H. Soetens E.

# 2 The PC20 system

**Contents**

## 2 THE PC20 SYSTEM

A programmable controller is an electronic system that is used for automatic control of machines or processes, with the purpose of increasing safety, reliability and production speed.

These electronic controllers are designated *PLC systems*, where:

*P stands for PROGRAMMABLE*, that is, effortless and simple programming of the various machine or process functions by means of instructions. For that purpose the functions are described as small and easily surveyable program parts, the program as such being stored in the system's program memory. It is clear that great flexibility in programming is of utmost importance.

*L stands for LOGIC*, that is, the execution of the functions depends on certain conditions described in logic (Boolean) algebra. For instance, if an electric motor must be energized when switches A and B are both closed, this will be described as:

CONDITION = A·B.

*C stands for CONTROLLER*. This term requires no further explanation. The controller executes the program written in its memory with amazing accuracy and consistency. It must be stressed here that the intelligence of the controller is no greater than that imparted to its "brains" via the user's program.

Our PC20 system can do more. It can perform with surprising speed data handling operations – shift, count and compare – as well as the basic arithmetic functions.

Any controller must first observe the circumstances that will influence its decisions. This applies equally well to our PC20 system. Figure 2-1 shows its organization. The system observes the world beyond – outputs of controlled process or machine – via its inputs. The levels of the input signals are "noted down" in the scratchpad memory, decisions are made by the central processor – the "active brains" of the system – and transferred to the scratchpad memory for communication to the outside world via the system's outputs. The decisions are derived from the program stored in the program memory. *Note, the system's intelligence depends entirely on the program entered by the user.* Communication with the user is through the programming unit and the programming unit interface. These are shown as dashed blocks: The interface is only necessary for programming and the programming unit is common to a number of PC20 systems. For utmost flexibility, the PC20 comes in modular form (see Section 2.3).

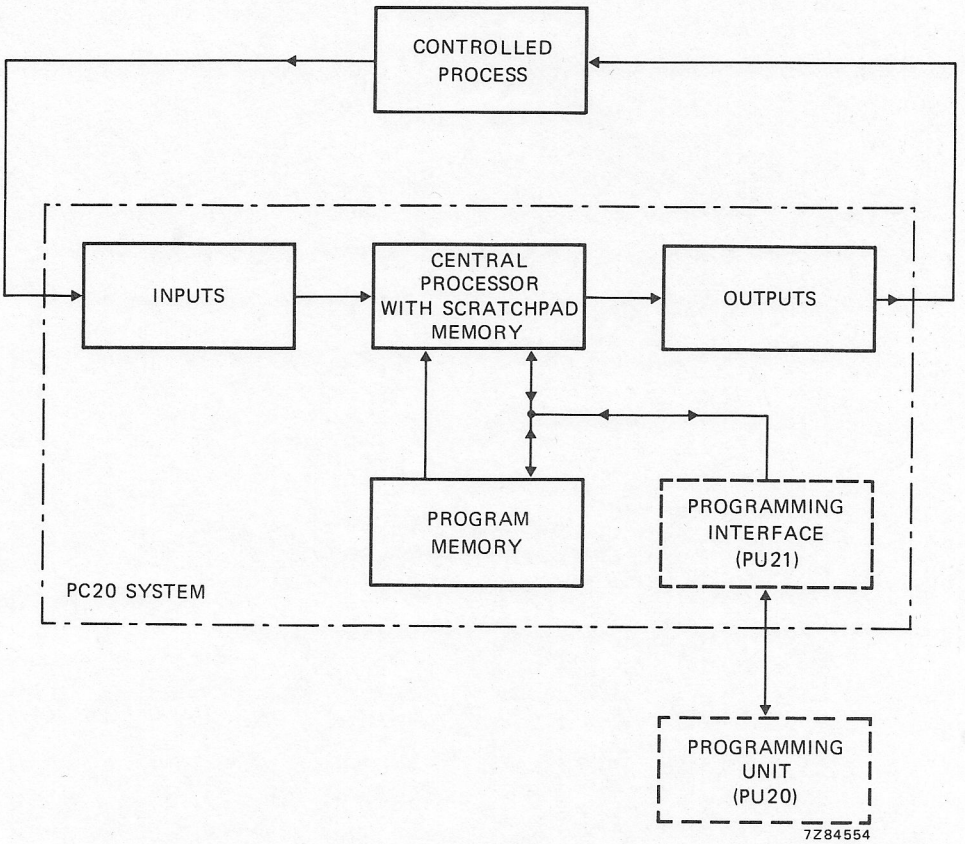



Fig. 2-1 The essential parts of the PC20 system.

Reverting to the previously given example where the motor must be energized when switches A and B are both closed, we obtain the electromechanical circuit of Fig. 2-2. When a PC20 system is used, the circuit will become as given in Fig. 2-3. Switches A and B are connected to inputs m and n and the motor M derives its supply from output p.

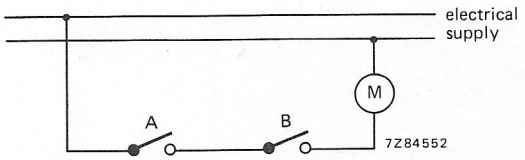



Fig. 2-2 A simple motor control circuit.

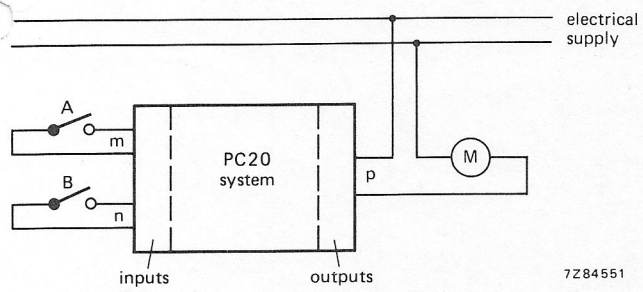



Fig. 2-3 Motor control by PC20 system; m and n are the inputs and p is the output.

Here the software program stored in the system program memory takes the place of the wiring in the electro-mechanical circuit equivalent. The following simple command can be described:

*When switch A AND switch B are closed, motor M must be energized.*

or in Boolean algebra:

CONDITION = A·B.

We will obtain the following program:

| line | instruction | address | |
|---|---|---|---|
| . | . | . | |
| . | . | . | |
| 705 | AND | m | When switch A is closed |
| 706 | AND | n | AND switch B is closed, |
| 707 | EQUALS | p | motor M will be energized. |
| . | . | . | |
| . | . | . | |

The program shown here starting on line number 705 is assumed to form part of a much larger program. It is seen that the program contains instructions and addresses. The AND instructions express together the condition and the EQUALS instruction expresses its execution; the addresses refer to scratchpad places corresponding to the inputs, m and n, and to the output p. (We will see in Chapter 3 that the instruction and the address form the program word.)

If we want to change the command to:

*Energize motor M when switch A OR switch B is closed,* the condition will become:

CONDITION = A + B.

The following program will now be written:

| line | instruction | address | |
|---|---|---|---|
| . | . | . | |
| . | . | . | |
| 705 | OR | m | When switch A is closed |
| 706 | OR | n | OR switch B is closed, |
| 707 | EQUALS | p | motor M will be energized. |
| . | . | . | |
| . | . | . | |

For the command changed to:

*Energize motor M when switch A OR B is open.*

or:

CONDITION = $\overline{A} + \overline{B}$,

the program becomes:

| line | instruction | address | |
|---|---|---|---|
| . | . | . | |
| . | . | . | |
| 705 | OR NOT | m | When switch A is NOT closed |
| 706 | OR NOT | n | OR switch B is NOT closed, |
| 707 | EQUALS | p | motor M will be energized. |
| . | . | . | |
| . | . | . | |

It is clear that a simple program change is sufficient. In the electromechanical example of Fig. 2-2 where a PC20 system is not used the wiring must be adapted to any specific condition; however, the PC20 system will ensure maximum flexibility because a change in wiring is no longer required. It is also seen that the program is amazingly simple in appearance.

## 2.1 The PC20 operating phases

We have seen in the previous section that the PC20 controller must observe the outside world (inputs) prior to making its decisions for external communication (outputs).

It is seen from Fig. 2-4 that observations and decisions occur in alternate periods or phases. In the observation phases $T_1$ to $T_2$, $T_3$ to $T_4$, etc., the PC20 scans its inputs and transfers the logic levels of each input, '0' or '1', to its SCRATCHPAD MEMORY. *So in the scratchpad memory each input must have its own, pre-allocated address.* In order to make a decision, the observations, i.e. the logic input signals, written in the scratchpad are processed during phases $T_2$ to $T_3$, $T_4$ to $T_5$, etc., in agreement with the program previously stored in the PROGRAM MEMORY.

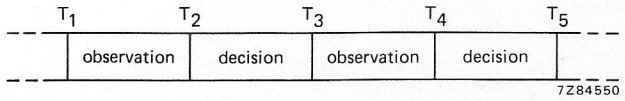



Fig. 2-4 The two major PC20 phases.

Let us have a look at our example program given before:

| line | instruction | address | |
|---|---|---|---|
| . | . | . | |
| . | . | . | |
| 705 | AND | m | When switch A is closed |
| 706 | AND | n | AND switch B is closed, |
| 707 | EQUALS | p | motor M will be energized. |
| . | . | . | |
| . | . | . | |

During the decision phase the following will occur. On line 705 the central processor will fetch the contents of scratchpad address (input) m. On line 706 the contents of scratchpad address (input) n will be fetched then combined in an AND operation with the contents of address m.

Finally, on line 707 the result, '0' or '1' of the AND operation will be stored on scratchpad address p for later transfer to the corresponding output. *To this end, all outputs have their own, pre-assigned addresses in the scratchpad, likewise the inputs.*

Similar operations to those described above will take place for all other lines of program, the results (decisions) being stored in the scratchpad memory. During the next observation phase, after the string of decisions has been completed, these results will be transferred to the outputs for external control. The time diagram of Fig. 2-4 can now be completed to show the flow of data; see Fig. 2-5. The observation phase and decision phase are given here their definite denominations used in this manual, namely: *I/O phase* and *data processing phase.* During an I/O phase, the inputs are written in (observations), input data are processed during the ensuing data processing phase, then in the subsequent I/O phase the results of data processing (decisions) are read out to the outputs. In the same I/O phase, data handling will be re-initiated by again reading in the inputs, etc. So during the I/O phases both write-in and read-out will occur. Each pair of succeeding data processing and I/O phases constitutes a *PC20 cycle,* each PC20 cycle showing a recurrence of events.

To ensure high-speed system performance, the I/O and data processing phases will occur in rapid succession. For 1k program length (1024 lines of program) and a total of one hundred inputs and outputs, the PC20 cycle will be as short as about 1 ms. This implies that the inputs are scanned a thousand times each second and the outputs accordingly changed if so required.

In a practical system, the I/O phase and data processing phase alone will not suffice. Before the system can be put to work, a program must be entered. It may also be desirable to reset all or part of the scratchpad places to zero level. Therefore, a programming and reset phase must be added to the PC20 working cycle. The programming phase is called the UpDate and Check phase. (The significance of this appellation will be clear from what follows.)

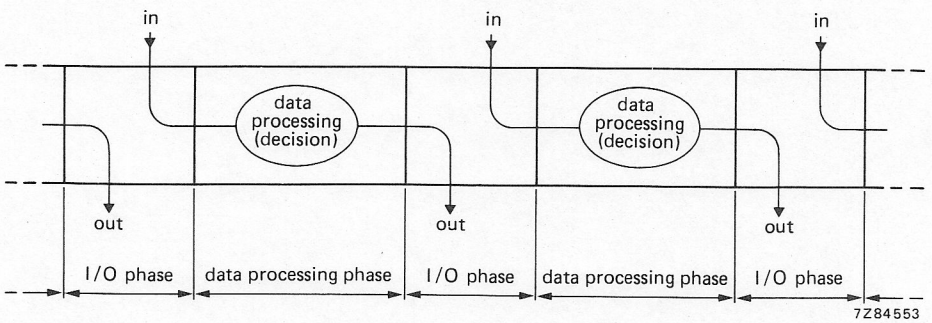



Fig. 2-5 Data flow during I/O phase and data processing phase.

The flow diagram of Fig. 2-6 shows how the PC20 runs through its successive phases. *After switching on or reset of the PC20 system, its central processor will start with the UDC phase.* This phase consists of two parts: *Check* and *Update.* During Check, which is always carried out, the 24 V d.c. supply of the system will be tested. Update will only occur upon request. During Update the program memory is accessible for programming or changing program words ("update"); data can also be entered into the scratchpad. This is explained in Chapter 4. The *reset phase* will always occur during the first PC20 cycle after switch-on or system reset. During the reset phase, all or only a few scratchpad places are reset to zero depending on the input level on the RSME (Reset Scratchpad Memory Enable) terminal of the central processor. For RSME HIGH or FLOATING all scratchpad places will be reset; for RSME LOW only partial reset will occur. The choice between complete and partial reset is by means of a jumper on the back panel. (The back panels are discussed in Section 2.3.9.) It is further shown in the diagram that an END instruction must be included in the program for the system to proceed from the data processing phase to the I/O phase. If the END instruction is not programmed the system will stay in the data processing phase unless it is reset or its d.c. supply interrupted.

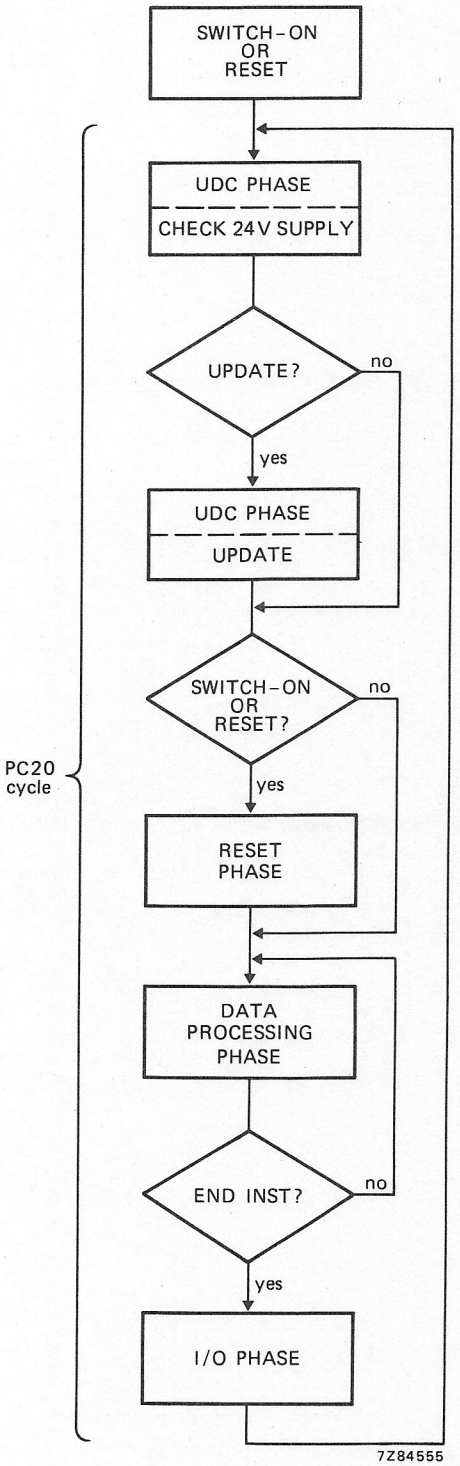



Fig. 2-6 Flow diagram of the PC20 cycle and its four phases.

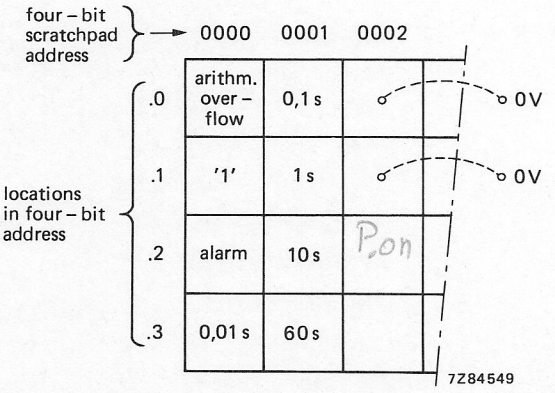



Fig. 2-7 The first twelve locations in the scratchpad memory. Location 0000.1 is always HIGH; location 0000.2 assumes '1' level when the supply voltage falls below 17,5 V. Locations 0000.3, 0001.0, 0001.1, 0001.2 and 0001.3 are the internal clock (fifty per cent signal duty factor). Locations 0002.0 and 0002.1 are normally connected to 0 V in the CP22 central processor.

For explanation of partial reset, Fig. 2-7 shows the first twelve locations of the scratchpad memory. The scratchpad is organized as four-bit places, the first two places, 0000 and 0001, being reserved for specific purposes; this is elaborated in Chapter 3. If during the Check part of the UDC phase the 24 V d.c. supply voltage of the PC20 is found to be lower than 17,5 V, location .2 of four-bit scratchpad place 0000 will assume '1' level (alarm). In the case of partial reset, only locations .0 to .3 of scratchpad place 0002 will be reset to '0' level. It must be noted here that in the CP22 central processor locations .0 and .1 of scratchpad place 0002 are connected to 0 V via a jumper on the module p.c. board; so they are not normally accessible for programming. Partial reset is essential where the scratchpad contents must be retained (battery back-up, Section 2.3.4).

## 2.2 PC20 system build-up

In the foregoing we have outlined the operation of the PC20 system. Here the system will be examined in greater detail, full particulars being given in Chapter 7 (module data), Chapter 10 (specification, assembly and testing) and Chapter 11 (applications).

The PC20 system is of modular design for quick and easy adaptation to any task. A survey of the available modules is given in the next section. Connectors on the rear of these modules fit into connectors on the back panel in the PC20 rack; this back panel has the necessary p.c. wiring for module interconnection. Thus installation of a PC20 is simple, inexpensive and time saving because the tedious and time-consuming job of intermodule rack wiring is eliminated.

The modules are placed side by side. For clarity the PC20 system is shown in an expanded form in Fig. 2-8. Here, only one of each of several input modules IM20 and output modules OM20 is shown. Each of these modules accommodates sixteen input or output stages. The SO20 supply and output module converts the system's 24 V ± 25% d.c. supply voltage into 10 V ± 10% to feed all modules; it contains in addition eight output stages. If the system supply demand exceeds the capacity of a single supply module, two or more SO20 modules are placed in parallel. For programming purposes, the PU20 desk-top programming unit (common to several PC20 systems) is plugged into the connector on the front panel of the PU21 programming interface for communication with the CP22 central processor.

The heart of the system is the CP22 central processor. It contains the timing and control circuit (pulse sequence generator) which is responsible for correct operation of all system functions, separately and in co-ordination. This circuit, together with the data processor, logic analyzer and address processor is housed in two dedicated LOCMOS LSIs. A further essential part is the scratchpad memory. Here data from the inputs are stored ("observations") as are the processed data for transfer to the outputs ("decisions"). For this purpose the input and output stages each have their own, individual one bit place in the scratchpad; addressing is by means of the address lines (address bus) $ADD_{0\text{-}10}$. During the data processing phase, the address information on bus $ADD_{0\text{-}10}$ originates in the address part of the program word. During the I/O phase the address information is supplied by the address counter in the address processor to write in the data from the input stages and transfer the processed data to the output stages. The input and output data are transmitted via the four-wire data bus, $DIO_{0\text{-}3}$, four bits at a time.

The PC20 program is stored in the MM21 program memory (8k16 capacity), each program word occupying an address (program line number) in the program memory. The MM21 is addressed via the thirteen lines $APM_{0\text{-}12}$: the sixteen-bit program words (Chapter 3) will successively appear on lines $PMB_{0\text{-}15}$ for storage in the CP22 program buffer. Here the program word – instruction part and address part – will be retained as long as necessary for processing.

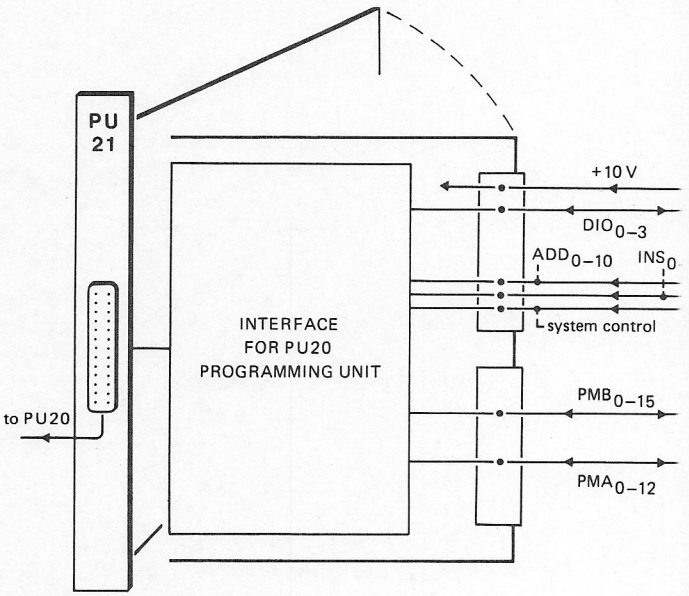
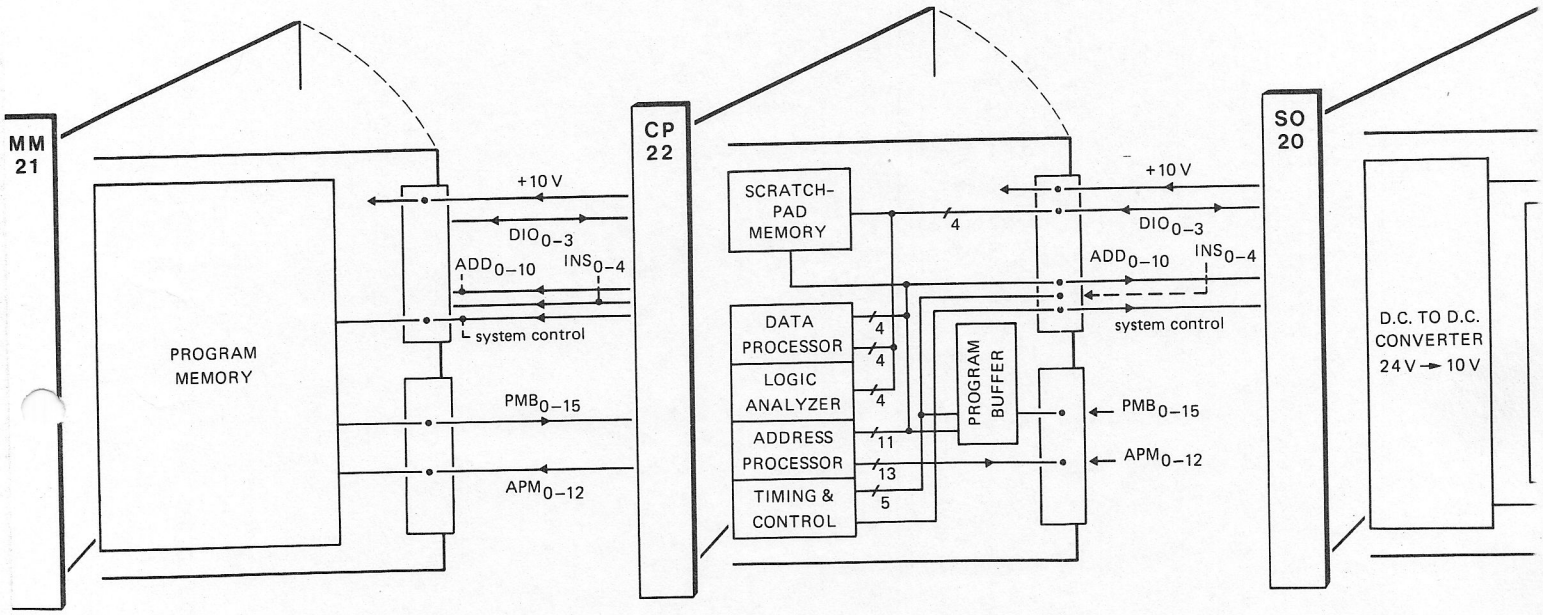
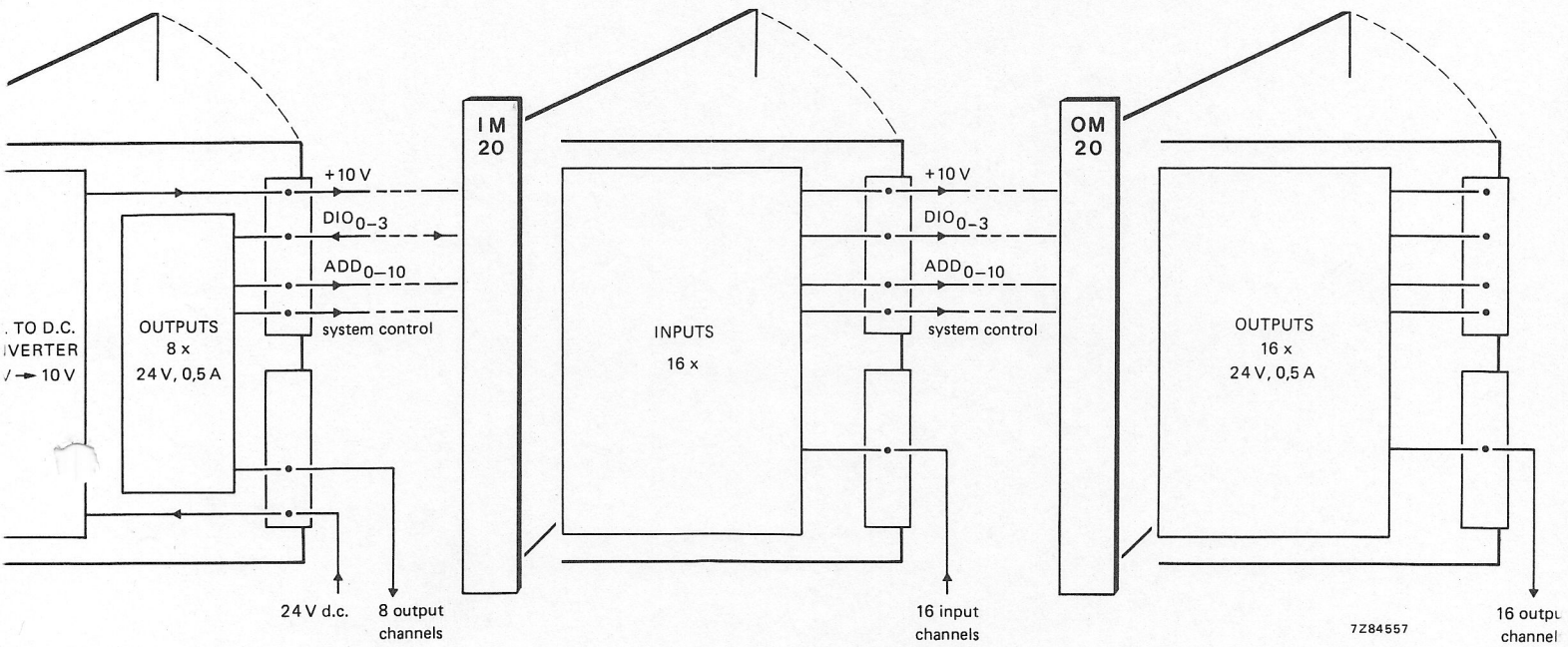



Fig. 2-8 Expanded view of PC20 system using CP22 central processor and MM21 program memory.
$DIO_{0\text{-}3}$ = data bus;
$ADD_{0\text{-}10}$ = address bus;
$INS_{0\text{-}4}$ = wires for instruction code;
$APM_{0\text{-}12}$ = wires for addressing MM21;
$PMB_{0\text{-}15}$ = wires for transmission of program word.


MM
21
PROGRAM
MEMORY
+10 V
DIO0–3
ADD0–10
INS0–4
system control
PMB0–15
APM0–12
CP
22
SCRATCH-
PAD
MEMORY
4
DATA
PROCESSOR
4
LOGIC
ANALYZER
4
ADDRESS
PROCESSOR
11
TIMING &
CONTROL
13
5
PROGRAM
BUFFER
+10 V
DIO0–3
ADD0–10
INS0–4
system control
PMB0–15
APM0–12
SO
20
D.C. TO D.C.
CONVERTER
24 V → 10 V


. TO D.C.
VERTER
/ → 10 V
OUTPUTS
8 x
24 V, 0,5 A
+10 V
DIO0–3
ADD0–10
system control
IM
20
24 V d.c.
8 output
channels
INPUTS
16 x
+10 V
DIO0–3
ADD0–10
system control
16 input
channels
OM
20
OUTPUTS
16 x
24 V, 0,5 A
7Z84557
16 outpu
channel

The program buffer permits reading out of a new program word while the previous one is being used for processing, thus speeding up operation of the central processor.

Table 2-1 illustrates the functions of the signal lines with reference to the sections where these functions are discussed.

As a conclusion to this section we will describe the processing part of the central processor using Fig. 2-9 for guidance. The flow of the most important signals is shown in heavy lines.

The program memory is addressed via lines $PMA_{0-12}$. As a result the program word (sixteen bits) will appear, via the program buffer, on the $INS_{0-4}$ lines (five-bit instruction part of program word) and the $ADD_{0-10}$ bus (eleven-bit address part of program word). For addressing purposes the address bus $ADD_{0-10}$ extends to the scratchpad memory and to the input and output modules. Address information is supplied during the data processing phase by the address part in the program word for addressing the scratchpad memory; it is supplied during the I/O phase by the address counter for addressing both the scratchpad and the input and output modules. The address bus $ADD_{0-10}$ is also used during the data processing phase to specify, in the jump instructions, the next line number or the number of lines to be skipped. Bidirectional flow of data is via the $DIO_{0-3}$ data bus (four data bits at a time).

As shown in the diagram of Fig. 2-9 (see also Fig. 2-8), the processing part of the central processor consists of:

- the timing and control circuit (pulse sequence generator);
- the logic analyzer;
- the data processor;
- the address processor.

**Timing and control (pulse sequence generator)**

The pulse sequence generator is the processor's nerve centre. It has three major functions.

1. It controls the initiation and termination of each of the four phases of the PC20 system (Section 2.1). Within each phase start pulses are generated at the correct time to initiate the actions of the various parts in the processor as well as those of other circuits in the PC20 system, such as the scratchpad and program memory, the input and output modules, etc.

2. It decodes the binary instruction signals appearing on the instruction lines $INS_{0-4}$ and sends the appropriate commands to the various parts in the data processor, the address processor and the logic analyzer.

3. It controls the internal and external traffic of data on the four-wire $DIO_{0-3}$ data bus: internal for exchange of data between the various processor parts; external for data communication between the scratchpad memory, the input and output modules and the PU21 programming interface (Fig. 2-8).

*TABLE 2-1 Functions of signal lines; $ADD_{0-10}$ is the address bus and $DIO_{0-3}$ the data bus.*

| wiring code | lines between modules | function | reference to Section |
|---|---|---|---|
| $ADD_{0-10}$ | CP22 → IM20, OM20, SO20 | addressing. | see above |
| | CP22 → PU21 (PU20) | SMA* monitoring. | 4.4.2 |
| $DIO_{0-3}$ | IM20 → CP22 | write in data from inputs. | see above |
| | CP22 → OM20, SO20 | transfer data to outputs. | see above |
| | CP22 → PU21 (PU20) | display $(SMA)^+$. | 4.4.1 |
| | PU21 (PU20) → CP22 | enter data into SMA*. | 4.4.5 |
| $INS_{0-4}$ | CP22 → PU21 (PU20) | display inst. code: | 4.3 |
| | | search instruction. | 4.3.6 |
| $APM_{0-12}$ | CP22 → MM21 | address MM21. | see above |
| | PU21 (PU20) → MM21 | address MM21 for: | |
| | | editing; | 4.3 |
| | | changing prog. word; | 4.4.4 |
| | | loading prog. word. | 4.10 to 4.12 |
| | MM21 → PU21 (PU20) | line monitoring. | 4.4.1 |
| $PMB_{0-15}$ | MM21 → CP22 | store program word in program buffer for data processing. | see above |
| | PU21 (PU20) → MM21 | edit; | 4.3 |
| | | change program word; | 4.4.4 |
| | | load program. | 4.10 to 4.12 |
| | MM21 → PU21 (PU20) | display program word; | 4.3 to 4.6 |
| | | dump program. | 4.7 to 4.9 |

* Scratchpad memory address. $^+$ Contents of scratchpad memory address.

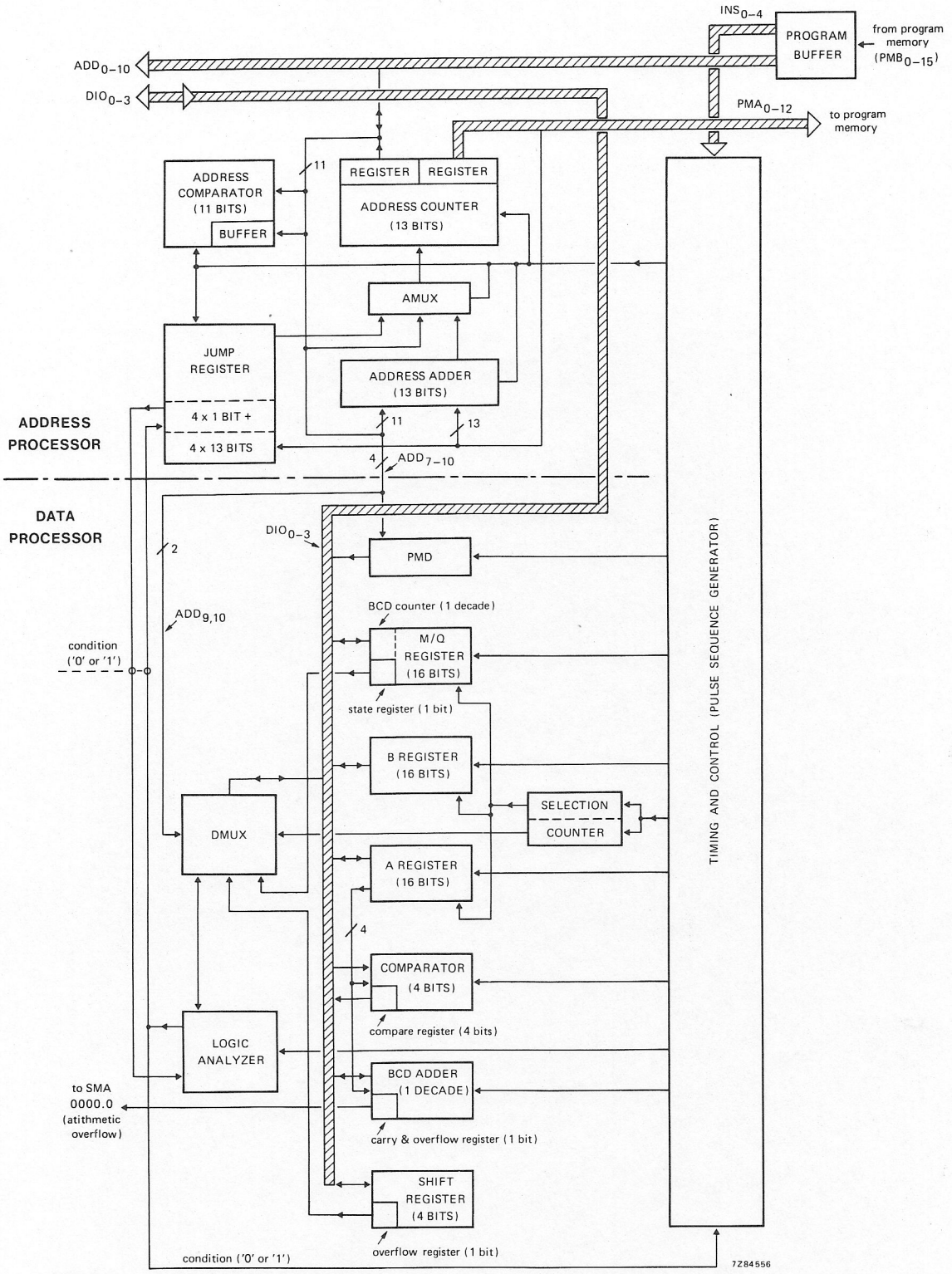



Fig. 2-9 Block diagram of processing part of central processor. The data processor, address processor, logic analyzer and timing and control are contained in two dedicated LOCMOS LSIs. For coded wiring see caption to Fig. 2-8.

**Logic analyzer**

The logic analyzer calculates the condition, which is defined in the PC20 program as a Boolean expression (see beginning of this chapter). The value, '0' or '1', of the condition is decisive for the execution of the conditional execute instructions. Therefore, the condition line is carried to the pulse sequence generator, as seen in Fig. 2-9.

**Data processor**

In Fig. 2-9 the data processor is shown in greater detail to explain its various functions. The $DIO_{0-3}$ data bus extends to within this processor for connection to the various registers.

*A register.* The A register is a sixteen-bit FIFO* register that operates either as a four-by-four bit register for the four-bit instructions or as a sixteen-by-one bit register for the one-bit instructions (Chapter 3). It is connected to the $DIO_{0-3}$ bus; in addition, outputs go to the comparator (for comparison) and to the BCD adder (for arithmetic operations). The A register is the most extensively used in the PC20 instructions.

*B register.* The organization of the B register is identical to that of the A register. The B register is only connected to the $DIO_{0-3}$ bus and is solely used in arithmetic operations.

*M/Q register.* This register is also a sixteen-bit FIFO type; it acts as a four-by-four bit or sixteen-by-one bit register depending on the type of instruction, four-bit or one-bit, respectively. The first four of sixteen bit places are organized as a BCD counter. The M/Q register is used in divide and multiply operations. Its BCD counter part is of further use in count-down and count-up instructions (Section 3.2.9). Down counting occurs by loading the nine's complement, counting up and taking the nine's complement of the result. The state register is initially set to '1' level but will be reset to '0' level for any counter position other than 9 (up counting) or 0 (down counting); for counter position 9 or 0, respectively, the contents of the state register will not change. The carry register – not shown in Fig. 2-9 – transfers a "carry" or "borrow" to the next count instruction.

*Selection counter.* The selection counter selects successive four-bit words stored in the A, B and M/Q register. Successive bits in a four-bit word are selected in the data-multiplexer (DMUX).

*DMUX (data multiplexer).* The DMUX selects single bits from the various auxiliary registers, such as the state register in the BCD counter included in the M/Q register and the overflow register in the shift register. It also selects single bits out of a four-bit word originating in the A, B or M/Q register. Further, one out of four bits is chosen (address lines $ADD_{9,10}$) for processing by the logic analyzer and for storing the result of logic processing in the scratchpad.

* First In, First Out.

*Comparator.* The four-bit comparator has four inputs connected to the $DIO_{0-3}$ bus and four inputs connected to the extra outputs of the A register. Comparison occurs by comparing the four bits on the $DIO_{0-3}$ bus with the four bits presented by the A register (Section 3.2.8). The result of comparison ($=, <, >, \neq$) is stored in the compare register to be used in a new comparison and to be ultimately stored in the scratchpad memory.

*BCD adder.* Here the various arithmetic operations are carried out (Section 3.2.7). The BCD adder adds a digit (BCD code) extracted from the B register and transferred to the $DIO_{0-3}$ bus to a digit in the A register and stores the sum in the latter register. Subtraction occurs by taking the nine's complement of the digit on the data bus. The carry and overflow register transfers a "carry" or "borrow" ('1' level) to the next addition or subtraction. This register will also assume the '1' state if the arithmetic result is greater than 9999 (register capacity exceeded) or negative, or if the divisor is zero. The carry and overflow register is coupled to scratchpad address 0000.0.

*Shift register.* The shift register is used in shift operations (Section 3.2.10); the fifth cell in this register acts as the overflow register.

*PMD.* The PMD (Program Memory Data) is a quadruple buffer that transfers to the $DIO_{0-3}$ data bus a four-bit constant originating in the program memory and presented on input lines $ADD_{7-10}$. It is used in the FETCH CONSTANT instruction (Section 3.2.5).

**Address processor**

The address processor decides what line of program (program memory address) to read next. Because of the presence of jump instructions in the program, the program lines will not always be read in numerical order. The address processor contains the following major parts (Fig. 2-9):

*Address counter.* This is a binary counter that serves a dual purpose: It is used in the I/O phase for addressing the scratchpad as well as the input and output modules ($ADD_{0-10}$ address bus) and in the data processing phase for addressing the program memory ($APM_{0-12}$ lines). At the start of the data processing phase, the address counter is reset by the pulse sequence generator (start line no. 0000). It will proceed step by step for most instructions (reading of program lines in numerical order). However, via the address multiplexer (AMUX) the address counter can be forced into a position (line number) as prescribed by jump instructions in the program. In the I/O phase, the address counter starts from the position (first address) specified by the END instruction; see Section 3.2.14.

*Address adder.* This is a thirteen-bit adder that is used in the JUMP FORWARD RELATIVE FALSE and JUMP BACKWARD RELATIVE FALSE program instructions (Section 3.2.13). It is fed by the $APM_{0\text{-}12}$ lines (line of program) and by the $ADD_{0\text{-}10}$ lines (address part of program word specifying the number of lines to jump forward or backward). Subtraction occurs by adding the two's complement (jump backward).

*Jump register.* This register stores the line number where a JUMP TO SUBROUTINE program instruction appears in the program as well as the appropriate condition (Section 3.2.11). It is a LIFO* register type and its organization is as follows:

- four-by-thirteen bits for storing the line number;
- four-by-one bit for storing the condition values.

It is seen that up to four line numbers and condition values can be kept in the jump register. The condition is derived from the logic analyzer and returned to the logic analyzer upon the RETURN instruction (Section 3.2.12).

*Address comparator.* The address comparator serves to indicate the end of the I/O phase. It compares the position (address) of the address counter with the address stored in its buffer by the LSTIO** instruction occurring last in the data processing phase. In the I/O phase the address counter starts at the first address (position) specified in the END instruction and steps through its positions until the position is reached that agrees with the information stored in the buffer of the address comparator. Then the pulse sequence generator will end the I/O phase and initiate the UDC phase.

*AMUX (address multiplexer).* The AMUX has three inputs to force the address counter into a predetermined position:

- Input from jump register (RETURN program instruction).
- Input from program buffer – lines $ADD_{0\text{-}10}$ (JUMP TO SUBROUTINE and END program instructions).
- Input from address adder (JUMP RELATIVE program instructions).

## 2.3 The PC20 modules and their functions

In the preceding section we have described the essential parts of the PC20 system and their basic functions. The PC20 is a modular system and its modules are discussed in detail here. It will be seen that the range of modules available allows the PC20 to communicate with eight-bit data sources and receivers, operate in master/slave conficurations, and communicate with teleprinters, compact cassette recorders, VDUs, data loggers and computer systems.

* Last In, First Out.

** LaST Input or Output address; Section 3.2.14.

The following parts exist for building a PC20 programmable controller:

- Input Module IM20.
- Output Modules OM20 and OM21.
- Supply and Output module SO20.
- Central Processor module CP20, CP21 and CP22.
- Program Memory Modules MM20 and MM21.
- Desk-top Programming Unit PU20 and interface module PU21.
- Input/output interface modules RP20 and RS20.
- Input/output interface module VI20.
- Back Panels BP23, BP25 and BP26.
- Small Controller cabinet SC20.

### 2.3.1 Input module IM20

A total of up to 127 input modules IM20 and output modules OM20 can be directly addressed† in a PC20 system. Each module contains sixteen identical input stages, thus a maximum of 2032 bits of input and output data can be handled. Individual input modules are addressed by nine lines of an eleven-line addressing system, and individual groups of four input stages in a module are successively selected by two lines in the same addressing system. Module identification connections on the back panel allow specific addresses in the scratchpad memory to be assigned to each input module. When addressed by the central processor, the input module will generate an identification signal to prepare the central processor to accept the data of the addressed group presented on the four-line internal data bus.

The sixteen input stages are driven from standard 24 V ± 25% machine signals (voltage between input terminals 0 V to 7 V or floating for logic LOW and 17 V to 30 V for logic HIGH) and can be adapted for drive from 5 V TTL levels (0 V to 0,8 V or floating for logic LOW; 3,5 V to 6 V for logic HIGH).

A photocoupler in each input stage ensures electrical isolation from the controlled process as well as between input stages, thus minimizing the effect of noise on the input lines. Common-mode rejection is provided.

Each input stage has a delay network for input noise suppression. Typical delay time is 1 ms, which can be increased by adding delay capacitance on the module printed-wiring board (double Eurocard): approximately 0,015 $\mu$F/s; a delay of 10 ms suffices to suppress the effect of contact bounce where the input signal is provided by a relay.

Sixteen LEDs (Light-Emitting Diodes) indicate the state of the input stages (lit when input stage is active – conducting). The LEDs can be switched off to save current consumption.

† For handling a larger number of modules, see Chapter 10.

### 2.3.2 Output modules OM20 and OM21

The OM20 output module contains sixteen identical output stages and the OM21 eight output stages with a higher output current rating. A total of up to 127 input modules IM20 and output modules OM20 can be used in a PC20 system*, thus a maximum of 2032 bits of input and output data is catered for; however, two output modules OM21 can be taken for one OM20 module. Individual output modules are addressed by nine lines (ten lines in the case of an OM21) of an eleven-line addressing system, and individual groups of four output stages in a module are successively selected by two lines (one line in the case of an OM21) in the same addressing system. Module identification connections on the back panel allow specific addresses in the scratchpad memory of the central processor to be allocated to each output module. Upon addressing by the central processor, the processed data presented on the four-line internal data bus are clocked into the four latch flip-flops of the selected group of output stages. The data are passed to the output stages via photocouplers, which provide electrical isolation from the controlled process, thus ensuring minimum noise effects.

The output stages are intended for a grounded load; they have short-circuit protection. A LED in each output stage indicates its condition – it is lit when the output stage conducts. A flywheel diode is included in each output stage to handle an inductive load. The output stages are fed from the 24 V d.c. machine voltage and will be forced into their non-conducting state when this voltage drops below 16 V.

For the OM20 (sixteen output stages), the current rating per output stage is 0,5 A and the rated output current 6 A for all output stages conducting. For the OM21 (eight output stages), these ratings are 2 A and 8 A, respectively. The supply voltage to drive the output circuitry via the output stages is 24 V d.c. ± 25%.

### 2.3.3 Supply and output module SO20

The supply and output module SO20 provides the logic supply voltage for the PC20, thus eliminating the necessity for the user to build a supply system himself. In addition, it contains eight output stages satisfying the OM20 specifications (see previous section); the rated output current is 3 A for all output stages conducting.

The built-in supply is a d.c.-to-d.c. converter fed by the standard 24 V ± 25% machine voltage and generating a logic supply voltage of 10 V ± 10%. Electrical isolation exists between d.c. input and d.c. output. This supply is short-circuit proof (current limiting mode). D.C. output current rating is 1,7 A (17 watts nominal), adequate to power a small PC20 controller. For a larger PC20 system, two or more SO20 modules are paralleled.

An alarm circuit monitors the 24 V d.c. supply and will be activated when this voltage drops below 17,5 V. It has two alarm outputs, one for external use (hardware) and one used internally for processing (software programming).

A LED indicates that the 24 V supply voltage is above the specified minimum level of 17,5 V.

### 2.3.4 Central processor modules CP20, CP21 and CP22

The central processors CP20, CP21 and CP22 use low-current drain, high noise-immunity LOCMOS technology. They contain the necessary electronic circuitry for communication with all parts of the PC20 programmable controller and include two dedicated LSIs for data and address processing. The central processor constitutes in essence the brains of the PC20, asking the input modules for input data, processing the data in conformity with the instructions read from the program memory, and supplying the results to the output modules for process control. The internal timing for all of these functions is generated by the central processor.

While the PC20 operates on-line, two major phases will occur in alternation: the input/output (I/O) phase and the data processing phase.

During an I/O phase, the central processor addresses each input module in turn and stores the data provided by these modules in its scratchpad memory, four bits at a time. The scratchpad memory is volatile and has provision for battery back-up for those applications where loss of data due to a power failure is not tolerable. In the same I/O phase, as input data are being stored in the scratchpad memory, the data processed during the previous processing phase are being clocked into the latch flip-flops of the output stages, four bits at a time.

In the data processing phase the input data stored in the scratchpad memory are processed in accordance with the sixteen-bit program words – part instruction, part address – obtained from the program memory and the resulting output data written in the scratchpad to control the output stages during the next I/O phase.

There are three types of central processor, the CP20, CP21 and CP22.

The CP20 has a 1k (¼k4) scratchpad memory and an integral EPROM program memory with 2k16 capacity. It is used in those PC20 systems where a relatively small scratchpad and program memory will suffice.

The CP21 has similar specifications to those of the CP20 but it has an integral 1k16 C-MOS RAM program memory. On-the-board battery back-up is provided for typically 40 hours.

For larger PC20 controllers, the CP22 with 2k4 scratchpad memory is available. In this memory, either single bits are addressable (max. 2k) or four-bit bytes can be addressed (up to 2k). The CP22 operates together with the MM20 or MM21 memory module.

The organization of the scratch-pad memory of the three central processor types is shown in Fig. 2-10.

* For handling a large number, see Chapter 10.

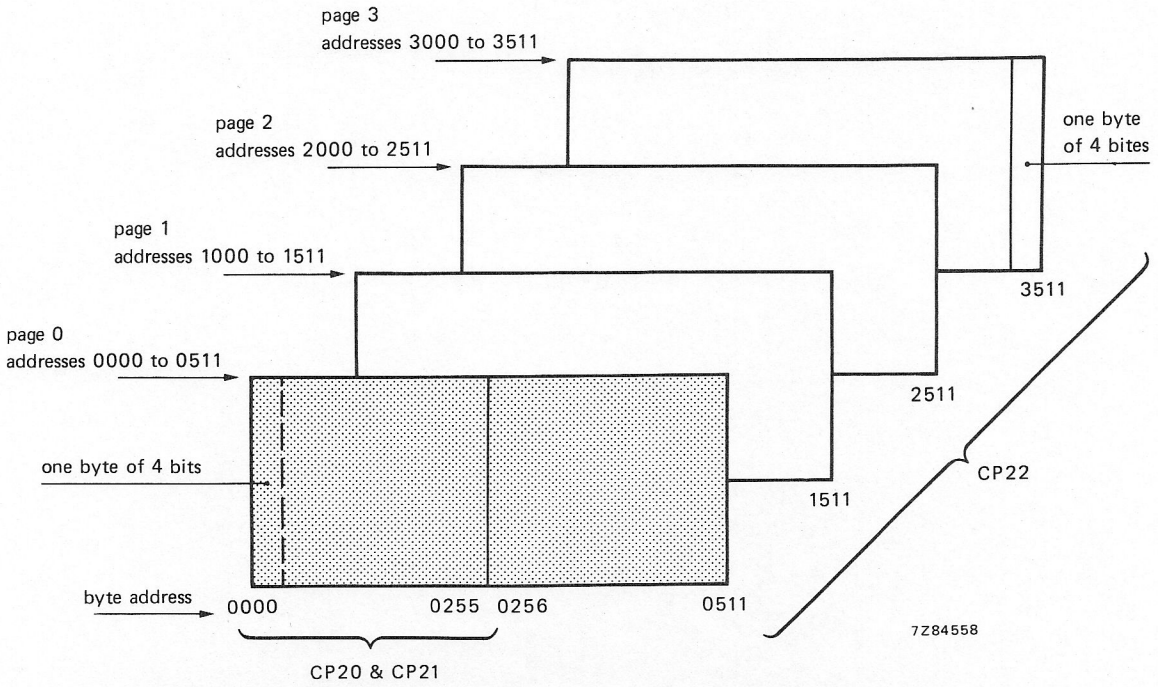



Fig. 2-10 Schematic presentation of CP22 scratchpad memory. In hatched area, page 0, each bit is individually addressable (2k memory) or each byte of four bits can be addressed (½k4). In white areas, pages 1 to 3, only bytes can be addressed. The capacity of the CP22 scratchpad memory is 4 × 512 four-bit bytes (2k4); that of the CP20 and CP21 scratchpad memory is 256 × 4 = 1024 bits (1k1) or 256 four-bit bytes (¼k4).

### 2.3.5 Program memory modules MM20 and MM21

Either the MM20 or MM21 memory module is used in conjunction with the central processor CP22. The MM20 has up to 8k16 EPROM memory capacity; an 8k16 C-MOS RAM with integral battery back-up is included in the MM21. The PC20 system can handle 8k16 memory capacity.

Table 2-2 surveys the memory and battery back-up facilities of the central processor and program memory modules available for the PC20 system.

### 2.3.6 Desk-top programming unit PU20 and interface module PU21

Together with the facility of the C-MOS RAM program memory, the programming unit PU20 gives the PC20 programmable controller its flexibility. In hard-wired systems, each change in function is time-consuming and requires considerable effort and checking. With the programming unit and a CP21 central processor or MM21 memory module, function changes are readily made by simply changing the program. The mains-powered desk-

*TABLE 2-2 Program memory capacity and battery back-up of central processor and program memory modules.*

| module | scratchpad memory | program memory | | battery back-up | |
|---|---|---|---|---|---|
| | | capacity | type | internal | external |
| CP20 | ¼k4 | 2k16 | EPROM | no | yes |
| CP21 | ¼k4 | 1k16 | C-MOS RAM | yes* | yes* |
| CP22 | 2k4 | – | – | no | yes |
| MM20 | – | max. 8k16 | EPROM | – | – |
| MM21 | – | 8k16 | C-MOS RAM | yes | yes |

* For scratchpad and program memory.

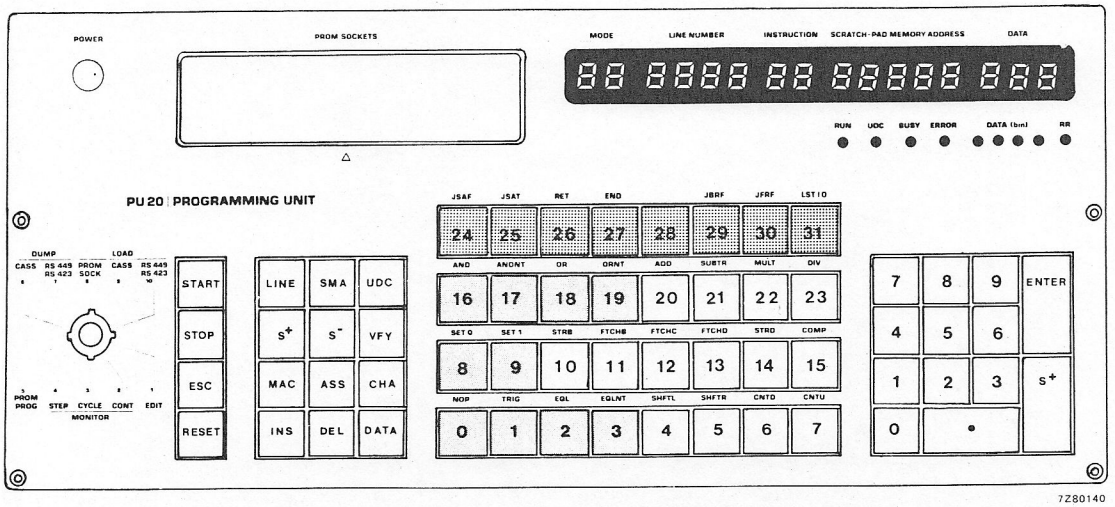



Fig. 2-11 PU20 layout.

top programming unit PU20 is portable and communicates with the central processor in the PC20 via a rack-mounted interface module PU21. For monitoring or program changes, the PU20 is plugged into the connector on the interface front panel; one programming unit can, therefore, be used with any number of PC20 systems. Because the PC20 must be switched off to remove or insert modules, it is recommended that the PU21 be left in its place for as long as program changes may be necessary so that the PU20 can be connected and disconnected without disrupting the operation of the PC20.

The PU20 has facilities for off-line operation (programming and program changes) as well as on-line operation (monitoring and fault finding). It has a key switch which determines the authority of the user. Without a key, the user can only monitor the operation of the PC20 and check the states of the scratchpad places. With the key switch ON, all programming facilities become available and programming as well as changes in an existing program can occur; moreover, cycle-by-cycle and step-by-step modes become available to facilitate fault finding. In the ON-position, the key is retained in its lock.

Figure 2-11 shows the layout of the PU20. In the lower part, the keyboard and the mode selector switch are located. The upper part accommodates the multi-digit LED display (seven-segment type) and EPROM sockets are located under a hinged cover to insert EPROMs for program loading or dumping. Connectors at the rear of the PU20 are for two-way communication with peripheral equipment, such as an audio cassette recorder (five-pins DIN connector) or a teleprinter, minicomputer, data logger, VDU, etc. (RS449/423).

The keyboard permits the user to issue all those commands necessary for communication with the PC20 and peripheral equipment. The accuracy of any command can be checked on the multi-digit tell-back LED display, which shows the mode of operation, line number (program memory address), the program word (instruction part and address part) and the contents, in numerical form, of the selected scratchpad address.

The mode switch selects any of the following modes:

- EDIT, for programming (off-line).
- MONITOR CONT(INUOUS), for checking the process functions.
- MONITOR cycle-by-cycle or step-by-step for fault finding.
- DUMP PC20 program into EPROMs (with verifying), cassette or RS449/423 peripherals.
- LOAD PC20-program memory from EPROMs, cassette or RS449/423 peripherals, with verify facilities (this mode only possible with CP21 central processor or MM21 memory module).

During monitoring, changes can be made in an existing program; data can also be written in the scratchpad.

### 2.3.7 Input/output interface modules RP20 and RS20

The RP20 interface is a module for bidirectional communication and is intended to handle logic information in parallel format. It has a 16 x 8 bit buffer memory and is connected to an eight-bit data bus for communication with up to sixteen data transmitters and/or receivers. They are selected in turn by enable signals originating in the RP20; logic signals from data sources and/or receivers can be inverted where the need exists. The data bus and the enable and control lines are photocoupler-isolated; they can handle 24 V machine levels as well as 5 V TTL levels. One PC20 system can be equipped with several RP20 modules.

The RS20, designed to handle data in serial format, allows two-way communication between master and remote slave stations, which are interconnected through a coaxial cable (up to 500 m length). Transmission speed is 400 kbits per second. The RS20 has a 256-bit buffer memory. It can be adapted on the back panel for three modes of operation: RS20A, RS20B or RS20C. The RS20A (master mode) is placed in the master PC20 system for communication with one slave station.

However, by including several RS20A interfaces in the PC20, the master station will be made to control more than one slave station. The slave station can consist of a PC20 system containing the RS20B interface (active slave mode) to handle 256 bits of data. Alternatively, only input and output modules may be included in the slave station to transfer data without further processing (passive slave mode); here, the RS20C is used.

### 2.3.8 Input/output interface module VI20

The VI20 is equipped with the standard RS449/423 interface for bidirectional communication with peripherals, such as TTYs, VDUs, data loggers and high-intelligence computer systems.

### 2.3.9 Back panels BP23, BP25 and BP26

Standard racks to accommodate the PC20 can be obtained in numerous versions from many suppliers. The back panels, BP23, BP25 and BP26 eliminate the tedious and time-consuming job of intermodule rack wiring, and thus make installation of a PC20 system simple and inexpensive.

The BP23 is intended for those systems requiring a CP20 or CP21 central processor unit or a CP22 central processor unit with an MM20 or MM21 program memory module. There is an eighteen-slot place to take the input and output modules including the SO20. To communicate with data sources and/or receivers, remote stations, peripherals, one or more input and/or output modules are replaced by the RP20, RS20 or VI20.

The BP25 and BP26, having spaces of fifteen and twenty-one slots, respectively, are for use in an extension rack or a passive slave station which employs only input and output modules or input/output interfaces.

### 2.3. 10 Small controller cabinet SC20

For a small controller, the SC20 cabinet intended for wall mounting is available. There is a space for a PU21 interface, CP20 or CP21 central processor, SO20 and six input, output or interface modules, bringing system capacity to 104 input/output channels (if using input and output modules solely) and 1k16 or 2k16 program-memory capacity. The connections to the outside world (screw terminals on BP22 back panels) are protected by a sloping cover on the lower part of the cabinet.

Fig. 2-12 illustrates how the modules discussed above are used in the PC20 system.

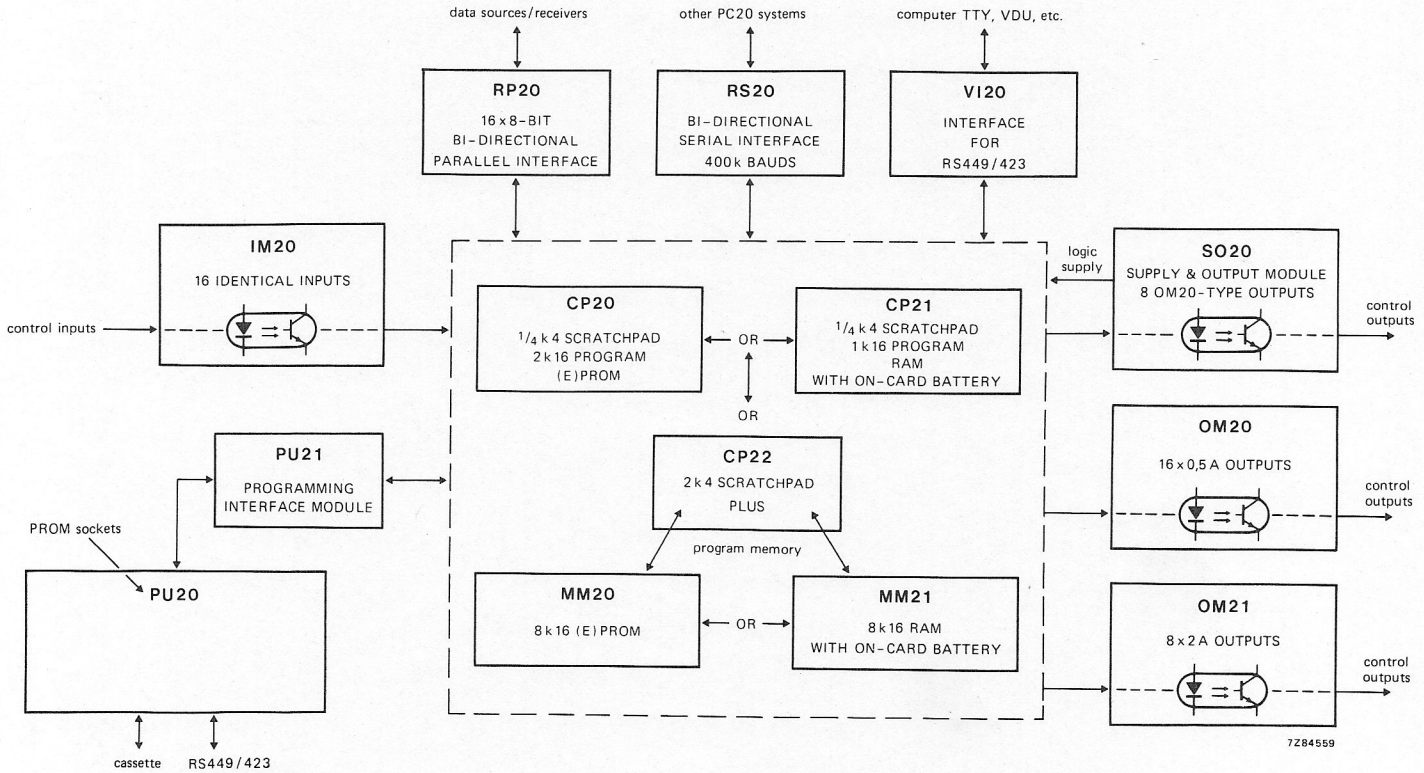



Fig. 2-12 Showing the use of the various modules in the PC20 grammable controller. The PU20 programming unit is common to several controllers.

# 3 The PC20 instruction set

**Contents**

# 3 THE PC20 INSTRUCTION SET

Conflicting requirements in designing PLC systems are a versatile instruction set and short execution times. In our PC20 system this conflict has been resolved by including two dedicated LOCMOS LSIs, one for data processing, one for address processing – see the previous chapter. This results in high-speed system operation and extreme compactness.

Our comprehensive set of thirty-one instructions offers the following:

- Handling of logic data and signalization of changes in logic input functions.
- Set and reset of logic output functions.
- Shift, count and compare.
- Arithmetic operations.
- Program skip, branching and sequencing as well as subroutines.
- Programming the length of the I/O phases for maximum time economy and for process start-up.

This chapter gives a detailed description of the PC20 instructions and makes them fully understandable to the user; it also discusses program build-up. Chapter 4 tells how to operate the PU20 programming unit (see Fig. 3-1) and explains its various modes of operation.

The following abbreviations are used:

LSB for least significant bit;

MSB for most significant bit;

LSD for least significant digit;

MSD for most significant digit;

SM for scratchpad memory;

SMA for scratchpad memory address;

PM for program memory;

PMA for program memory address (line number).

## 3.1 PC20 program build-up

For process control to occur a suitably designed program must be entered beforehand into the program memory of the PC20. This program consists of a number of *lines* each containing a *program word.* It is split up into easily surveyable portions called *instruction blocks.* In each instruction block the PC20 system will make one or more decisions, which may cause a command to be issued to the controlled process, such as operating a heater or a valve. In many instances the decision can be of a purely internal nature, which means that its outcome is "noted down" in the scratchpad memory and will not be communicated externally.

Fig. 3-1 Desk-top PU20 programming unit.

Each instruction block in the program consists of two parts, as follows:

INSTRUCTION BLOCK { CONDITION / EXECUTION

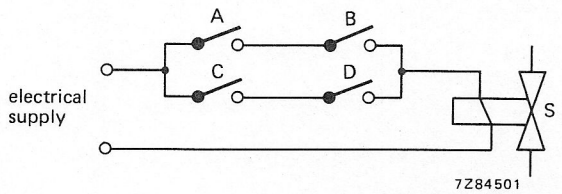



Fig. 3-2 Electrical valve control diagram representing the condition: A·B + C·D.

The *condition* is expressed according to Boolean algebra. So, in the simple valve control diagram of Fig. 3-2 the input conditions A, B, C and D are collectively described as:

CONDITION = A·B + C·D.

The translation of this condition in PC20-program language is:

```
line   prog. word
0000   AND  input A  ┐
0001   AND  input B  ├ condition
0002   OR   input C  │
0003   AND  input D  ┘
 :      :
 :      :
```

The *execution* following the condition is the "valve open" or "valve close" command, which will be written in the program as:

```
line   prog. word
 :      :
 :      :
0004   EQUALS output S → execution
 :      :
 :      :
```

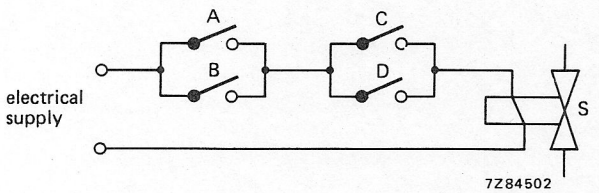



Fig. 3-3 Electrical valve control diagram representing the condition: (A + B)·(C + D).

For the case of Fig. 3-3 the condition is expressed as:

CONDITION = (A + B)·(C + D).

Because the central processor can only read a condition written as a sum of products, the bracketed logic condition shown above must be converted into:

CONDITION = A·C + A·D + B·C + B·D.

The following conversion according to De Morgan's theorem will require fewer lines of program:

$$\text{CONDITION} = \overline{\overline{A}\cdot\overline{B} + \overline{C}\cdot\overline{D}}.$$

In the above program example the condition is described by the four *logic instructions*; "AND input A", "AND input B", "OR input C" and "AND input D". The execution part of an instruction block contains one or more *execute instructions*; in the program example only one execute instruction is used: "EQUALS output S". Depending on the value ('0' or '1') of condition A·B + C·D, this execute instruction will cause output S to assume either state '0' (valve closed) or state '1' (valve open).

To allow either of decisions "close valve" or "open valve" to be made, the condition is written in the result register (condition flip-flop in the central processor); Section 4.4.6 in the next chapter explains how the condition is arrived at while the logic instructions are being read by the central processor.

It is seen that the execute instruction "EQUALS output S" is *unconditional*; that is, it will always be obeyed irrespective of the value of the condition (contents of the result register). As a result, the output state will follow the condition. It should be noted here that other execute instructions in the PC20 instruction set are *conditional*; that is, they will become effective only when the condition has a given value, most of them being executed on condition 'true' (state '1').

As an illustration of what has been discussed above, the flow diagram of Fig. 3-4 shows how, during the data processing phase, the central processor steps through the consecutive instruction blocks. A characteristic property of the PC20 system is clearly shown here: if the condition does not have the value prescribed for execution, the system will proceed to the next instruction block with minimum delay. This may occur in either of the two following ways:

- The execute instructions are read but they will remain ineffective; this will result in a shorter time being spent on a line of program (instruction time).
- A "jump forward" instruction (JFRF, see Section 3.2.13) is used to skip the execute instructions in the instruction block.

Clearly, the time required for an instruction block will become shorter when execution is not desired. So the duration of a data processing phase will depend on the value of the condition prevailing in each instruction block. As a result, appreciable time is saved and high-speed performance ensured.

It is further seen from Fig. 3-4 that the condition formulated at the beginning of an instruction block will apply only to that block. So the next block is opened by defining a new condition.

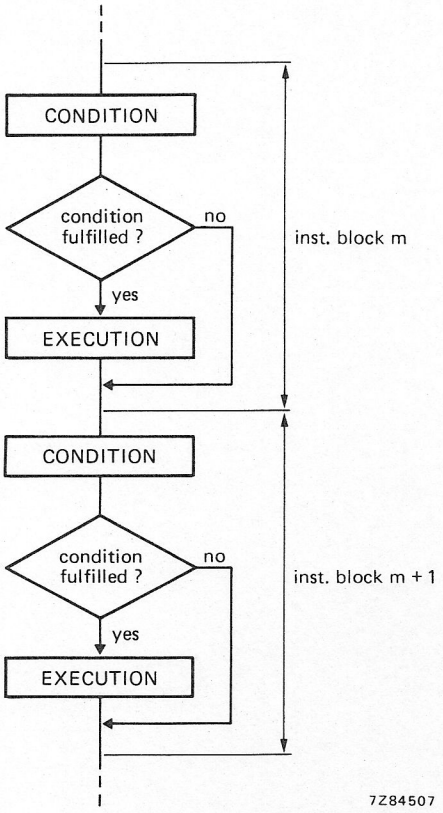



Fig. 3-4 Flow diagram for two successive instruction blocks with conditional execute instructions.

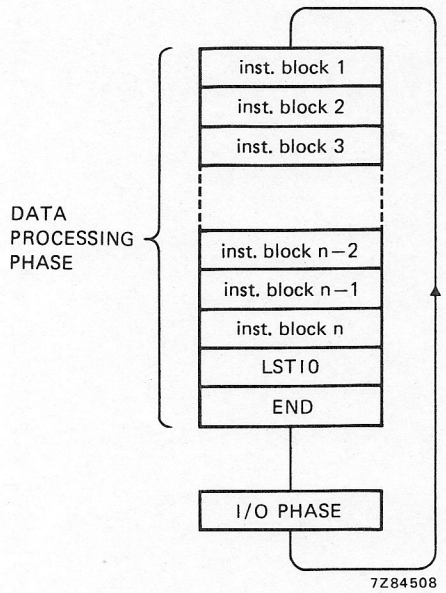



Fig. 3-5 Diagram showing cyclic operation of PC20 system.

The transition of the central processor to the I/O phase must be programmed. To this end two instructions are used as a pair: the LSTIO (LaST Input or Output address) and the END instruction (Section 3.2.14).

*If the END instruction is not programmed the central processor will never come out of the data processing phase unless the PC20 system is reset or its supply interrupted.*

Figure 3-5 shows the PC20 program in its most basic form. It contains a number of instruction blocks and it is terminated by the LSTIO and END instructions. The END instruction will conclude the data processing phase to initiate the I/O phase. (Although the LSTIO and END instructions are used to end the data processing phase, they are not necessarily written at the end of program, as we shall see later in this chapter.)

When the PC20 system enters its first working cycle after switch-on or reset, the condition (result register) will be set to state '1'. The END instruction will force the condition into state '0', which will be the state under which the next data processing phase is entered.

Figure 3-6 shows part of a PC20 program with the instruction blocks labelled. Since each instruction block opens with a condition described in Boolean algebra, the beginning of a block can readily be found by looking for logic instruction no. 16 [AND], no. 17 [ANDNT], no. 18 [OR] or no. 19 [ORNT].

```
#0044 00          [ NOP ]
#0045 17   017.2  [ANDNT]  ┐
#0046 01   102.0  [TRIG ]  │
#0047 13 0023     [FTCHD]  │
#0048 13 0022     [FTCHD]  ├ inst. block n
#0049 13 0021     [FTCHD]  │
#0050 14 0015     [STRD ]  │
#0051 14 0014     [STRD ]  │
#0052 14 0013     [STRD ]  ┘
#0053 16   017.0  [ AND ]  ┐
#0054 01   102.1  [TRIG ]  ├ inst. block n+1
#0055 02   102.2  [ EQL ]  ┘
#0056 16   017.1  [ AND ]  ┐ inst. block n+2
#0057 30 0006     [JFRF ]  ┘
#0058 16   102.2  [ AND ]  ┐
#0059 07 0015     [CNTU ]  │
#0060 07 0014     [CNTU ]  ├ inst. block n+3
#0061 07 0013     [CNTU ]  │
#0062 10   006.0  [STRB ]  ┘
#0063 17   017.1  [ANDNT]  ┐ inst. block n+4
#0064 30 0006     [JFRF ]  ┘
#0065 16   102.2  [ AND ]  ┐
#0066 06 0015     [CNTD ]  │
#0067 06 0014     [CNTD ]  ├ inst. block n+5
#0068 06 0013     [CNTD ]  │
#0069 10   006.0  [STRB ]  ┘
#0070 00          [ NOP ]
#0071 00          [ NOP ]
#0072 31 0030     [LSTIO]
#0073 27 0000     [ END ]
```

Fig. 3-6 Print-out with mnemonics of part of PC20 program showing instruction blocks.

## 3.2 Description of the PC20 instructions

Table 3-1 shows the various ways in which the PC20 instruction set can be broken down. In the centre all of the thirty-one instructions are tabulated under their main headings. At the left-hand side distinction is made between unconditional and conditional instructions (discussed in the previous section); as is seen, most of the conditional instructions will be executed on condition '1'. At the right-hand side the instructions are classified as one-bit instructions, four-bit instructions and program instructions. The program instructions are essential to give flexibility to the program to satisfy the specific requirements of the user.

*TABLE 3-1 The PC20 instruction set.*

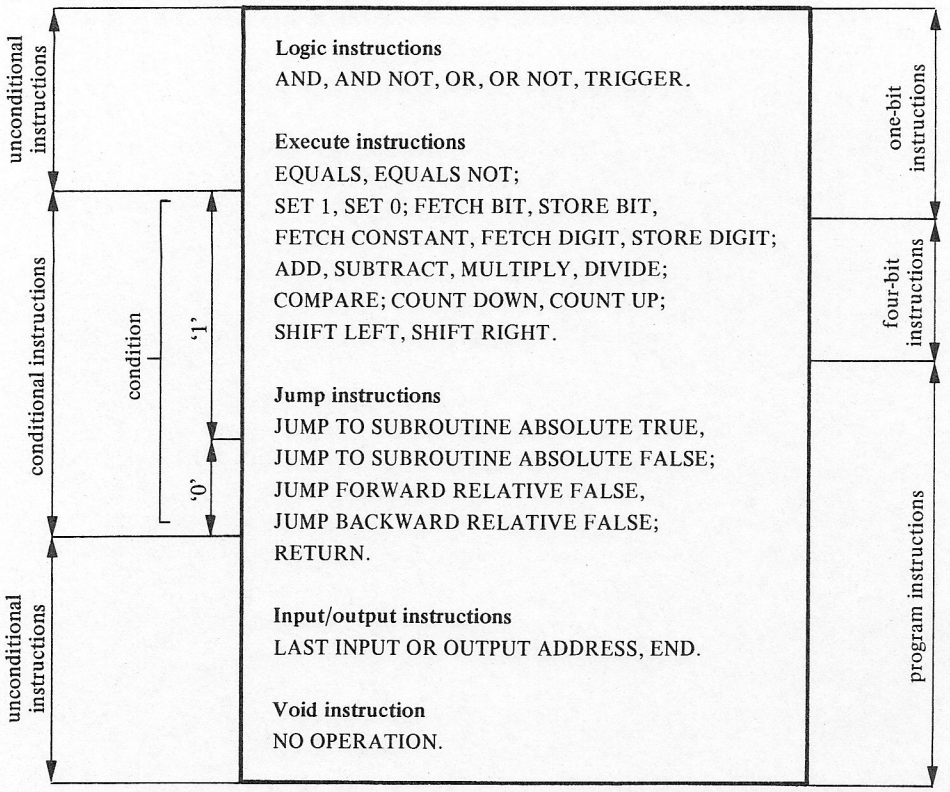



Table 3-2 gives a survey of the sections discussing the PC20 instructions in detail.

To begin with, the function of the one-bit and four-bit instructions will be explained.

The PC20 program memory is made up of a number of sixteen-bit locations where the program is stored, one word per location. So each location represents a line of program. Storage is in binary form as is usual in data processing systems. The information typed in by the user in decimal code is converted by the PU20 programming unit into binary code before being entered into the program memory. Figure 3-7 shows the organization of the sixteen-bit program word. The first five bits, 0 to 4, are reserved for the instruction code; thus, up to thirty-two distinct types of instruction can be given. The remaining eleven bits, 5 (LSB) to 15 (MSB), contain the necessary information to carry out the instruction specified in the instruction code. Their significance is described in the figure and discussed below.

For *scratchpad addressing* all of the eleven bits, 5 to 15, are used and so 2048 ($2^{11}$) addresses can be specified:

- In a *one-bit* instruction, one out of 2048 single-bit places is selected to carry out the single-bit operation prescribed by the instruction code.
- In a *four-bit* instruction, one out of 2048 four-bit places is chosen to carry out the four-bit operation prescribed by the instruction code. This does not apply, however, to the four-bit instruction FETCH CONSTANT; here, only four bits are used, i.e. bits 12 to 15 (Fig. 3-7) to store a *numeral* between 0 and 15 (Section 3.2.5).

*TABLE 3-2 Survey of sections discussing the PC20 instructions.*

| instructions | section |
|---|---|
| AND, AND NOT, OR, OR NOT | 3.2.1 |
| EQUALS, EQUALS NOT | 3.2.2 |
| SET 1, SET 0 | 3.2.3 |
| TRIGGER | 3.2.4 |
| FETCH BIT, FETCH DIGIT, FETCH CONSTANT | 3.2.5 |
| STORE BIT, STORE DIGIT | 3.2.6 |
| ADD, SUBTRACT, MULTIPLY, DIVIDE | 3.2.7 |
| COMPARE | 3.2.8 |
| COUNT DOWN, COUNT UP | 3.2.9 |
| SHIFT LEFT, SHIFT RIGHT | 3.2.10 |
| JUMP TO SUBROUTINE ABSOLUTE TRUE/FALSE | 3.2.11 |
| RETURN | 3.2.12 |
| JUMP FORWARD/BACKWARD RELATIVE FALSE | 3.2.13 |
| LAST INPUT OR OUTPUT ADDRESS, END | 3.2.14 |
| NO OPERATION | 3.2.15 |

It is seen that, with the eleven bits available in the program word for scratchpad addressing, either one out of 2048 one-bit scratchpad places or one out of 2048 four-bit scratchpad places can be selected. If now the 2048 one-bit places are arranged as 512 four-bit places and the 2048 four-bit places are arranged in four groups (pages) of each 512 four-bit places, the set-up of the scratchpad will become as shown in Fig. 3-8. This now is the organization of the 2k4 scratchpad memory in the CP22 central processor. It follows that one-bit instructions can only cover page 0 whereas all of the four pages 0 to 3 can be covered using the four-bit instructions. *Thus, only on page 0 can all single-bit places be individually addressed.* The distinction between coverage for the one-bit and four-bit instructions will, of course, disappear in the case of the ¼k4 scratchpad memory included in the CP20 and CP21 central processors (256 four-bit locations).

In Fig. 3-8, SMA 0511.1 means location .1 (one-bit) on four-bit Scratchpad Memory Address 0511. Further, SMA 2000 indicates four-bit Scratchpad Memory Address 000 on page 2.

In Fig. 3-8 the first three scratchpad addresses, 0000 to 0002, are shown shaded; these have been assigned special functions. The functions of addresses 0000 and 0001 are given in Table 3-3. These addresses can be used in the normal way in programming but their state cannot be influenced externally. Scratchpad addresses 0002.0 and 0002.1 are not normally accessible for programming.

Addresses 0002.2 and 0002.3 are available to the user; they are, however, reset to '0' level when the PC20 programmable controller is switched on or reset (see Chapter 2).

*TABLE 3-3 Functions of reserved scratchpad places 0000 and 0001.*

| address | function | address | function |
|---|---|---|---|
| 0000.0 | Arithmetic overflow | 0001.0 | 0,1 s clock* |
| 0000.1 | Constant '1' | 0001.1 | 1 s clock* |
| 0000.2 | Alarm: 24 V supply voltage ⩽ 17,5 V | 0001.2 | 10 s clock* |
| 0000.3 | 0,01 s clock | 0001.3 | 60 s clock* |

* Fifty per cent duty factor.

Scratchpad addressing is also used to define the last (highest) and first (lowest) four-bit scratchpad place in the LSTIO and END instructions, respectively, between which the scratchpad memory will be scanned during the I/O phase (Section 3.2.14). In a jump-to-subroutine program instruction (Section 3.2.11) the eleven bits 5 to 15 specify the *first line of program* where the subroutine starts. Further, these bits are used to define the *number of lines* to jump forward or backward in the case of a relative-jump instruction (Section 3.2.13).

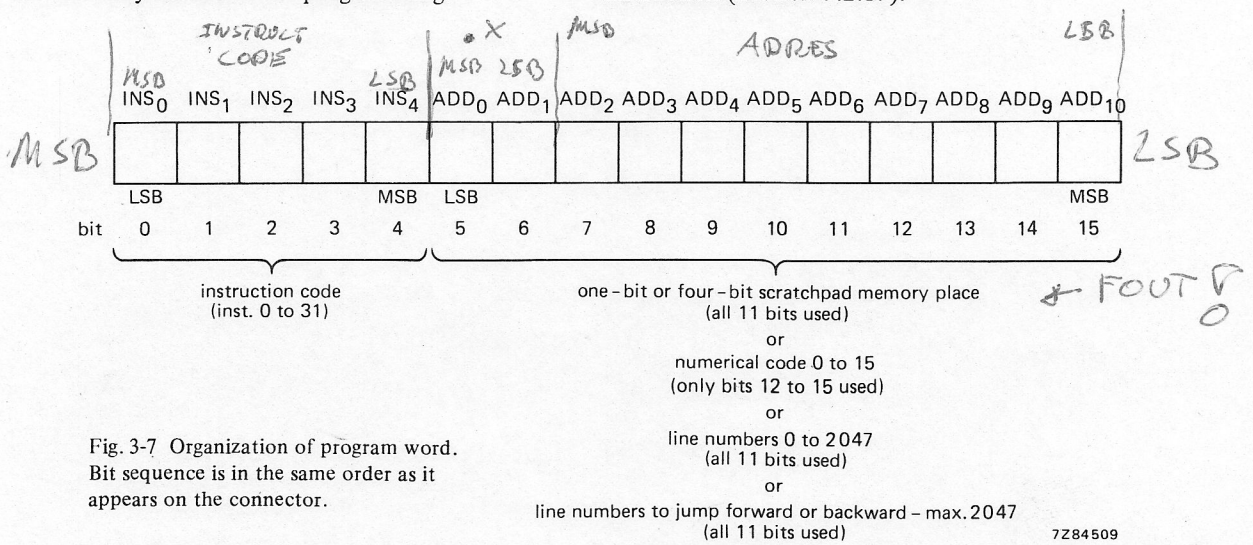



Fig. 3-7 Organization of program word. Bit sequence is in the same order as it appears on the connector.

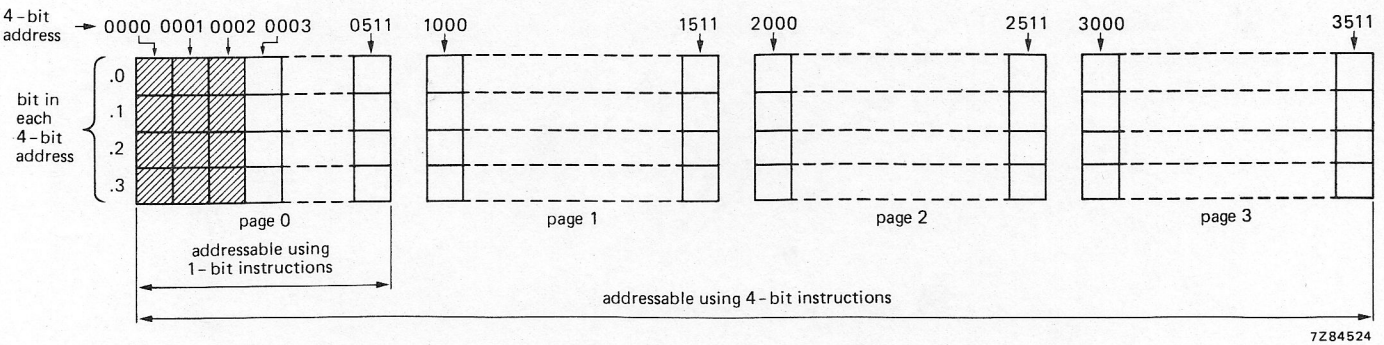
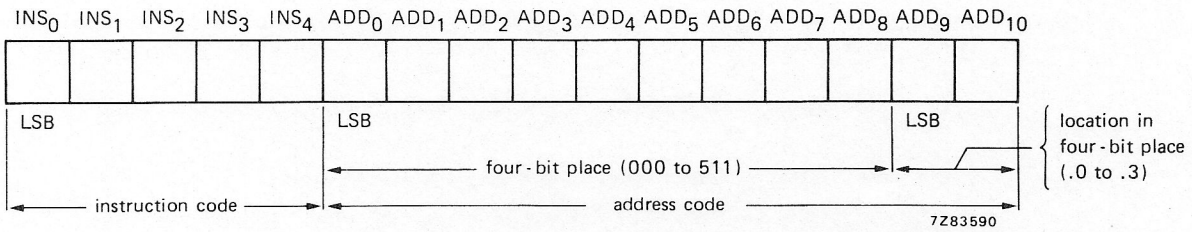



Fig. 3-8 Organization of 2k4 scratchpad memory in CP22 central processor. Scratchpad addresses 0000 to 0002 (hatched) are reserved for special purposes.

### 3.2.1 The AND, AND NOT, OR and OR NOT logic instructions

| denomination | inst. no. | instruction code | | | | | mnemonic | condition for fulfilment |
|---|---|---|---|---|---|---|---|---|
| | | $INS_0$ (LSB) | $INS_1$ | $INS_2$ | $INS_3$ | $INS_4$ (MSB) | | |
| AND | 16 | 0 | 0 | 0 | 0 | 1 | AND | 0 or 1 |
| AND NOT | 17 | 1 | 0 | 0 | 0 | 1 | ANDNT | 0 or 1 |
| OR | 18 | 0 | 1 | 0 | 0 | 1 | OR | 0 or 1 |
| OR NOT | 19 | 1 | 1 | 0 | 0 | 1 | ORNT | 0 or 1 |




The (one-bit) logic instructions are used to formulate the condition with which the instruction block starts. *To this end, the instruction block must open with an AND, AND NOT, OR or OR NOT logic instruction.* To be understood by the central processor, the condition must be described as a *sum of products.* The string of logic instructions defining the condition can have an arbitrary length (theoretically limited by the size of the scratchpad or program memory). The logic instructions are unconditional, which means that they will always be carried out irrespective of the prevailing condition, '0' or '1'.

In this section the AND, AND NOT, OR and OR NOT logic instructions will be dealt with and an application example given. The program word describing the logic instruction consists of an instruction part to define the logic operation and an address part to specify the scratchpad place which will act as the source of logic information; see the above figure.

AND *Instruction no. 16. This instruction causes the data, level '0' or '1', stored on the addressed one-bit scratchpad place to be transferred to the logic analyser. The AND instruction has a dual function: (a) it opens the chain of logic instructions; (b) it causes the transferred data to be processed by the logic analyser in an AND operation with the data transferred in an immediately preceding logic instruction or with the result of one or more immediately preceding AND (NOT) operations.*

ANDNT *Instruction no. 17. This instruction is similar to the AND logic instruction, however, with the data from the scratchpad inverted.*

OR *Instruction no. 18. This instruction causes the data, level '0' or '1', stored on the addressed one-bit scratchpad location to be transferred to the logic analyser and to be processed with the data transferred in any subsequent AND (NOT) instruction(s).* As such the OR instruction can be used to open the chain of logic instructions. The OR logic instruction initiates a new term in the sum of products describing the condition. If, as a result of the foregoing term(s), the condition has assumed the value '1', the OR logic instruction will cause the condition to hold that value. This is true because the condition is expressed as a sum of products; if any one term (product) is '1', the condition will become '1' and cannot within the given instruction block, be reset to '0' unless a TRIGGER instruction is given (Section 3.2.4).

ORNT *Instruction no. 19. This instruction is similar to the OR instruction but here the complement of the scratchpad data will be processed.* What has been said about the OR instruction will also apply here.

One application example is given here to illustrate the use of the logic instructions discussed above. Note, there is no execute instruction to terminate the instruction block; this is indicated by the dotted lines.

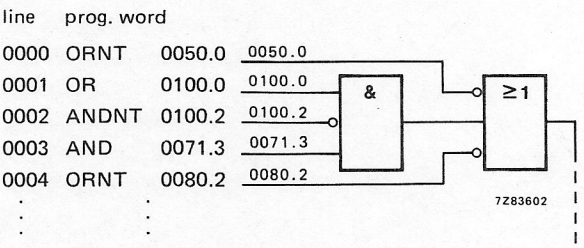
**Example 3-1 Use of the AND, ANDNT, OR and ORNT logic instructions.**

Starting at program line no. 0, the following condition must be programmed:

$$\overline{A} + B \cdot \overline{C} \cdot D + \overline{E} =$$

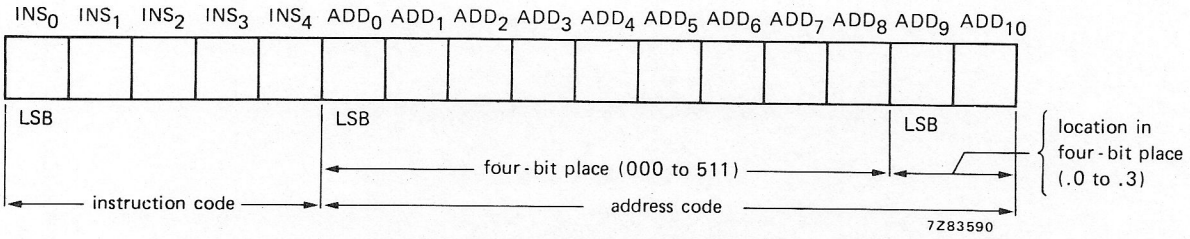
$$= \overline{(0050.0)} + (0100.0)\,\overline{(0100.2)}\,(0071.3) + \overline{(0080.2)}.$$

The program together with its equivalent hardware circuit is as follows:



### 3.2.2 The EQUALS and EQUALS NOT execute instructions

| denomination | inst. no. | instruction code | | | | | mnemonic | condition for fulfilment |
|---|---|---|---|---|---|---|---|---|
| | | $INS_0$ (LSB) | $INS_1$ | $INS_2$ | $INS_3$ | $INS_4$ (MSB) | | |
| EQUALS | 2 | 0 | 1 | 0 | 0 | 0 | EQL | 0 or 1 |
| EQUALS NOT | 3 | 1 | 1 | 0 | 0 | 0 | EQLNT | 0 or 1 |




In the PC20 instruction set the EQUALS and EQUALS NOT one-bit execute instructions are the simplest types. They are intended to put the value of the condition to immediate use in the program; this is further clarified in Example 3-2.

The program word describing the EQUALS or EQUALS NOT execute instruction (see the above figure) contains an instruction part defining the type of instruction and an address part specifying the scratchpad place where the value of the condition or its complement is to be stored. The EQUALS and EQUALS NOT execute instructions are unconditional, which implies that the contents of the allocated scratchpad place will be a replica of the condition.

EQL *Instruction no. 2. This instruction causes the value, 0 or 1, of the condition obtained from the preceding logic operation in the instruction block to be transferred to the addressed one-bit scratchpad place.*

EQLNT *Instruction no. 3. This instruction is similar to the EQL execute instruction, but now the complement of the condition will be transferred to the addressed one-bit scratchpad location.*

**Example 3-2 Use of the EQL and EQLNT execute instructions.**

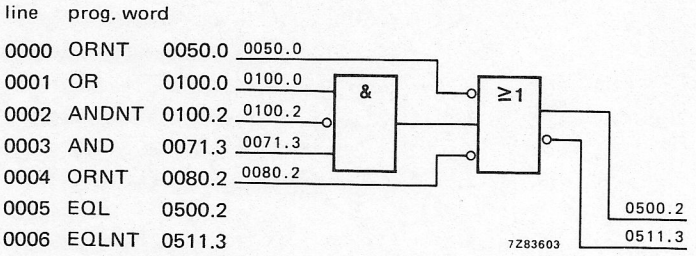
This example is an extension of Example 3-1 and will complete the instruction block. The following Boolean expression is programmed here:

$$\overline{A} + B \cdot \overline{C} \cdot D + \overline{E} = P = \overline{Q},$$

or taking scratchpad addresses:

$$(\overline{0050.0}) + (0100.0)\,(\overline{0100.2})\,(0071.3) + (\overline{0080.2}) =$$
$$= (0500.2) = (\overline{0511.3}).$$

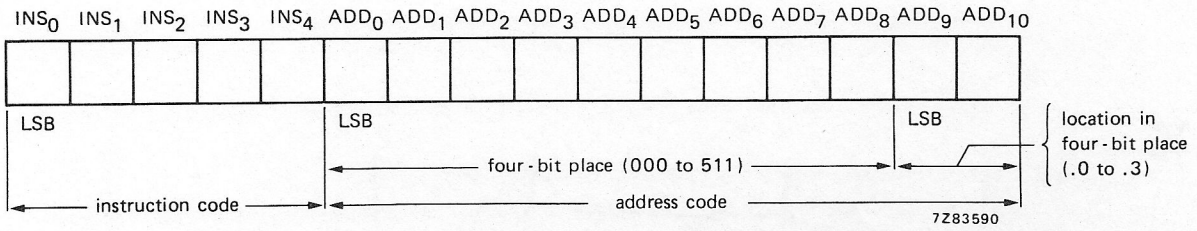
The program and its equivalent hardware circuit are as shown below.




It is clear that the condition $\overline{A} + B \cdot \overline{C} \cdot D + \overline{E}$ transferred to scratchpad address 0500.2 as well as its inverse value stored on scratchpad address 0511.3 are immediately available here for further use in the program. Note that scratchpad address 0511.3 is the highest addressable one-bit location in the scratchpad (CP22 central processor); see Fig. 3-8, Section 3-2.

### 3.2.3 The SET 1 and SET 0 execute instructions

| denomination | inst. no. | instruction code | | | | | mnemonic | condition for fulfilment |
|---|---|---|---|---|---|---|---|---|
| | | $INS_0$ (LSB) | $INS_1$ | $INS_2$ | $INS_3$ | $INS_4$ (MSB) | | |
| SET | 9 | 1 | 0 | 0 | 1 | 0 | SET 1 | 1 |
| RESET | 8 | 0 | 0 | 0 | 1 | 0 | SET 0 | 1 |




The SET 1 (SET) and SET 0 (RESET) one-bit execute instructions have a memory function as will be clear from what follows. The program word describing these instructions (see the figure) contains an instruction part defining the type of execute instruction and an address part specifying the scratchpad place where the result of the operation is to be stored. As the table shows, the SET 1 and SET 0 execute instructions become effective on condition '1'.

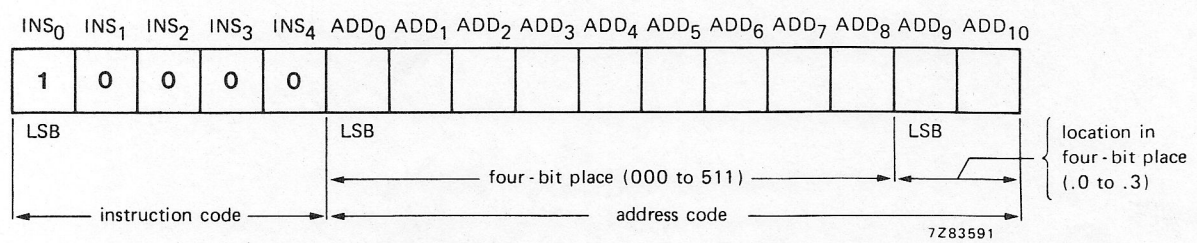
SET 1 *Instruction no. 9. This instruction has a SET function as it will cause bit level '1' to be stored on the addressed scratchpad place when the condition is '1'.*

SET 0 *Instruction no. 8. This instruction has a RESET function as it will cause bit level '0' to be stored on the addressed scratchpad place when the condition is '1'.*

Whereas in the case of the EQL and EQLNT execute instructions the bit level stored on the chosen scratchpad place can change from '0' to '1' and vice versa in order to follow the condition, it will change in one direction only for the execute instructions explained here, i.e. from '0' to '1' for the SET 1 instruction and from '1' to '0' for the SET 0 instruction. These execute instructions are, therefore, extremely well suited for building software flip-flops. A typical application is given in Example 3-3 dealing with the TRIGGER instruction.

### 3.2.4 The TRIGGER instruction

| denomination | instr. no. | instruction code | | | | | mnemonic | condition for fulfilment |
|---|---|---|---|---|---|---|---|---|
| | | $INS_0$ (LSB) | $INS_1$ | $INS_2$ | $INS_3$ | $INS_4$ (MSB) | | |
| TRIGGER | 1 | 1 | 0 | 0 | 0 | 0 | TRIG | 0 or 1 |



The TRIGGER instruction is the fifth in the group of (one-bit) logic instructions and the most difficult to understand. It is intended to detect a positive change in the value of the Boolean expression described by the preceding logic instruction(s) in the instruction block. Each PC20 cycle, when the TRIGGER instruction is met, the logic analyser compares the new value of the Boolean expression with the previous value originating in the foregoing PC20 cycle.

For comparison, the previous value of the Boolean expression is retained at an auxiliary scratchpad place as specified in the program word describing the TRIGGER instruction (see the above figure). When this instruction is met in the program, the contents of the auxiliary scratchpad place will be transferred to the logic analyser for comparison with the new value of the Boolean expression. *After comparison the new value will be written in the auxiliary scratchpad place and retained for comparison in the next PC20 cycle.*

Table 3-4 shows the value of the condition resulting from the comparison. It is seen that the condition is '1' when a positive change is detected; in all other cases it will be '0'.

*TABLE 3-4 Truth table for TRIG logic instruction.*

| value of Boolean expression preceding TRIG instruction | | condition as a result of TRIG instruction |
|---|---|---|
| previous PC20 cycle | present PC20 cycle | |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 0 |

The following simple program part is illustrative.

```
line   prog. word
 :       :
0100  AND   0012.1  ┐
0101  TRIG  0100.3  ├ Instruction block
0102  EQL   0016.0  ┘
 :       :
```

The Boolean expression is described here in the logic instruction "AND 0012.1". In terms of process control: the condition at the beginning of the instruction block is formed by the process function at input 0012.1. To find a positive change, the value of the process function is stored for comparison at auxiliary address 0100.3 chosen by the TRIG instruction. The value of the condition as a result of this instruction is transferred by the EQL execute instruction to output address 0016.0.

Figure 3-9 further illustrates the effect of changes in input 0012.1. While the input is at '0' level, the previous value of the input function existing at instant $T_0$ on auxiliary address 0100.3 and the new value entered at instant $T_1$ after comparison by the logic analyser are both '0'. As a result, the condition will remain at '0' level (see the above truth table); the new value will be retained in the auxiliary address for comparison in the next PC20 cycle. At the end of PC20 cycle n – 1, input 0012.1 will assume '1' level. So during the n-th cycle, while the previous value at the auxiliary address 0100.3 is still '0' at instant $T_2$, the new value entered at instant $T_3$ (after comparison) will become '1'. Because the logic analyser detects a positive change here, from '0' to '1', the condition which has been set to '1' by the "AND 0012.1" logic instruction will remain at that level. As a result, output 0016.1 will go HIGH to follow the condition. During the next PC20 cycle, denoted n + 1, the previous value (instant $T_4$) and the new value entered (instant $T_5$) are both '1'. As a change in input has not occurred, the condition will be reset to '0' and the output will accordingly go LOW again. Finally, in the PC20 cycle marked n + 2 the previous value (instant $T_6$) and the new value written in (instant $T_7$) are '1' and '0', respectively (input returned to '0' level). Consequently, the condition will be held at '0' level.

It is seen from the waveforms of Fig. 3-9 that the contents of auxiliary scratchpad place 0100.3 follow input 0012.1 whereas output 0016.0 assumes a changed state for only one PC20 cycle after the positive input change (one-shot function).

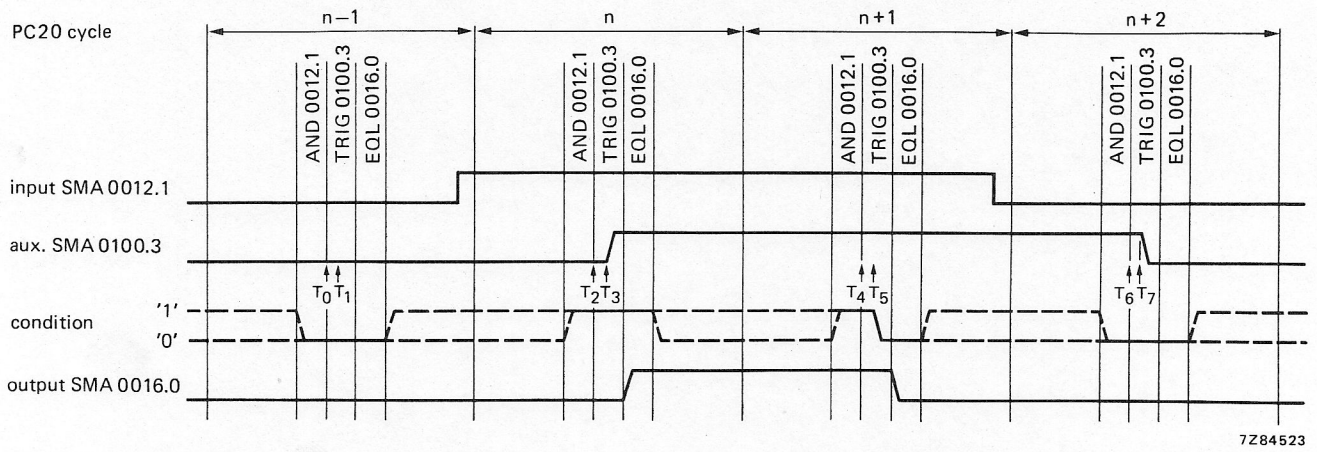



Fig. 3-9 Time diagram explaining TRIG instruction. At times $T_0$, $T_2$, $T_4$ and $T_6$, previous value of Boolean expression – here (0012.1) – still exists on SMA 0100.3; at times $T_1$, $T_3$, $T_5$ and $T_7$, new value is entered after comparison by logic analyser. Dashed waveform parts indicate that condition depends on program (value '0' or '1'). Contents of auxiliary address follows input; output follows the condition as a result of TRIG instruction and stays HIGH (condition '1') for only one PC20 cycle.

Obviously, from the foregoing account, the condition can become '0' even though the value of the Boolean expression specified in the instruction block is '1'. This expression is formulated as a sum of products. So, if any one term in the Boolean expression is '1', the condition will become '1' and cannot be reset to '0' level unless the TRIG instruction is met.

The following definition summarizes the TRIGGER function.

TRIG *Instruction no. 1. This instruction is used to detect a positive change in the outcome of its preceding logic instruction(s). It compares the outcome, in a given PC20 cycle, of the logic instruction(s) with the outcome, in the foregoing PC20 cycle, of the same logic instruction(s). For comparison, the outcome is stored on an auxiliary one-bit place addressed by the program word of the TRIG instruction* (see the figure at the head of this section). *Condition '1' will result whenever the new outcome shows a positive change with respect to the previous outcome.*

The TRIG instruction may be followed by: (1) one or more execute instructions, (2) one or more logic instructions terminated by one or several execute instructions.

Execute instructions following the TRIG instruction, which become effective on condition '1', will be obeyed once only, namely as a result of a positive change in input. This feature is used for counting and shift operations; see the examples given in Sections 3.2.9 and 3.2.10. Another application is where an arithmetic operation must be executed only on a positive-going input (Example 3-8, Section 3.2.7). The JUMP FORWARD RELATIVE instruction is executed on a false ('0') input condition (Section 3.2.13). So when placed after a TRIG instruction, it will always be active except when a positive input change occurs (condition becoming '1'). This is useful where a program section (contained within the relative jump) must be executed once only, such as loading a timer; examples are given in Chapters 5 and 6. The example in Section 3.2.13 illustrates the use of the JUMP FORWARD RELATIVE instruction in a D flip-flop.

The value of the condition generated when the TRIG instruction has been carried out is coupled in an *AND operation* to the value of a new Boolean expression described by any logic instructions following the TRIG instruction. If the value of the condition due to the TRIG instruction is called $C_T$, the following applies:

$C_T \cdot$(Boolean expression).

If, for instance, the new Boolean expression is $A \cdot B + C \cdot D \cdot E$ this will take the form:

$C_T \cdot (A \cdot B + C \cdot D \cdot E)$.

So, if the value of that Boolean expression is '1', the value of $C_T$, '0' or '1', will be maintained. Where the value of the Boolean expression is '0', the condition will also become '0' regardless of the value of $C_T$.

In Example 3-3 the TRIG instruction is followed by a logic instruction.

**Example 3-3 Software edge-triggered D flip-flop.**

| line | prog. word | | | |
|---|---|---|---|---|
| 0041 | AND | 0124.2 | When clock input 0124.2 goes HIGH | inst. block n. |
| 0042 | TRIG | 0075.0 | | |
| 0043 | AND | 0124.1 | and data input 0124.1 is HIGH | |
| 0044 | SET 1 | 0100.2 | output 0100.2 will become HIGH | |
| 0045 | AND | 0124.2 | When clock input 0124.2 goes HIGH | inst. block n + 1. |
| 0046 | TRIG | 0075.1 | | |
| 0047 | ANDNT | 0124.1 | and data input 0124.1 is LOW | |
| 0048 | SET 0 | 0100.2 | output 0100.2 will become LOW. | |

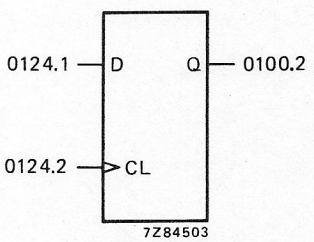



Fig. 3-10 Example 3-3: Software edge-triggered D flip-flop.

Two instruction blocks are used here to describe the software flip-flop (Fig. 3-10). In instruction block n, the output 0100.2 will be set when the clock input 0124.2 goes HIGH and the data input 0124.1 is HIGH. On the other hand, the output will be reset in instruction block n + 1 when the clock input goes HIGH while the data input is LOW; see the waveforms of Fig. 3-11.