Using a microprocessor

2 — Hardware and programming

by J. Skinner, *Leafields Engineering Ltd*

At the end of the first part of the article, the flow chart had been derived. Consquently, the designer is now able to develop the programme and translate his thoughts into hardware.

Programming

In the completed programme, each instruction is denoted by a mnemonic and a binary machine-code word. The binary coding is used by the microprocessor and programmed instructions must end up in this form, but the procession of ones and zeros is not the easiest way to see what is happening. It is common, therefore, to use the mnemonic form of the instruction for juggling about with a programme and to convert it into machine code later, with the aid of the instruction-set table. Assembler programmes will, when run on a microprocessor, convert mnemonic codes into machine codes. The abbreviated instruction set for the 8080 is shown in Table I.

Points to bear in mind when tackling the programme include the way in which each instruction is handled by the c.p.u. Two or three bytes are needed to carry out each instruction and this fact must be taken into account to preserve the logical sequence. The programme is held in memory in a sequence in which the step number is the actual memory address, so that the order of addressing the memory by the c.p.u. is vital.

I/O. The simplest way of selecting the I/O block required for a particular function is to use binary code (1, 2, 4, 8, 16, etc.) which can be produced automatically by the c.p.u. This binary code can be read into the c.p.u. in the ordinary way as data and transferred to the address lines when needed. In this way, each address line calls up a separate I/O block, as in Fig. 4 of part 1.

Jump instructions. Instructions which call for the programme to jump consist of three bytes, the second and third of which are the least significant and most significant bits respectively of the address to which the programme is to jump.

Table 1. *Abbreviated instruction set for the 8080, showing only those instructions used in the programme discussed.*

Mnemonic	Machine code	Machine code (hex)	Function
MVI, A	00111110	3E	Load accumulator
OUT	11010011	D3	Output
E1	11111011	FB	Enable interrupt
HLT	01110110	76	Halt
MVI, D	00010110	16	Store in register D
MOVA, D	01111010	7A	Move data from register D to accumulator (A)
IN	11011011	DB	Input
ANI	11100110	E6	AND with data in accumulator immediately
FO	11110000	FO	Bits generated to perform AND function in text of article Not part of instruction set
RRC	00001111	OF	Shift accumulator right
CMPL	10111101	BD	Compare the content of L with content of accumulator
JM	11111010	FA	Jump if result of last operation is minus quantity
DCRD	00010101	15	Decrement or count down content of register D
JNC	11010010	D2	If the relevant "flag" is zero, jump (Jump on no carry)
DI	11110011	F3	Disable interrupt
JMP	11000011	C3	Jump to assigned address unconditionally

Register code	Register letter
000	B
001	C
010	D
011	E
100	H
101	L
110	Memory
111	Accumulator

Rotation. The data held in the accumulator can be shifted to the right or left. As it moves out of the register, the data will be lost unless it is fed back to the beginning, in which case eight shifts will return an 8-bit register to its ordinary state. This process is termed "rotation" for obvious reasons. A bit shifted out of the register can be tested for a value of 1 or 0 and a condition "flag" signal set or reset. For example, at address 52 (34 in hexadecimal or 00110100 in binary) the contents of the control valve register E have been transferred to the accumulator, rotated right and transferred back to E. If the flag bit is zero, the programme is to jump to the next control line address.

Initializing. It may be necessary, as in this programme, to see that the output ports are in the correct condition, since the reset function of the 8080 (wired) is only concerned with the programme counter; c.p.u. registers must be set to their initial conditions. Immediately on switching on, therefore, the accumulator and valve controls are set to zero. Since the programme has now started, it must be halted and an interrupt start signal awaited for the main part of the programme to continue.

Coding. It is common to translate the pure binary of the machine code into hexadecimal for ease of handling. The code is shown in Table 2 for those who are unfamiliar with it. For example,

Table 2. *Decimal, binary and hexadecimal equivalents.*

decimal	binary	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

using the eight-bit word of the 8080, the instruction to read in data is 'IN' (mnemonic), 11011011 (binary), DB (hexadecimal).

Programme. The final form of the programme is seen in Table 3, in which the hex. code is used for the programme address and machine-language instructions, for which mnemonics are also given. Incidentally, the division of the eight-bit machine code into two four-bit words, each being given a hex. code, does not mean that this is how the code is made up. In the MOV instructions, for example, the first two bits are always 01, followed by two, three-bit addresses for destination and source of the data to be moved. Register B has the code 000 and register D is coded 010; as in Table 2, so that the instruction "Move the contents of register B to register D" would be coded 01 010 000, which can be grouped 0101 0000, translating into hex. code as 50.

Use of r.a.m. Where the data storage provided in c.p.u. is not sufficient, extra capacity in the form of r.a.m. may be included, as shown in Fig. 1. The memory element is coupled to address data-bus lines in exactly the same way as the r.o.m. and I/O elements, but an additional control function has to be provided in order to distinguish between the r.o.m. amd r.a.m. elements in the read mode. Usually, there are spare address lines available and these can be used to control the memory elements via the chip-select (CS) function provided. Thus if A0-A7 are used for normal addressing for 8-bit, 256-word r.o.m. and r.a.m. A8 can be used to supply CS for r.a.m. For the r.o.m. it is necessary to invert A8 and gate with memory read ($\overline{MR}$). Instructions involving r.a.m. must then include an address code starting at 2`. A similar technique starting at higher addresses may be used where a larger r.o.m. is required. If insufficient address lines are available for this technique to be used, address decoding must be used, following the same general philosophy.

A technique known as "memory mapping" is described in the INTEL users manual. This technique treats the I/O elements as part of the memory array, selection being via the appropriate address code. This has the advantage of allowing direct transfer of data between I/O and registers of memory, without data having to be routed through the accumulator.

Table 3. *Complete programme*

Address (Hx)	Mach. Code (Hx)	Mnemonic	Function
0	3E	MVI. A	Set accum
1	00	0	=0
2	D3	OUT	Output '0' to valve controls
3	08	8	(I/O block address=8)
4	FB	EI	Enable interrupt
5	76	HLT	Halt (and await interrupt start signal)
6	D3	OUT	Output '0' to card select column and complete flag
7	10	16	(I/O block address=16)
8	16	MVI. D	Store number of card columns to be read in register 'D'
9	07	7	(0 to 7)
A	1E	MVI. E	Store number of valves to be processed in register E
B	80	80H	(0 to 7 in binary)
C	7A	MOVA. D	transfer from register D to select
D	D3	OUT	next card column
E	10	16	I/O address
F	DB	IN	Fetch card m s.b. data from
10	01	1	I/O address 1
11	67	MOVH. A	Store card m.s.b. data in register 'H'
12	DB	IN	Fetch card l.s.b. data and d.v.m. l.s.b. data
13	02	2	from I/O address 2
14	E6	ANI	Blank off d.v.m. l.s.b. (This is the AND function
15	F0	F0	referred to in part 1 of the article.)
16	0F	RRC	
17	0F	RRC	
18	0F	RRC	Shift right 4 times
19	0F	RRC	
1A	6F	MOVL. A	Store card l.s b. data in register 'L'
1B	7B	MOVA. E	Transfer data from register E to
1C	D3	OUT	select next valve
1D	08	8	I/O address
1E	DB	IN	Fetch card l.s.b. and d.v.m. l.s.b. data
1F	02	2	from I/O address 2
20	E6	ANI	Blank off card l s b (The AND function)
21	0F	0F	
22	BD	CMPL	Subtract card l s.b. from d v.m l s.b
23	FA	JM	Return to Fetch if result negative
24	1E		l s.b jump address
25	00		m s.b. jump address
26	DB	IN	Fetch d.v.m. m.s.b. data
27	04	4	I/O address 4
28	BC	CMPH	Subtract card m s b from d v m m.s b
29	FA	JM	Return to fetch if result negative
2A	26		l s b. jump address
2B	00		m.s.b. jump address
2C	3E	MVI. A	Set accum. to
2D	00	0	=`0'
2E	D3	OUT	Output '0' to control valves
2F	08	8	I/O address
30	15	DCRD	Count down card column select register
31	7B	MVA. E	
32	0F	RRC	Count down control valve select register
33	5F	MOVE. A	
34	02	JNC	If flag is zero, return and select next
35	7A	C	control line l s b. jump address
36	00		m s b jump address
37	3E	MVI A	
38	08	8H	Output signal to 'complete flag
39	D3	OUT	
3A	10	16	
3B	F3	DI	Disable interrupt
3C	C8	JMP	Return to start
3D	00		l s.b. jump address
3E	00		m s.b jump address

Hardware.
The complete system, used for developing and proving the programme described above, is shown in Fig. 1, with a glossary in Table 4. Although r.a.m. was not required for this application, it has been included so as to be available for future use. This configuration will, we hope, prove to be universal. There are several proprietary m.p.u. systems now available in p.c. form, although none has yet been seen with the I/O structure as described in this article, most of the products being best suited to data-transmission applications. It is appreciated that most of the interface elements, such as the universal, asynchronous, receiver-transmitter (u.a.r.t.) and programmable peripheral interface (p.p.i.) could be used in the system of Fig. 1; but they are unnessarily complicated and more expensive that the simple device described (actually, little more than an 8-bit latch). Most of the system components have already been described but some additional comments may be helpful.

System control. This is a single element provided by Intel for decoding and synchronizing the control bus. A bi-directional data bus driver is included, as is isolation of memory and I/O controls.

I/O. The Intel 8212 element is used, as mentioned above, for sheer simplicity. It is basically an 8-bit latch with 3-state output for bus operation. A mode control enables either input or output function to be selected. In the system of Fig. 1, this is determined by a wired link, but could also be programmed by the c.p.u. Interrupt and clear facilities are provided, these not being required in this application.

R.a.m. 8 bits × 256 words of storage are provided in the form of 2, 4-bit, 256-word elements. The two sets of four data bits appear side by side to form the 8-bit data word. Addresses are common to both elements. Gating for r.a.m./r.o.m. selection is provided by a single 7400.

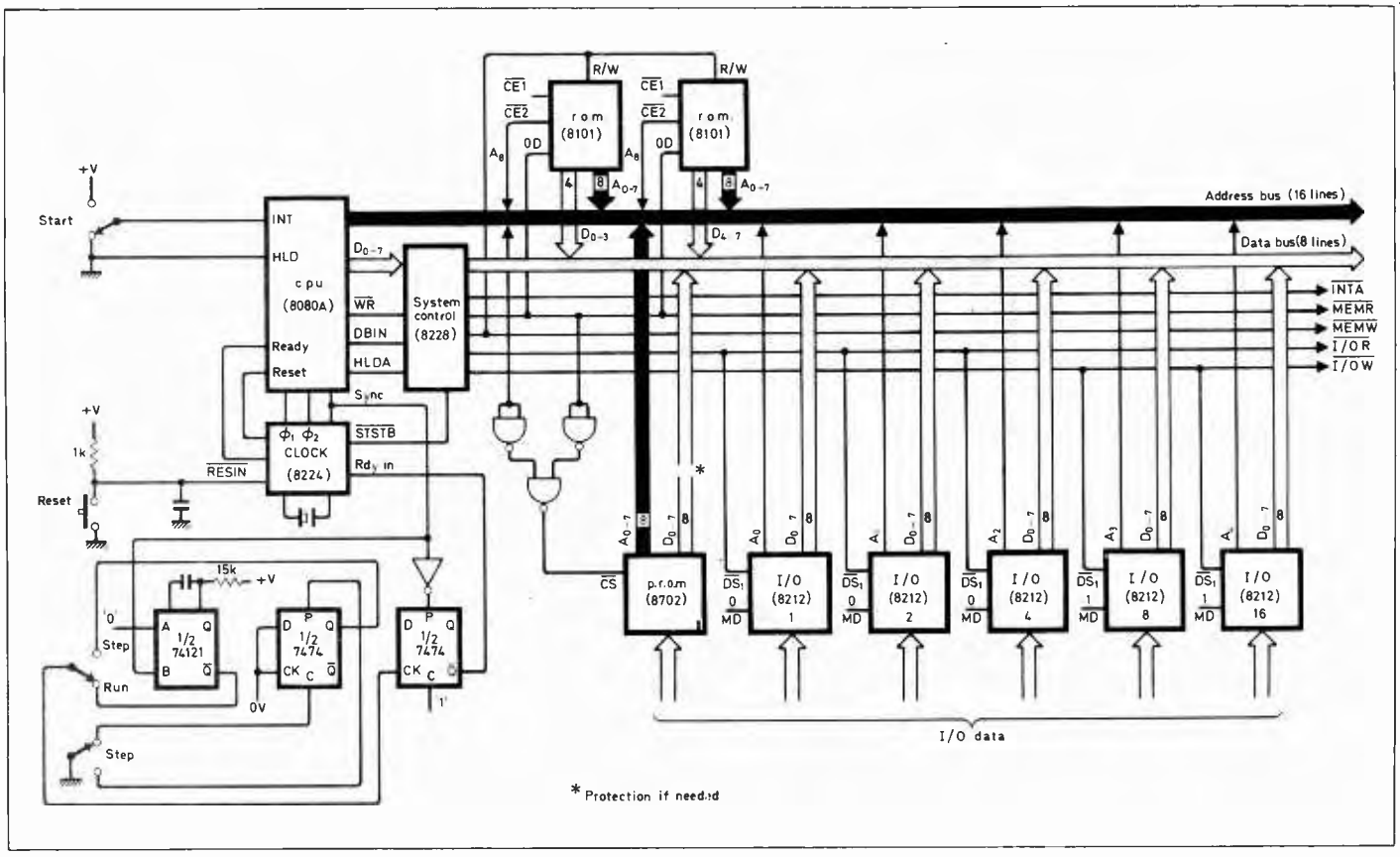

Fig. 1. The complete circuit of a universal microprocessor. The three modules at the lower left form the 850ns memory access (right and left i.cs) and a single-step function (centre and right i.cs).

P.r.o.m. An 8 × 256-bit p.r.o.m. is shown, whose size can easily be increased, since there are spare address lines available. A r.a.m. was used for this function during development, a plug-in version simulating the 8702 p.r.o.m. being purchased. This could be constructed very easily and cheaply but, since we were more interested in developing the m.p.u. technique than developing a r.o.m. simulator, we decided to buy one. The simulator is provided with hex. coded programme and address thumbwheels and binary display of the data which, apart from its usefulness for programming, we found useful during programme check out.

R.a.m. and r.o.m. speed. The 8080 c.p.u. is designed to operate with memory components having an access time of approximately 450-550ns, although times of up to 850ns are suggested as being suitable. Cost is, of course, related to speed and many users will wish to use the slower devices — the 8702 for example has a maximum access time of 1.3μs. Provision for slower devices can be made by controlling the "ready" input to the c.p.u. (the clock controller in this example). One or more clock periods are used to provide a "wait" state suited to the access time of the memory system used. The two functions of 850ns memory access and single-step drive are incorporated in the complete system of Fig. 1.

De-bugging. Faults are of two kinds — hardware and software. Monitoring the data lines enables the programme sequence to be verified, and address-line monitoring can also be useful, while buffered l.e.ds plugged into a spare socket or even wired in permanently will prove invaluable even to the experienced. Checking correct operation of all components, with the exception of the c.p.u. is straightforward. The c.p.u. can prove difficult to test because of its high operation speed and also because of its complexity. Fault finding equipment is costly and substitution is the usual way out.

P.r.o.m. protection. Intel mention in their Memory Design Handbook the need to protect p-type p.r.o.m. data inputs from the negative levels produced on the data bus by an n-type r.a.m. The 8702 p.r.o.m. is a p-type and the 8101 r.a.m. is an n-type so that protection should be provided in order to avoid damaging the p.r.o.m. All that is required is the inclusion of a series limiting resistor of 250Ω and shunt diode, in each of the p.r.o.m. data input lines.

Conclusion

This is a system which has been tried and proved. The programme may be used to prove hardware. It is hoped that the stages in development of both hardware and software have been dealt with in sufficient detail for constructors to proceed with their own designs. Neither the hardware nor software is considered to be unique but it is hoped that it will prove to be applicable to many future problems.

The author gratefully acknowledges the assistance of Howard Kornstein of Intel and the staff of Rapid Recall Ltd., in developing the system. Thanks are also due to K. Sharman who constructed and tested the system and also developed the single stepping facility.

Table 4. *Abbreviations used in system diagram.*

$\overline{CE1}$	Chip enable
$\overline{CE2}$	
R/W	Read/write input
OD	Output disable
INT	Interrupt request
$\overline{INTA}$	Interrupt acknowledge
$\overline{HLD}$	Hold
$\overline{WR}$	Write output
DBIN	Data bus in Signal to system controller that data bus is in input mode
HLDA	Hold acknowledge Signal in response to hold signal
$\overline{STSTB}$	Status strobe
$\overline{CS}$	Chip select input
$\overline{DSI}$	Device select input
MD	Mode
$\overline{MEMR}$	Memory read
$\overline{MEMW}$	Memory write
$\overline{I/OR}$	I/O read
$\overline{I/OW}$	I/O write

Negated names indicate that the function is active when the signal is low