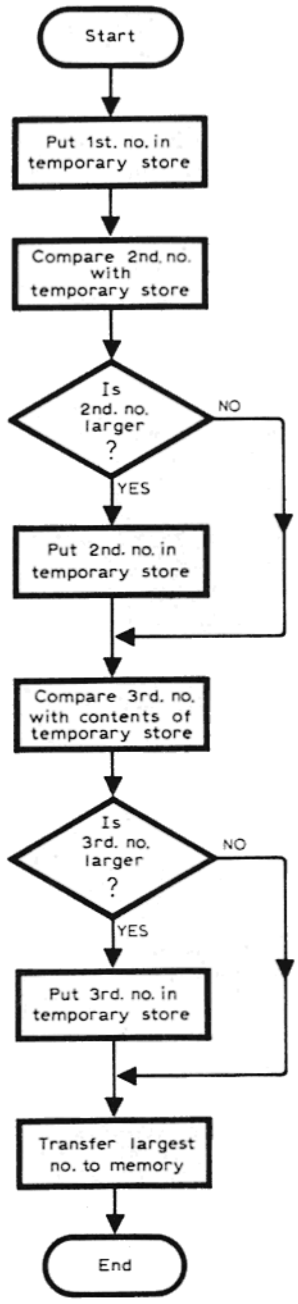
Microcomputer design — 5

Introduction to microcomputer programming

by Phil Pittman B.Sc in association with NASCO Ltd

A microcomputer is capable of storing information, controlling other devices, performing calculations, making decisions based on the results and completing a given task very rapidly. The processor cannot, however, perform these tasks without direction. Each step which the computer is to perform must first be worked out by the programmer.


Start
Put 1st. no. in temporary store
Compare 2nd. no. with temporary store
Is 2nd. no. larger ?
NO
YES
Put 2nd. no. in temporary store
Compare 3rd. no. with contents of temporary store
Is 3rd. no. larger ?
NO
YES
Put 3rd. no. in temporary store
Transfer largest no. to memory
End

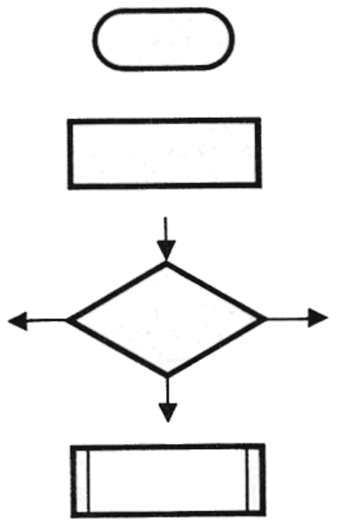
AS explained in previous articles a programme is a list of instructions for the computer to follow in order to execute a given task. When a complex task has to be performed the programme may involve many steps, and writing it often becomes long and confusing. A method for solving a problem which is written in words, and possibly mathematical equations, is extremely difficult to follow, and compiling computer instructions from such a document would be equally difficult.

A technique called "flowcharting" is used to simplify the writing of programmes. A flowchart is a graphical representation of a given problem, indicating the logical sequence of operations that the computer is to perform. Having a diagram of the logical flow of a programme is a tremendous advantage to the programmer when he is determining the method to be used for solving a problem, as well as when writing the coded programme instructions. In addition, the flowchart is often a valuable aid when the programme checks the written programme for errors.

Fig. 1 is a flowchart which shows the sequence of operations for a programme which selects the largest of three numbers. The assumption is that the numbers are stored in consecutive memory locations and that the selected largest number is to be stored in the fourth consecutive location. To help with Fig. 1, Fig. 2 shows some common symbols used in drawing flowcharts. The rectangle represents an operation to be performed within the programme. The diamond shape is used to indicate a decision point where one of two or more paths is selected by the programme. There are other symbols for various other functions, but those shown are the most frequently used ones.

The flowchart of Fig. 1 clearly illustrates the method for selecting the largest of three numbers. Essentially, adjacent numbers are compared and the larger at each comparison is saved and used as one of the numbers for the next comparison. At the end of the sequence of comparisons the last "saved" value will be the largest from the group. By repeating the process the method may be extended to any number of values. For more complex problems the initial flowchart may not give as much detail about the operation of the programme as is shown in Fig. 1. For example, the task of selecting the largest of a group of numbers may be only a small part of a much larger item of software. Conse-

Fig. 1. *This flowchart is a preliminary to writing a programme for selecting the largest of three numbers. The programme itself is shown in Fig. 5.*



Represents the start, the end, or an interruption of the programme depending on the word contained in the box.

Represents a given task accomplished by the programme, the description of the task being briefly indicated inside the rectangle.

Indicates that a test must be made to determine the subsequent path taken by the programme. The test is specified within the diamond and its results marked above the appropriate output paths.

Represents one or several operations which are not detailed on the flowchart in question but are detailed on another flowchart. A sub-programme is often represented in this way.

Fig. 2. *Common symbols used in flowcharts.*

quently, on a different flowchart this complete operation may be represented by a single box, as indicated in Fig. 3.

The very first flowchart for the operation of a microcomputer project may contain very little detail of the actual method by which the central processing unit will solve the problem. However, each block must then be broken down into smaller and smaller operations, probably resulting in several "levels" of flowchart, depending on the complexity of the problem, being generated along the way. Finally, as with the Fig. 1 example, the flowcharts will contain sufficient detail to be translated directly to machine instructions. By adopting this method of generating various levels of flowcharts, a more orderly solution to the software problem will result.

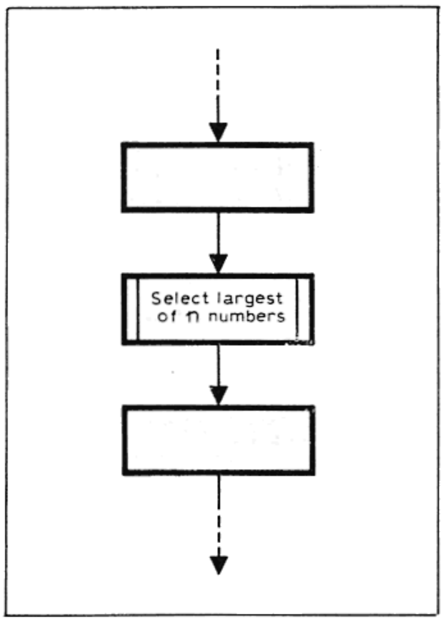


Fig. 3. *Example of a flowchart referring to a sub-programme.*

Programming models and instruction types

Flowcharts are generally "machine independent" in that identical flowcharts can be used as a basis for generating programmes for virtually any computer. However, in order to translate a flowchart into a programme for a particular machine, the programmer must be completely familiar with the instruction set of the c.p.u. (see December 1977 issue, p.56 and p.59) and know which registers within the c.p.u. are accessible by these instructions.

Fig. 4 shows what is called the programming model or internal register organization of the Z80 microprocessor chip. Before proceeding with a programming example it is necessary to study these aspects of the c.p.u. Note that some of the registers are duplicated in the Z80 and are referred to as the main and alternate registers sets. Within the Z80 there is a means for selecting one or other set for current working. The current discussion will be limited to considering the main set only, plus some of the other special-purpose registers. Each register has a particular significance in the overall operation of the c.p.u.

There is a register known as the accumulator. This 8-bit register, which is denoted by the letter A, is always used for one of the operands in any 8-bit arithmetic or logical operation, and as such is a very special and important register of the c.p.u. For example, if two 8-bit numbers are added, subtracted, compared, etc., one of them must reside in the accumulator and this is also where the result of the operation is left. Registers called B, C, D, E, H, L are general purpose 8-bit registers which may be used as stores in a similar way to any external memory locations. However, being part of the c.p.u. means that they may be accessed faster and more easily than external memory. In addition to being general purpose stores, registers B and C, D and E, and H and L may be used in pairs to form 16-bit registers for many types of arithmetic operations. Also, these 16-bit registers may be used to hold memory addresses for certain memory reference operations. This is particularly true of the H and L pair, which may be used to contain an address for many register-to-memory and memory-to-register data transfers, arithmetic and logical operations.

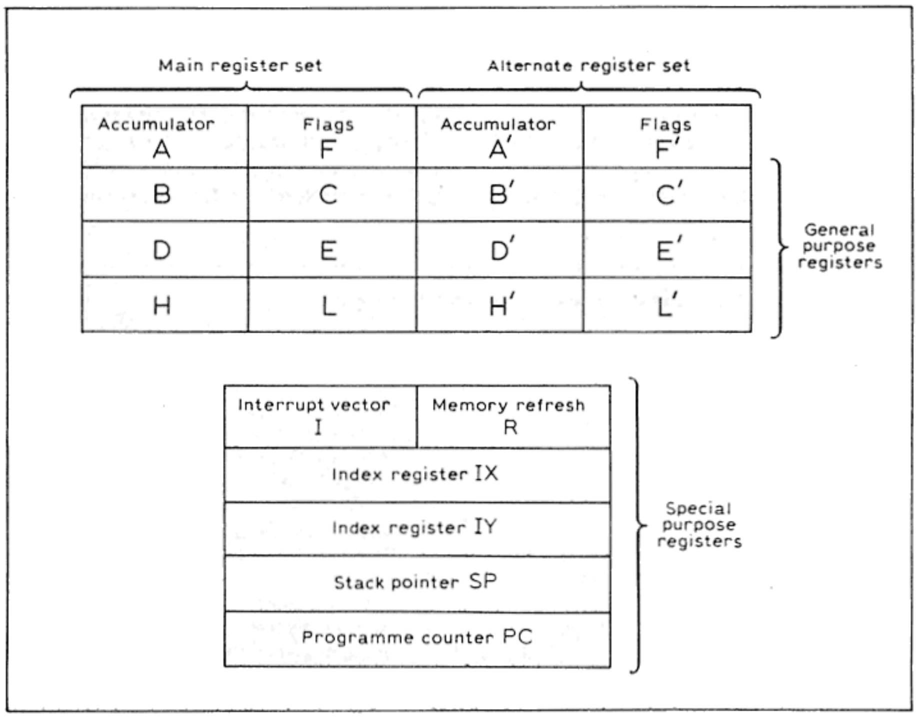


Fig. 4. *Diagram showing the organization of the internal registers of the Z80 microprocessor, known as a "programming model".*

Register F in Fig. 4 is not really a register in the normal sense but is the collection of c.p.u. status bits which are affected by the a.l.u. operations and which may be tested by the conditional jump instructions.

Registers IX and IY are 16-bit registers used primarily for holding memory addresses for special "indexed" addressing operations. Arithmetic operations may also be performed, using these registers.

The SP or "stack pointer" register is another special purpose address register whose function will be explained in a later article. Register PC is the 16-bit programme counter which keeps track of the current instruction address in the programme memory.

Registers I and R have special functions which will also be explained in a later article.

Instructions which operate on data within the above registers or memory locations may be classified into various groups. Any computer will have similar instruction groups, although the actual instructions within these groups are likely to differ between different designs of c.p.u.

The instruction set of the Z80 consists of 158 different instructions, which may be broken down into the following major groups.

Load and exchange
Block transfer and search
Arithmetic and logical
Shift and rotate
Bit manipulation
Jump, call and return
Input/output
Basic c.p.u. control

The load instructions move data internally between c.p.u. registers or between c.p.u. registers and external memory. All of these instructions must specify a source location from which the data is to be moved and a destination location. The exchange instructions can swap the contents of certain c.p.u. registers.

A unique set of block transfer instructions is provided in the Z80. With a single instruction a block of memory data of any size can be moved to any other area of memory. These instructions are extremely valuable when large strings of data must be processed. The block search instructions are also valuable for this type of processing. With a single instruction a block of external memory of any desired length can be searched for any 8-bit character. When the character is found, the instruction automatically terminates.

The arithmetic and logical instructions operate on data stored in the accumulator and other general purpose c.p.u registers or memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. This group also includes various 16-bit arithmetic facilities.

The shift and rotate instructions allow data in the accumulator or other 8-bit registers to be shifted or rotated in various ways, often including the carry flag as a ninth bit.

Bit manipulation instructions allow any bit in the accumulator, any general purpose register or any external memory location to be set, reset or tested with a single instruction. This group is especially useful in control applications and for controlling "software flags" in general purpose programming.

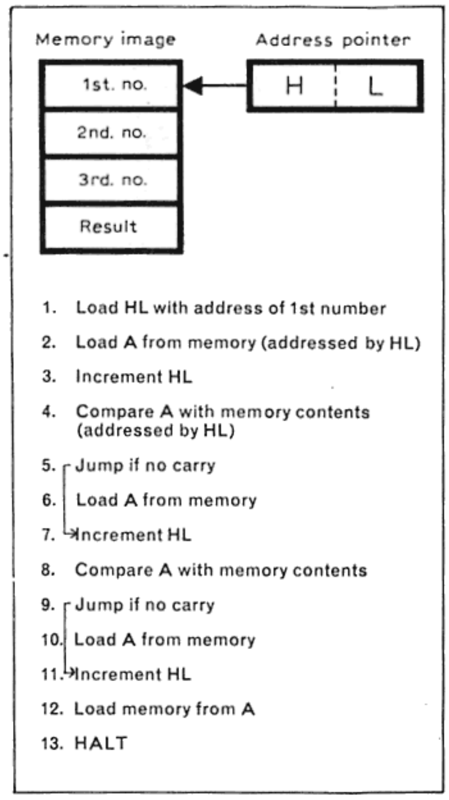
The jump, call and return instructions are used to transfer information between various locations in the user's programme. This group uses several different techniques for obtaining the new programme counter address from specific external memory locations. Programme jumps may also be achieved by loading the contents of registers H and L, IX or IY directly into the programme counter, thus allowing the jump address to be a complex function of the programme being executed.

The input/output group of instructions allow for a wide range of transfers between external memory locations or the general purpose c.p.u. registers and the external i/o devices.

Finally, the basic c.p.u. control instructions allow various options and modes including instructions for effecting the interrupt response.

Coding the programme

With the instruction set at his disposal, the programmer can begin to translate the detailed flowcharts into actual machine instructions. In the case of the "selection of largest number" programme, the flowchart of Fig. 1 is sufficiently detailed to give approximately a one-to-one correspondence between a flowchart block and a c.p.u. instruction. This will not always be true as problems get more complex and as the programmer becomes more proficient and confident. In Fig. 5 the flowchart has been translated into a list of programme instructions.



1. Load HL with address of 1st number
2. Load A from memory (addressed by HL)
3. Increment HL
4. Compare A with memory contents (addressed by HL)
5. Jump if no carry
6. Load A from memory
7. Increment HL
8. Compare A with memory contents
9. Jump if no carry
10. Load A from memory
11. Increment HL
12. Load memory from A
13. HALT

Fig. 5. *Actual programme to select the largest of three numbers.*

Notice that in order to address the sequential data memory locations it is convenient to use the 16-bit pair of registers H and L of the c.p.u. This is set up at the start of the programme (line 1 in Fig. 5) to contain the memory address of the first number in the data list. Consequently, as the other numbers have to be accessed the HL address pointer, as it is called, may be advanced by one each time with a suitable instruction (lines 3, 7 and 11 in Fig. 5). An alternative method would be to include the absolute address of each number as part of a suitable instruction at the relevant parts of the programme. However, in this case it would have resulted in a more inflexible programme and would require additional memory locations for the 16-bit address values to be stored in the programme.

The temporary store referred to in the flowchart has been chosen to be the accumulator register of the c.p.u. This is because in order to compare two numbers, one of them has to reside in this register. Therefore, unnecessary data movements can be avoided if this is used as the temporary store in this case.

Once the data address is set in the HL register, there are instructions available for loading the accumulator from memory address by HL (line 2) and vice versa (line 12). Also, other operations such as "compare A with memory" use the contents of HL as a memory address (line 4 and 8). Remember that the compare instructions work in a similar way to a subtraction, in that if the memory content is larger than the accumulator content then a carry (or borrow) will be generated, thereby setting the carry flag. The "jump if no carry" instructions will test this flag and decide whether the accumulator contains the larger value or whether it needs to be loaded into the accumulator from the currently addressed memory location. Consequently, before the execution of the instruction at line 7 or line 11, the accumulator will contain the largest value so far.

Instruction mnemonics

Programmes are rarely written as shown in Fig. 5 since it becomes very tedious to write out all the instructions in this form. The c.p.u. instructions are commonly abbreviated as shown in Table 1. Mnemonics are used for the various types of instruction, e.g. load — LD, compare — CP, jump — JP, increment — INC, etc. Also, the operands for the instructions are specified by suitable abbreviations for the storage locations in which they are held, for example A, HL, (HL).

These mnemonics are collectively known as the programming language. In particular they are the assembly language of the Z80 c.p.u. Some computer systems have an "assembler" which is a special programme for automatically translating the assembly language mnemonics into binary machine code.

Note that the parenthesis in the case of (HL) means that the operand is not actually the contents of the HL register but the contents of memory "addressed by" the contents of HL. Note also that

Table 1 — Mnemonic coded programme with comments

Line No.	Instruction	Comment
1	LD HL, 0900	Set data address in HL
2	LD A, (HL)	Get 1st data byte into accumulator
3	INC HL	Update address pointer
4	CP (HL)	Compare 2nd data with 1st
5	JP NC, LINE 7	Test carry flag
6	LD A, (HL)	Carry, so put larger in accumulator
7	INC HL	Update address pointer
8	CP (HL)	Compare 3rd data with accumulator
9	JP NC, LINE 11	Test carry
10	LD A, (HL)	Put larger in accumulator
11	INC HL	Update pointer
12	LD (HL), A	Store largest in memory
13	HALT	Halt c.p.u. at end of programme

Table 2 — Machine coded programme listing

Line No.	Address	Machine	Instruction mnemonic
1	0800	21 00 09	LD HL, 0900H
2	0803	7E	LD A, (HL)
3	0804	23	INC HL
4	0805	BE	CP (HL)
5	0806	D2 0A 08	JP NC, LINE 7
6	0809	7E	LD A, (HL)
7	080A	23	INC HL
8	080B	BE	CP (HL)
9	080C	D2 10 08	JP NC, LINE 11
10	080F	7E	LD A, (HL)
11	0810	23	INC HL
12	0811	77	LD (HL), A
13	0812	76	HALT

for the jump instructions the addresses of the instructions at lines 7 and 11 would, in practice, need to be inserted as part of the jump instruction.

In order to make the programme more readable it is good practice to include comments as shown in Table 1, indicating how the programme operations relate to the task in hand.

Finally, before the programme can be entered into the computer's memory the instructions must be converted into the appropriate binary codes. This is done by referring to the c.p.u.'s instruction set details. At this stage it is also necessary to allocate memory addresses both for the programme and also for any data which must reside in memory.

The r.a.m. of the microcomputer kit (November 1977 issue, p. 45) starts at address 800 (hex), this being the beginning of the third 1K address block (December 1977 issue, Fig. 4). Consequently this would be a suitable address at which to store the programme. Four data memory locations are also required and so addresses 900 to 903 (hex) could arbitrarily be chosen for these.

We are now in a position to generate the machine code programme. Table 2 shows the resulting programme, indicating the relevant memory addresses for the instructions. Note that the hexadecimal number system has been used throughout. Where an instruction requires more than one memory location all the bytes of information have been shown on one line and the memory address of the next line is adjusted accordingly. The Z80 c.p.u. requires that whenever a 16-bit address is specified as part of an instruction, the least significant byte must be placed first in the memory, followed by the most significant byte.

Initially it is not possible to fill in the jump addresses at lines 5 and 9 until the memory addresses of the jump destinations (lines 7 and 11) have been established. So, on the first pass through, memory locations must be reserved for these values. Having established the memory locations required, one can then fill in the remaining memory references. For example, the jump instruction at line 5 must contain the address of line 7. Consequently the value 080A must reside in memory locations 0807 and 0808.

Running the programme

The following paragraphs illustrate how the above programme may be verified by running it on the microcomputer kit. A typical operational sequence is given, starting with the entry of the programme into the computer's memory, continuing with executing and verifying, and finally making a permanent record of the programme on cassette tape. In the discussion which follows the display listing produced by the kit is given. Those parts shown in bold characters are those which are typed by the user. The remainder is generated by the system. The reader should refer to Part 2 of this series (December 1977 issue) for a description of the system commands.

The first step is to type the programme into the memory using the M command. Each byte of machine code is entered starting from address 800. The M command responds with the current memory contents. The user must then type a space followed by the new value required. A carriage return then gives the contents of the next memory location on a new line and so on as shown below.

```
• M 800
0800  00  21
0801  00  00
0802  00  09
0803  00  7E
0804  00  23
0805  00  BE
0806  00  D2
0807  00  0A
0808  00  08
0809  00  7E
080A  00  23
080B  00  BE
080C  00  D2
080D  00  10
080E  00  08
080F  00  7E
0810  00  23
0811  00  77
0812  00  76.
•
```

The programme memory may be checked by using the "tabulate" command:

```
• T 800 812
0800 21 00 09 7E 23  BE D2 0A
0808 08 7E 23 BE  D2  10  08 7E
0810 23 77 76
•
```

Three data values must be entered in addresses 900-902. These can be any convenient 8-bit numbers and may be entered with the M command:

```
• M 900
0900  00  12
0901  00  34
0902  00  0B•
```

Everything is now ready for the programme to be run. However, it is rarely advisable to try to run the whole of a new programme without any intervention by the operator at any point since even the simplest programme is likely to contain errors initially. It is therefore desirable to set a breakpoint at a convenient place, after a few instructions will have been executed. A suitable point is at line 7 (address 080A). A breakpoint here will cause the programme to stop before the INC HL instruction is executed. At his stage of the programme the accumulator should contain the larger of the first two numbers..The following print-out shows the setting of the breakpoint, the start of programme execution and the display of programme counter and accumulator contents when the breakpoint is reached.

```
• B 80A
• E 800
080A 34
•
```

See that the programme has stopped with the programme counter at address 80A as specified by the breakpoint instruction. This confirms that at least some of the programme has indeed been executed. The accumulator appears to have the value 34 which is the larger of the first two numbers. However, in order to check that the programme branches correctly it should be tried again with numbers of different relative magnitudes.

To check the next part of the programme a breakpoint could be set at address 812. This will ensure that the final c.p.u. register states will be preserved for examination if required. The following sequence sets the new breakpoint, continues programme execution from the previous breakpoint, and displays the final programme counter and accumulator contents.

```
• B 812
• E
08012 34
•
```

Continued on page 88

Continued from page 80

The accumulator contains the value 34, which suggests that the programme is working correctly since this was the largest of the three numbers.

To finally verify the correct operation, memory address 903 should be examined.

```
• M 903
0903 34 •
•
```

If for any reason the programme had required changing, this could be accomplished with the "modify" command again. For example, if we wish to select the smallest rather than the largest of the numbers, this can be achieved by changing the "jump if no carry" instruction to a "jump if carry." This involves changing the JP NC, . . . (op code D2) to JR C, . . . (op code DA), e.g.

```
• M 806
0806 D2 DA •
• M 80C
080C D2 DA •
•
```

The programme could now be executed again in a similar manner to that shown above.

Finally, to keep a permanent record of the programme it can be saved on tape by the "dump" command. This also produces a display of the saved information.

```
• D 800 812
0800 21 00 09 7E  23 BE DA 0A
0808 08 7E 23 BE DA 10  08  7E
0810 23 77 76
•
```

At a later time the programme may be quickly re-loaded into the memory with the "load" command.

```
• L
•
```

These examples have illustrated some of the fundamental principles and implications of writing programmes for a microcomputer or any other computer system. However, much more detail than can be given here is required in order to get a greater appreciation of the programming facilities offered by the c.p.u. and the techniques for exploiting these facilities. Future articles will go some way to explain these very important aspects of microprocessor system design.

Reference

The Z80 c.p.u. Technical Manual ☐

Owing to production difficulties the remainder of Dr Shelton's articles on microcomputer hardware have had to be postponed, but will be resumed as soon as possible.

All Finniston's persons

THE first meeting of the committee of inquiry into the engineering profession, chaired by Sir Monty Finniston, took place on December 20 at Great Smith Street, London. The names of its members, announced a few days before, were: Catherine Avent, careers guidance ILEA; W. Buckley, Warrington technical college; T. Crispin, T & GWU; H. Darnell, British Steel; J. Dawes, ex-Rolls Royce; J. Dickinson, North Staffs polytechnic; J. Horlock, Salford University; W. Howie, *New Civil Engineer;* B. Lindley, ERA; H. Macdonald Smith, Army; W. McCall, Institution of Professional Civil Servants; J. Menter, London University; H. Nelson, Ransome Hoffman Pollard; J. Powell, EMI; E. Sadler, Ove Arup Partnership; D. Weir, Scottish Business School; J. Wilson, Tayside Region.

The secretary to the committee is Mr M. V. Boxall, who will accept submissions at Abell House, John Islip Street, London SW1.

Dr Powell's career has led him to the Clarendon Laboratories, Oxford University, Ottowa's National Research Laboratory, Marconi, where he worked on semiconductors, and Texas Instruments, where he moved from engineering to management. He joined EMI in 1974.

An article in the journal of the Institution of Production Engineers points out that, surprisingly, none of their members is represented on the committee. "Apparently eschewing the talents of MIProdEs, the committee includes seven educationalists, a magazine publisher, a civil servant, a trade unionist and four industrialists." The list does, indeed, have the look of a fairly typical selection from the Book of the Great and the Good, and one would have thought the civil service is going to have quite enough influence on the committee's work without putting one of its members on the committee as well. ☐

Defence research spawns commercial success

THE 1977 MACROBERT award has gone to a team of five who developed a device, the Malvern correlator, which uses lasers to measure flow rates. The range of applications is said to extend from the flow of blood through the blood vessels at the back of the eye, the only non-invasive method of doing this, to the rate of flow of gases through an engine.

Four of the winners come from the Physics group of the Royal Signals and Research Establishment, Malvern, and the fourth is the managing director of the firm which produced a commercial version of the device, Malvern Instruments Ltd.

The instigator of the project was Dr Roy Pike, one of the RSRE team, who was engaged in a study of the structure of light. In particular, they wanted to study laser light. The laser had only just been invented and few uses had been found for it. They reasoned that once they understood the nature of what came out of the laser it might be put to practical, probably defence, use.

They began to concentrate on the measurement of the characteristics of laser photons. The flow measurement technique stems from that. A laser beam is split into two beams, which converge in the centre of the flow. The optical fringes formed by the interference of the two beams, when observed at the other side of the flow, are disturbed by the flow particles. This disturbance, or scattering, is caused by the photon pulses bunching together as the particles move through a light area of the optical fringes. The intensity distribution of the fringes therefore gives a guide to the particle velocity. The periodicity of the pulse train is measured by auto-correlation technique — multiplying the pulse train by many time-delayed versions of itself.

One of the team, Mr D. S. Trudgill, left RSRE in 1971 and with help from the NRDC, set about making a commercial version of the equipment. The firm of which he is now managing director, Malvern, started selling them in 1972 and last year won a Queen's Award. The firm has 35 employees compared with the six it had when it began. They have sold over 200 Malvern correlators.

The MacRobert award is the most prestigious in UK engineering. It is worth £25,000 and a day at Buckingham Palace, where Prince Philip presented the awards at a private ceremony just before Christmas.

The chairman of the CEI, which sponsors it, Sir Charles Pringle, reminded those who gathered after the presentation that no award had been made last year for lack of entries of a high enough standard. This year, however, there had been a number which would have been eligible, and the problem this year had been one of selection. Perhaps the absence of an award last year had given the MacRobert prize a shot in the arm. ☐

IN BRIEF

Marconi are to supply tv signal monitoring equipment for the studios broadcasting the 1980 Moscow Olympic Games.

Ferranti have bought linear i.c. makers Interdesign of California.

EMI have a new company, EMI Industrial Electronics, to co-ordinate the £50 million worth of business they conduct in that area.

Voice of America have installed a short wave dipole curtain aerial array at their Delano, California, relay station. The aerial operates at 250 to 500kW with 100% modulation on 49m, 40m and 31m. It was supplied by TCI. Satisfactory signals have been received in the Philippines.

The Ministry of Defence have bought 400 u.h.f. radio relays from Marconi, nearly four years after a £7.6 million order for the equipment. The present order, for Triffid transportable equipment, is worth £12 million. Triffid is a modification of a design by Siemens and AEG for the Netherlands, and will work in the Ptarmigan network (*WW* Sept. 77, page 49).

A contract to install 470,000 new lines to the Saudi Arabian automatic telephone system has been won by the Philips/Ericsson/Bell Canada consortium. The project will take three years. Philips and Bell will install the equipment, worth $2 million, and Bell will maintain it for five years.