M. This short article is intended to form an introduction to PL/M, to describe some of its characteristics and to highlight some of its particular advantages.

PL/M, which stands for Programming Language for Microcomputers, is a subset of the IBM language PL/1. A single line of PL/M will generate the same amount of object code as several lines of an assembly language. Source programmes written in PL/M are therefore much shorter to write than the equivalent assembly language programme. Programming costs have been shown to be proportionally lower.

Source programmes written in Assembly language can be assembled to produce the object code either by a microcomputer or a mainframe. At the present time PL/M programmes can only be compiled by a mainframe computer.

A programmer writing a PL/M programme need not have a detailed knowledge about the architecture or the mnemonic instruction set of the microcomputer he is writing the programme for; in fact, a PL/M source programme can be compiled for use on either the 8080 or the 8008. However, the PL/M programmer need not hold himself aloof from the nitty gritty of the cpu internal operations if he does not want to. For example, he still has the power to manipulate the stack pointer just as he has when writing in assembly language. He can also call utilities (sub routines) that have previously been written in assembly language.

The following PL/M programme fragment is a sub-routine (called a procedure in PL/M) to be run in an Intellec system which calls the character print routine that is resident in the Intellec monitor at 3809H

```
                                        /*THIS IS A COMMENT*/
PRINT$CHAR: PROCEDURE(CHAR);
    DECLARE CHAR BYTE;          /*CHAR IS AN EIGHT-BIT QUANTITY*/
    DECLARE IOCO LITERALLY '3809H';/*IOCO IS MONITOR PRINT ROUTINE*/
    GO TO IOCO;                 /*PRINT THE CHARACTER*/
    END PRINT$CHAR;
```

The character to be printed is passed to this procedure when the procedure is called:

```
CALL PRINT$CHAR('A');  /*PRINT A*/
```

If desired, a second procedure could be written which would call PRINT$CHAR and use it to print a string of characters of variable length. The address of the first character in the string and the number of characters in the string will be passed to the procedure when it is called:

```
PRINT$STRING: PROCEDURE(NAME,LENGTH);
              DECLARE NAME ADDRESS,   /*NAME IS A 16-BIT QUANTITY*/

              (LENGTH,I,CHAR BASED NAME)BYTE;/*LENGTH,I AND CHAR
                                               ARE ALL 8-BIT
                                               QUANTITIES. IN
                                               ADDITION,CHAR IS A
                                               BYTE WHICH IS POINTED
                                               AT BY THE ADDRESS
                                               POINTER NAME*/

              DO I = 0 TO LENGTH-1;  /*CONTIN UE UNTIL THE NUMBER OF
                                       CHARACTERS SPECIFIED BY
                                       LENGTH - 1 HAVE BEEN PRINTED*/
              CALL PRINT$CHAR (CHAR(I )); /*PRINT THE CHARACTER
                                            POINTED AT BY NAME + I
                                            (I IS INCREMENTED ON EACH
                                            ITERATION OF THE 'DO LOOP'*/
              END;    /*END OF DO LOOP*/
END PRINT$STRING;
```

This routine could be made to print the text TEMPERATURE in the following way:

```
CALL PRINT$STRING (.TEMPERATURE,11); /*THE '.' CAUSES THE POINTER
                                       'NAME' TO BE SET AT THE FIRST
                                       CHARACTER 'T'. 11 IS THE
                                       LENGTH*/
```

The examples have served to illustrate what a PL/M source programme looks like and it can be seen that the programmes are easier to read and understand than an assembly language programme.

When a PL/M programme is compiled it will probably generate more object code than a well written assembly language programme which does the same job. In other words, a PL/M programme will increase memory cost because it will require more ROM. On the other hand, it will cost far less to write. Obviously, these costs will be equal when the cost of the amount of extra ROM required, multiplied by the number of systems to be built, is equivalent to the difference between the cost of programming in assembly language and PL/M.

As a general rule of thumb, for an average microcomputer system, with a production run of less than 100 systems when using EPROM, or less than 1,000 systems using ROM, it is more cost effective to use PL/M.

The PL/M compiler is available on several time sharing services in the U.K. You can rent a terminal from Timesharing Ltd, Tymeshare and Honeywell Ltd and start writing microcomputer programmes with very little capital outlay. Obviously, when you get down to hardware development and hardware/software integration, a development system, such as the MDS, will be necessary.

PL/M T.M. HIGH LEVEL PROGRAMMING LANGUAGE MCS-8 T.M. AND MCS-80 T.M. CROSS COMPILERS

Reduces program development time and cost
Improves product reliability and eases maintenance
Available for 8008 and 8080
Comprehensive user documentation
Hexadecimal or BNPF object code formats
Written in ANSI standard FORTRAN IV
Instantly available on worldwise timesharing services

PL/M is a high-level system programming language, specifically designed to ease the programming task for INTEL's 8-bit microcomputers, the 8008 and the 8080. PL/M is a powerful tool, well suited to the requirements of the microcomputer system designer and implementor. The language has been designed to facilitate the use of modern techniques in structured programming. These techniques can lead to rapid system development and checkout, straightforward maintenance and modification, and high product reliability.

The PL/M compilers convert a free-form symbolic PL/M program into an equivalent 8008 or 8080 object program. The compilers themselves take care of all the details of machine or assembly language programming, which permits the programmer to concentrate entirely on effective software design, and the logical requirements of his system.

Output from the PL/M compiler may be loaded directly into the 8008 or 8080 simulator programs for interactive, symbolic debugging or may be punched to paper tape in hex format for loading into an Intellec® Microcomputer Development System. It may also be punched in BNPF format to program ROMs.

The PL/M compilers are written in ANSI standard FORTRAN IV and are designed to run on any large-scale computer system with a minimum 32-bit integer format (word size). They are also available for immediate use on several worldwide timesharing systems.

Programme Library Enlarged Threefold

During recent months we have increased the number of programmes in our library more than three-fold. These programmes are available to all users of our microcomputers and can save a great deal of time as there could be a routine in the library which you could adapt for your latest project. You can join the library for a period of one year free of charge by submitting an acceptable programme for inclusion or by paying a fee of £60. Some of the programmes currently available in the library are:

8-bit Library

Cross products

Cross assembler for HP 2100 DOS
Cross assembler for NOVA 1200
8080 Cross compiler for PDP-11
Cross assembler for HP-3000
PLM83 (PL/M 80, pass 3)
PDP-8 Cross compiler
Cross assembler for IBM 360 and NOVA 1200
Cross Assembler for PDP-11
Macro Assembler for PDP-11
Absolute Loader for PDP-11
8008 Macro Definition Set for Assembly on PDP-11
8080 Macro Definition Set for Assembly on PDP-11
Cross Assembler for Nova 1220, IBM 360/40 and CDC3300
Nova Cross Assembler for Intel 8080
Intel 8008 Cross Inverse Assembler for HP 2100

General

Re-entrant interrupt handler
Paper tape labeller
8080 Triple byte counter
Seven segment hex display
Banner print and punch
Low pass filter
Page Listing Programme
Load to buffer and save pointer: Retrieve from buffer by pointer
Clock subroutine
Interrupt driven clock routine
Comparison of two memory fields
Octal console emulator
Data array move
Octal PROM programming
Digital-to-analogue conversion for eight outputs
Terminal editor
Compare register pairs BC and DE
8080 Idle analyzer for approximate cpu utilization
MAC8080 Demonstration programme
Shellsort (array sorter)
8080 Macro definition set
Reformat of PL/M80 assembly listing
Calendar subroutine

Mathematical

8080 Floating point with bcd conversion
Fast conversion: binary to bcd
Floating point multiplication
Signed magnitude bcd addition
ASCII to EBCDIC Conversion
Calculator control programme for Intellec 80
8080 Least squares quadratic fitting
Random number generator
Natural logarithm
Subroutines to find sin *x* and cos *x*
PL/M Histogram procedure and random number generator
Fixed and floating point routines
Fixed point square root
8-bit psuedo random number generator
Binary to bcd subroutine
Fast floating point square root routine
Natural log routine
BCD Multiplication
Sin *x*, cos *x* routines
n-Byte binary multiplication and lead zero blanking
Decimal multiply subroutine
Hex to decimal conversion routine
ASCII to hex check and conversion

Operating, testing and debugging

Binary punch/read routine
Tally R2050 high speed reader driver
Integrate Tally 2200 printer to an Intellec 8/MOD 80
Symbol table list generator
8080 Debug programme
Memory check for static and dynamic RAMs
CRC Table generator
CRC Calculator
Assembler oriented line printer handler
High-speed paper tape reader with stepper motor control
Time sharing communications
Driver for very high-speed reader
CRC Generator
Octal debugging programme for MCS-80
Operating system to replace Intel monitor
Symbolic microcontroller assembly language
Digital filter programme
16-bit CRC routine
Trace programme
Table look-up and build-up routine for 8080
Reader test
RAM Diagnostic routine for 8080
Interrupt service routine
8080 Dis-assembler
8080 System loader
DISASM 8080 Dis-assembler
Transfer parameter addresses between calling programme and subroutine
Elementary function package
Memory diagnostic programme
CRECH (CRC for IBM floppy disc)
Paper tape to mag cassette
I Command (loads hex into sequential locations)
Compare object tape with memory
DEBUG (two PROM packages for minimum 8080 systems)
TTY Reader/punch test
Centronics model 101 printer handler
TI ASR733 Terminal editor
TI Silent 700 to Mod 80 interface
BOOT (Bootstrap programme loader)
Routine to write/read blocks of data on standard audio cassette recorder
IBM Selectric 731 output programme

Games

Blackjack	Maze
NIM	Game of life
The word game	Numbers
Gambol	Kalah
Mastermind	

4-bit Library

Cross Assembler for PDP-8
Cross Assembler for NOVA computer
BNPF Tape Generator for PDP-8
MCS-4 Simulator for PDP-8
Chebyshev Approximation Package
Parity Checker/Generator
Parity Generator, ASCII Character
Code Conversion: ASCII to EBCDIC
Delay subroutines
Bit Manipulation Routine
Universal Logic Subroutines
8-bit Parity Check Annex
Binary to BCD Converter
Data Compare
Paper Tape Edit

Introduction to Microcomputers

THE COMMERCIAL ADVANTAGES

by Keith Chapple, Intel Corporation (UK) Ltd

The reasons why the microcomputer has so rapidly become successful are threefold:

1) They save money and time.
2) They can achieve a very high performance and result in equipment that is versatile.
3) They have captured the imagination of engineers who, by and large, enjoy working with them.

In this paper we will discuss point 1. Other papers adequately cover point 2. Point 3 is evident from the correspondence we receive and the discussions we have with our customers.

The microcomputer provides the means to replace randomly connected logic gates and flip flops with a sequence of instructions that is stored in some form of read only memory. Experience has shown that between 8 and 16 bits of memory are the logical equivalent of an average gate. Since an average TTL i.c. contains the equivalent of 10 gates, it takes between 80 and 160 bits of ROM to replace a TTL package.

ROMs are currently available with capacities of up to 16,384 bits; so a single ROM of this size will replace between 100 and 200 packages. At the other end of the spectrum, a 2,048 bit ROM will replace between 13 and 25 TTL packages.

It is not difficult to see that the package count in a microcomputer system will be far lower than in an equivalent TTL system. Obviously, the microcomputer system requires a cpu, I/0 ports etc. which require a few packages in addition to the ROM, but overall the package count will be considerably less.

Fewer packages mean smaller circuit boards, smaller cabinets, fewer cooling problems, fewer interconnections, fewer connectors, lower power consumption, increased reliability and lower manufacturing cost. In addition, the work load on support departments such as Q and A, goods inwards, etc. is much reduced.

The overall reduction in manufacturing costs can be very substantial; in fact customer estimates indicate savings in the electronic portions of equipment ranging from 20 to 80%.

The rapid acceptance of microcomputers and the large number of applications which are currently using them in equipment have enabled us to invest large sums of money in the development of new components, development aids and software. This investment directly reduces the cost of developing microcomputer based systems and can greatly shorten the time it takes to actually get your system on the market.

Customers indicate that product development cycles have been reduced by between six and twelve months. At the start of a new project product definition is often speeded up once the decision to use a microcomputer has been taken because the incremental cost of adding additional features is usually small and can be easily estimated. These additional features may, in themselves, be quite major, but may only require a longer programme and perhaps an extra ROM in the final product. By the same token, through software changes, a single identical circuit board can be used as the basis for a whole range of products; each product having its own characteristics which may be very different from others in the range.

In summary, the use of a microcomputer is likely to provide you with the following advantages:

: Simple, and less, hardware

: One circuit board may be used for many different applications

: Lower production costs

: Increased reliability

: Reduced power supply, cooling and cabinet requirements

: Reduced work load on QA, board layout, incoming inspection and other support requirements

: Easy to make engineering changes

: Easy to customize equipment

: Easy to provide more facilities

: Reduced development time

: Reduced development costs

: Fewer field failures (not so many components)

: Fewer warranty problems

: Increased profitability

8080A OVERVIEW

by Howard Kornstein, Intel Corporation (UK) Ltd

The 8080 is a complete family of components that enable powerful 8-bit microcomputers to be constructed. The family is supported by a comprehensive and sophisticated set of hardware and software aids including the MDS and its peripherals which are the subject of a separate paper. 8080 systems can also be obtained ready-made for OEM applications (SBC-80/10) or in kit form (SDK-80) for one-off use or for educational purposes.

At the present time the 8080 component family comprises three 8080A cpu's which offer a choice of operating speed and temperature range, a clock generator, system controller, 12 peripheral i.cs (five of which are programmable), 8K EPROMs, 16K ROMs, 1K CMOS RAMs and 4K n-mos RAMs.

Software available for the 8080 includes a resident assembler and editor, a variety of cross assemblers and editors, a cross compiler for the high-level language PL/M and a disc operating system.

The 8080A itself is an 8-bit parallel central processor for use in general purpose digital computer systems. It is made on a single chip using our n-channel silicon gate n-mos process.

The 8080A contains six 8-bit general purpose working registers and an accumulator. The six general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset four testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, programme counter and all of the six general purpose registers. The sixteen-bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microcomputer has been designed to simplify systems design. Separate 16-line address and 8-line bi-directional data busses are used to facilitate easy interface to memory and I/0. Signals to control the interface to memory and I/0 are provided directly by the 8080A. Ultimate control of the address and data busses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for direct memory access (DMA) or multi-processor operation.

A computer, no matter how sophisticated, can only do what it is "told" to do. One "tells" the computer what to do via a series of coded instructions referred to as a Programme. The realm of the programmer is referred to as Software, in contrast to the Hardware that comprises the actual computer equipment. A computer's software refers to all of the programmes that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The cpu is designed so that a specific operation is performed when the cpu control logic decodes a particular instruction. Consequently, the operations that can be performed by a cpu define the computer's Instruction Set.

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logical operations (e.g. OR the contents of two registers) and register operate instructions (e.g. increment a register) are included in the instruction set. A computer's instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/0 device. Most instruction sets also provide Conditional Instructions. A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a programme with a decision-making capability.

By logically organizing a sequence of instructions into a coherent programme, the programmer can "tell" the computer to perform a very specific and useful function.

The computer, however, can only execute programmes whose instructions are in a binary coded form (i.e. a series of 1's and 0's) that is called Machine Code. Because it would be extremely cumbersome to programme in machine code, programming languages have been developed. There are programmes available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is Assembly Language. A unique assembly language mnemonic is assigned to each of the computer's instructions. The programmer can write a programme (called the Source Programme) using these mnemonics and certain operands. The source programme is then converted into machine instructions (called the Object Code). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an Assembler programme. Assembly languages are usually machine dependent (i.e. they are usually able to run on only one type of computer).

The 8080 instruction set includes five different types of instructions:

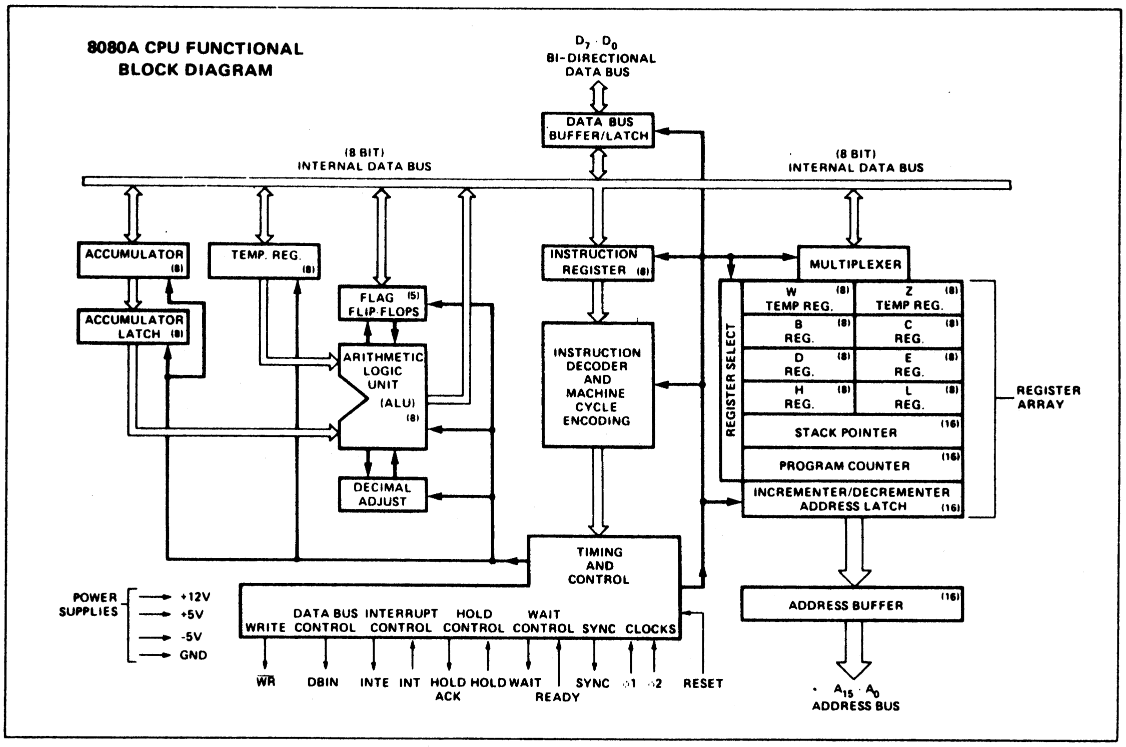
: Data Transfer Group - move data between registers or between memory and registers.

: Arithmetic Group - add, subtract, increment or decrement data in registers or in memory.

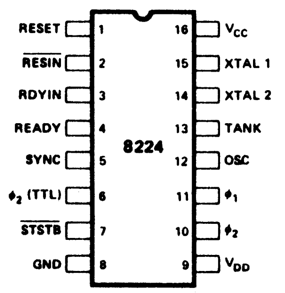
: Logical Group - AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory.

: Branch Group - conditional and unconditional jump instructions, subroutine call instructions and return instructions.

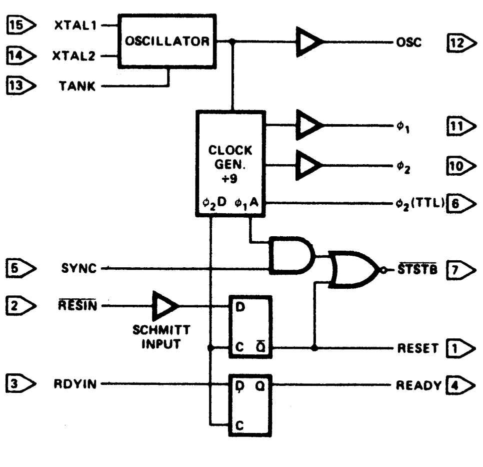
: Stack, I/0 and Machine Control Group -includes I/0 instructions, as well as instructions for maintaining the stack and internal control flags.

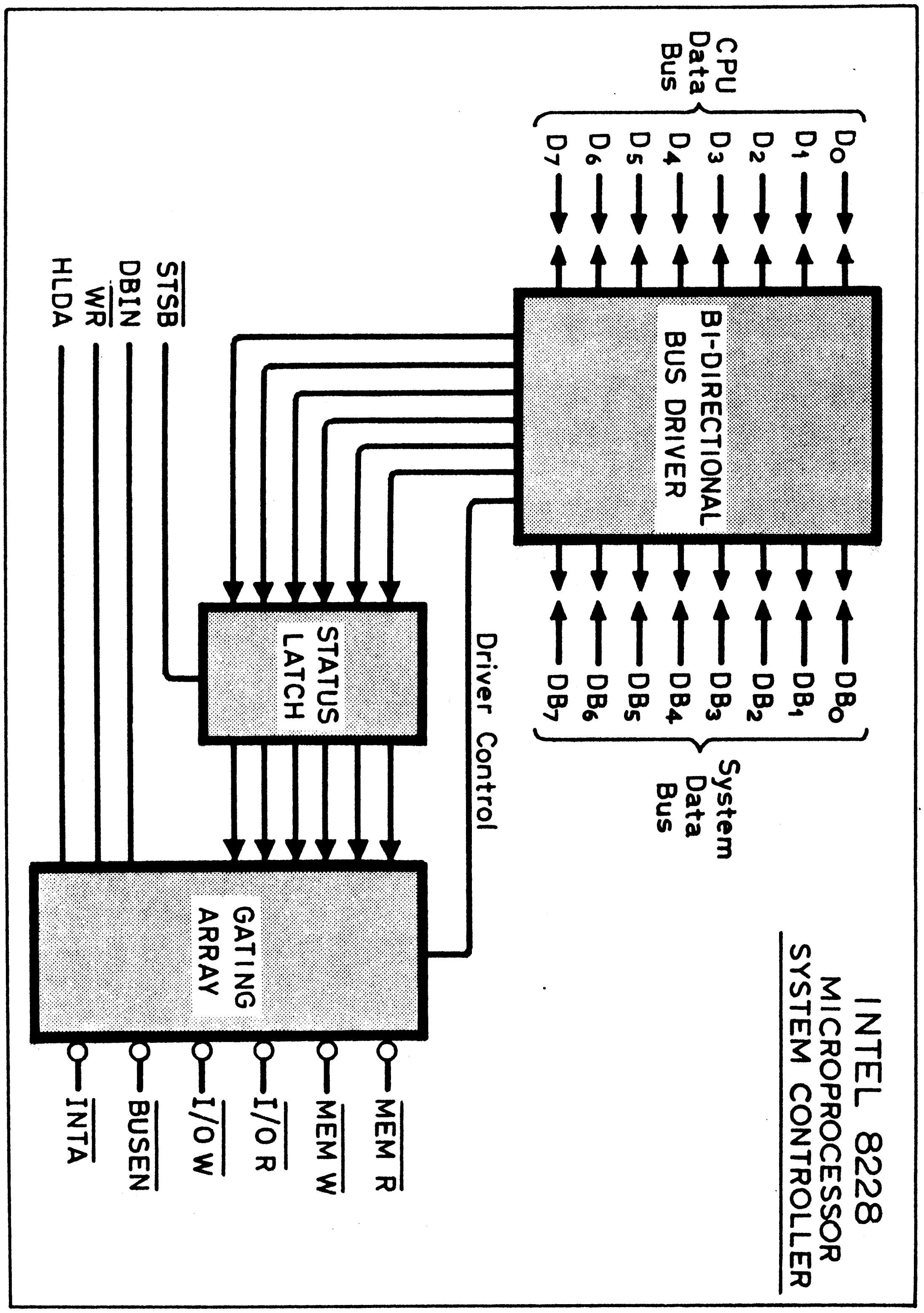


PIN CONFIGURATION



BLOCK DIAGRAM



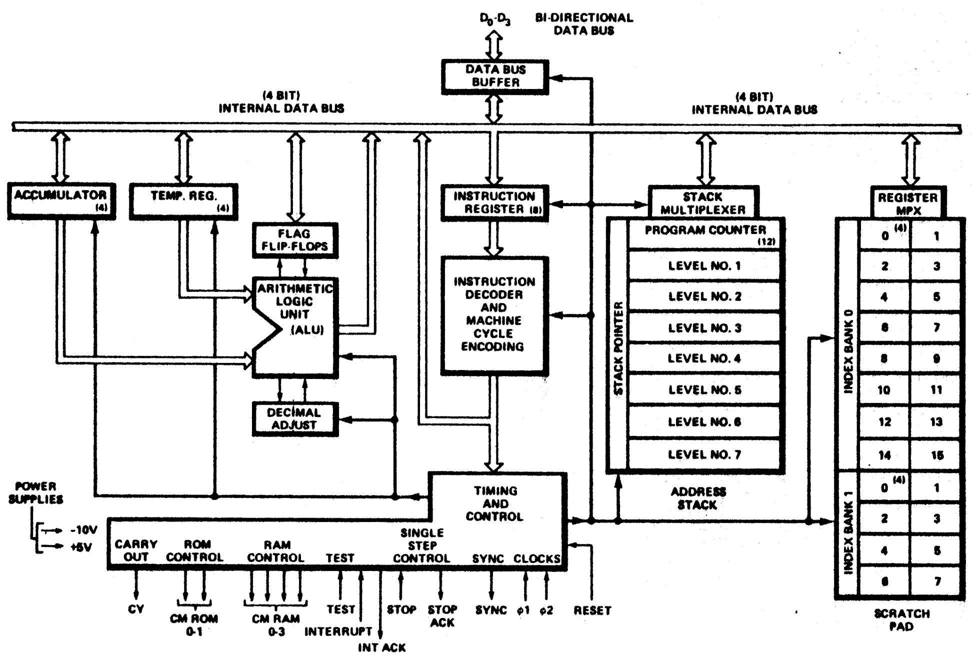

INTEL 8228
MICROPROCESSOR
SYSTEM CONTROLLER
CPU Data Bus
D0
D1
D2
D3
D4
D5
D6
D7
BI-DIRECTIONAL BUS DRIVER
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
System Data Bus
Driver Control
STATUS LATCH
GATING ARRAY
STSB
DBIN
WR
HLDA
MEM R
MEM W
I/O R
I/O W
BUSEN
INTA

4040 OVERVIEW

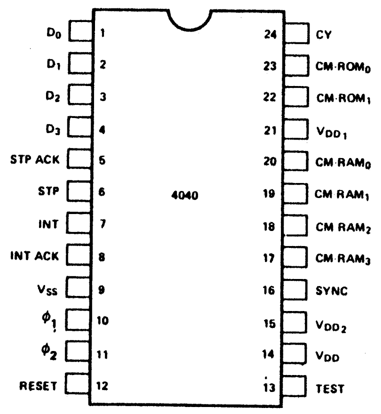
The Intel 4040 is a complete 4-bit parallel central processing unit which has been designed to be a replacement for random logic. The cpu can directly address 4K eight-bit instruction words or 8K with a bank switch. Seven levels of subroutine nesting, including interrupt, and 24 randomly accessible index registers (24 x 4) are provided which may be used for addressing or for scratch pad memory for storing computation results. The interrupt feature permits a programme sequence to be interrupted, with normal programme execution continuing after the interrupt service routine is completed. Provisions have also been made to permit single-stepping the cpu using the 'Stop' and 'Acknowledge' signals.

The 4040 is an enhanced version of the 4004 and as such retains all the functional capability of that device. It will execute all the 4004 instructions, and is also electrically compatible with the other members of the MCS-4 family (4001, 4002, 4003).

4040 CPU BLOCK DIAGRAM



4040 Pin Definition



D_0-D_3:	Bidirectional Data Bus. All address and data communication between the processor and the RAM and ROM chips occurs on these 4 lines.
STP:	'Stop' input. A logic "1" level on this input causes the processor to enter the Stop mode.
STPA:	'Stop Acknowledge' output. This signal is present when the processor is in the stopped state. Output is "open drain" requiring pull-down resistor to V_{DD}.
INT:	'Interrupt' input. A logic "1" level at this input causes the processor to enter the Interrupt mode.
INTA:	'Interrupt Acknowledge' output. This signal acknowledges receipt of an Interrupt signal and prevents additional Interrupts from entering the processor. It remains active until cleared by the execution of the new 'Branch Back' and SRC (BBS) instruction. The output is "open drain" requiring a pull-down resistor to V_{DD}.
RESET:	'Reset' input. A logic "1" level at this input clears all flag and status registers and forces the programme counter to zero. To completely clear all address and index registers, 'Reset' must be appl ied for 96 clock cycles (12 machine cycles).
TEST:	'Test' input. The logical state of this signal may be tested with the JCN instruction.
SYNC:	'Sync' output. Synchronization signal generated by the processor and sent to ROM and RAM chips. It indicates the beginning of an instruction cycle.

CM-RAM$_0$ - CM-RAM$_3$: CM-RAM outputs. These are bank selection signals for the 4002 RAM chips in the system.

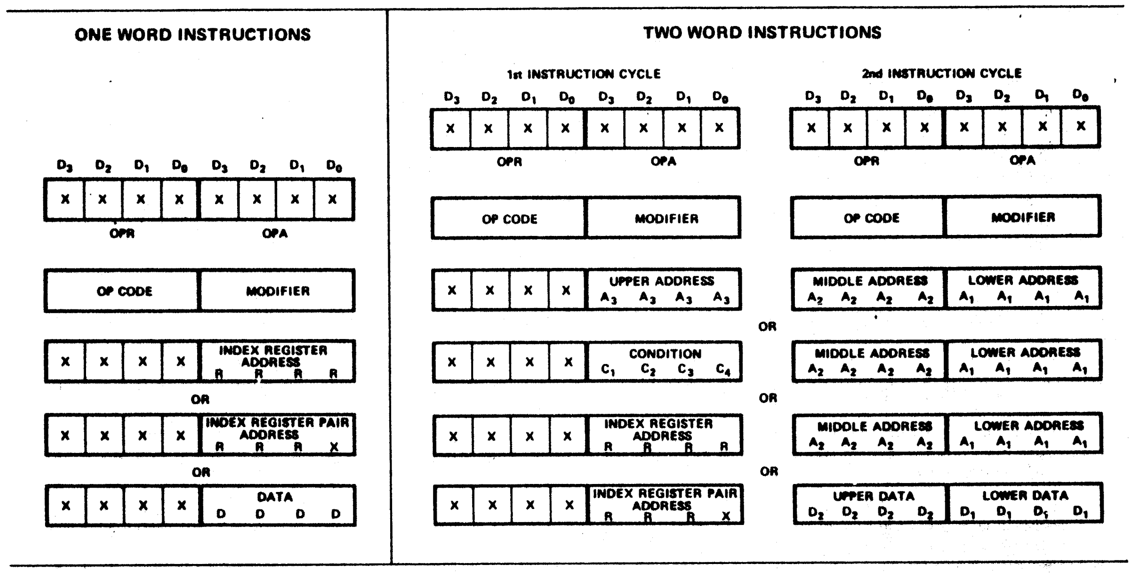
CM-ROM$_0$ - CM-ROM$_1$: CM-ROM outputs. These are bank selection signals for programme ROM chips in the system.

CY: 'Carry' output. The state of the carry flip-flop is present on this output and updated each X_1 time. Output is "open drain" requiring pull down resistor to V_{DD}.

Instruction Set Format

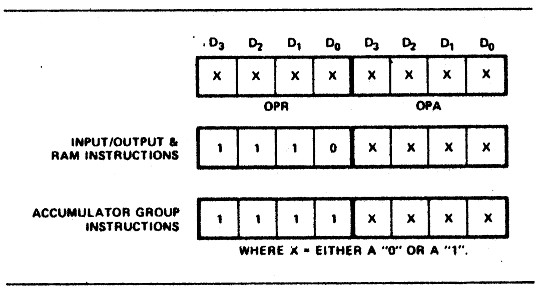
Each instruction is divided into two 4-bit fields. The upper 4-bits are the OPR field containing the operation code. The lower 4-bits are the OPA field containing the modifier. For two word instructions, the second word contains address information or data.

The upper 4-bits (OPR) will always be fetched before the lower 4-bits (OPA) during M_1 and M_2 times respectively.

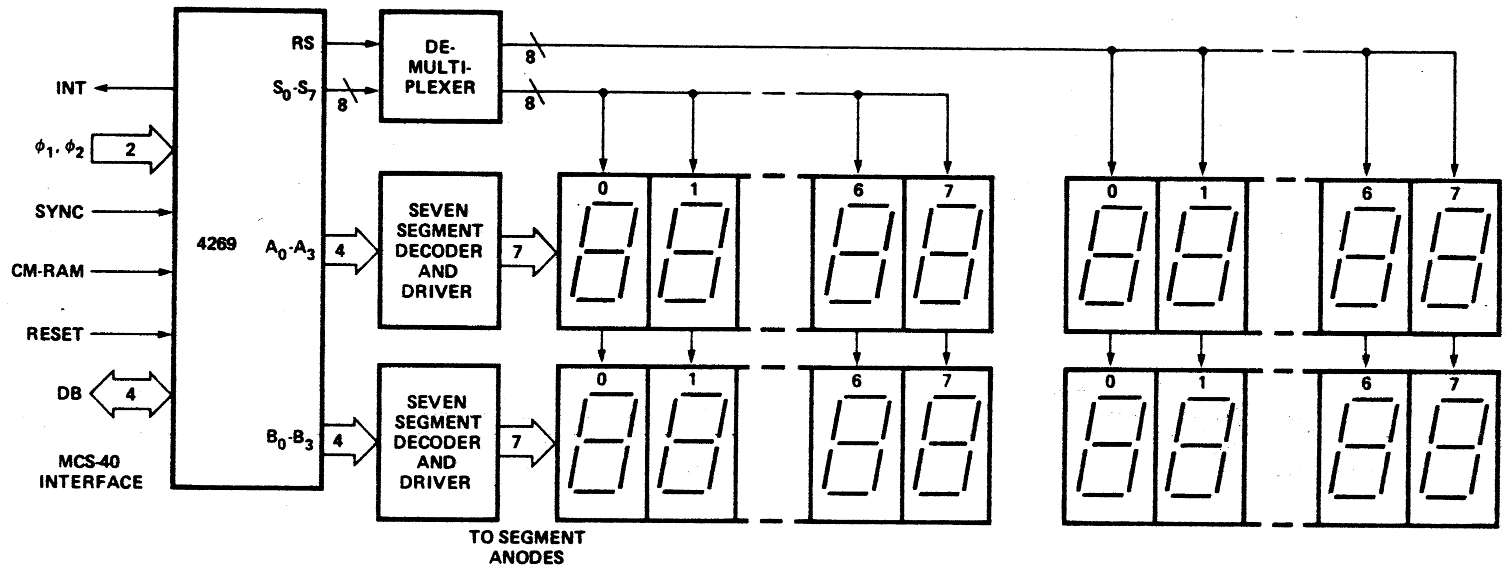


Input/Output and RAM Instructions and Accumulator Group Instructions

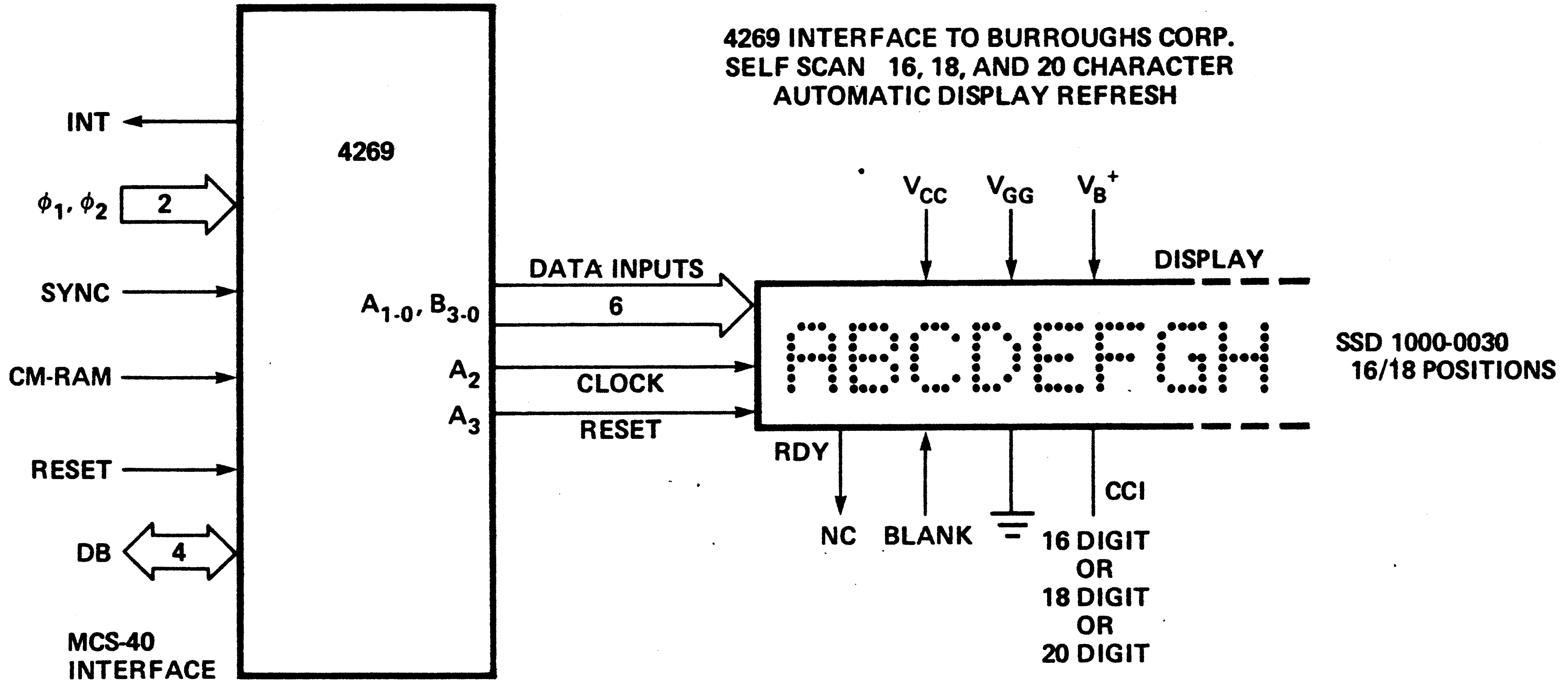
In these instructions (which are all single word) the OPR contains a 4-bit code which identifies either the I/0 instruction or the accumulator group instruction and the OPA contains a 4-bit code which identifies the operation to be performed. Table II illustrates the contents of each 4-bit field.


D3 D2 D1 D0 D3 D2 D1 D0
OPR
OPA
INPUT/OUTPUT & RAM INSTRUCTIONS
1 1 1 0 X X X X
ACCUMULATOR GROUP INSTRUCTIONS
1 1 1 1 X X X X
WHERE X = EITHER A "0" OR A "1".

4269 INTERFACE TO DUAL 16 X 4 NUMERIC SCANNED DISPLAYS



4269 INTERFACE TO BURROUGHS CORP. SELF SCAN 16, 18, AND 20 CHARACTER AUTOMATIC DISPLAY REFRESH


INT
ϕ_1, ϕ_2
2
SYNC
CM-RAM
RESET
DB
4
MCS-40 INTERFACE
4269
A_{1-0}, B_{3-0}
A_2
A_3
DATA INPUTS
6
CLOCK
RESET
V_{CC}
V_{GG}
V_B^+
DISPLAY
ABCDEFGH
RDY
NC
BLANK
CCI
16 DIGIT OR 18 DIGIT OR 20 DIGIT
SSD 1000-0030 16/18 POSITIONS