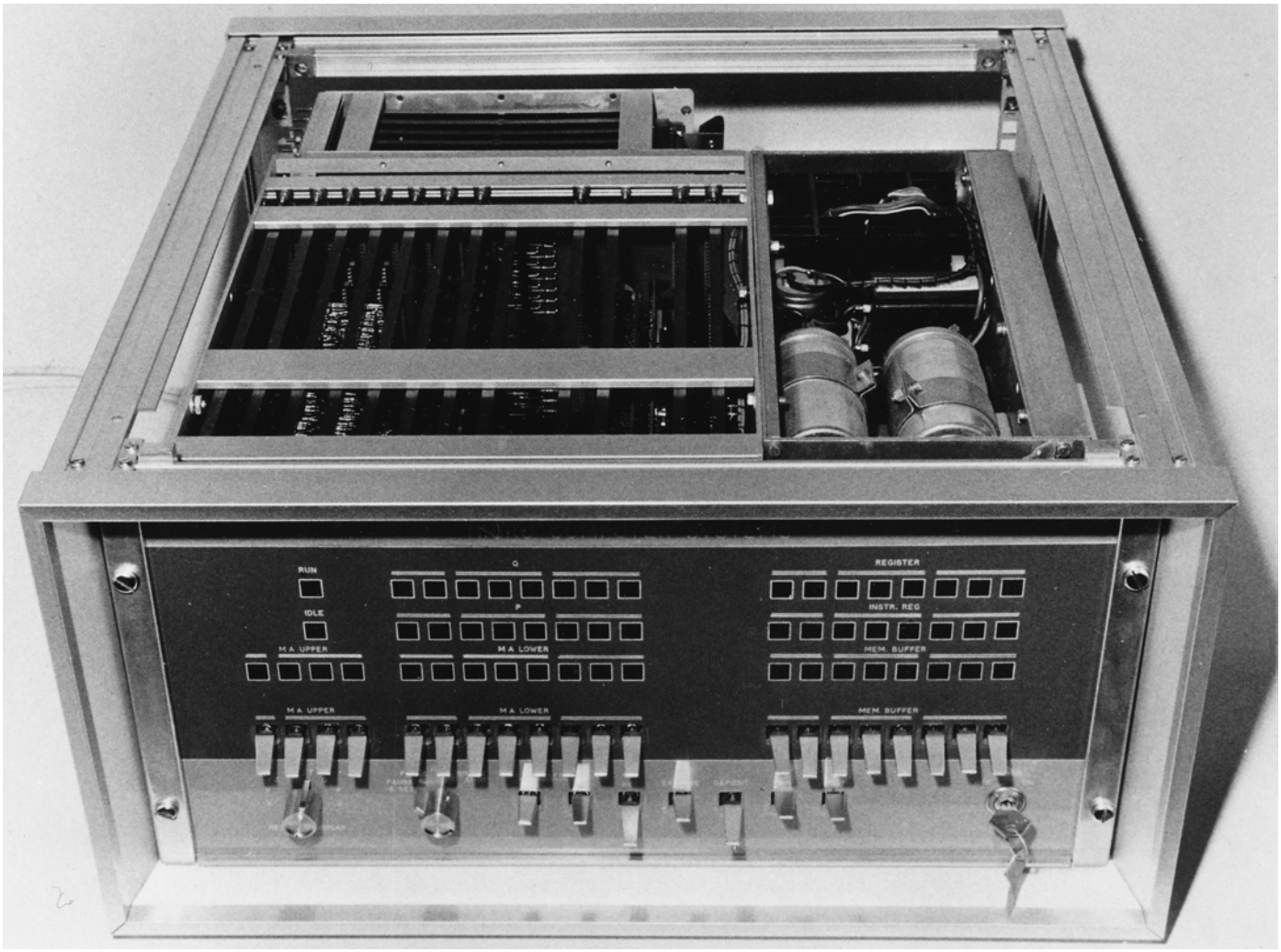
October 1969

# M I N I C  I

Man-Machine Systems Laboratory

Department of Electrical Engineering Science,

University of Essex.




MINIC I Prototype developed at University of Essex 1968-1969

Summary

MINIC is a modular, microprogrammed minicomputer based on a set of functional blocks which may be plugged into standard hiway racking to form a range of specialised and general-purpose computers, controllers and data-communication systems.

MINIC I consists of a set of MINIC modules which are microprogrammed to behave as a general-purpose digital computer. This machine is byte-orientated and the majority of instructions are 8 bits in length and operate on one or more 8-bit "bytes" of data.

This machine includes eight programmer-visible 8-bit registers used to hold operands and program and data pointers.

The instruction set of MINIC I enables programs to be executed and data to be referenced in up to 65,536 words of core store. Operations available include byte multiplication and division, half-word operations for decimal arithmetic, and stack manipulation of the operand registers. A change environment instruction automatically saves the eight programmer-visible registers on entering a new program, and may be executed by an external device on the multi-level priority interrupt system. The 8-bit input/output transfer instruction enables sixteen device channels to be directly addressed, two 8-bit data or command words to be sent out, one 8-bit data word to be received, and a skip of up to 255 program steps to be executed, within a simple instruction cycle.

MINIC I is intended as a base-level machine, general-purpose and powerful in its own right, but capable of extension through the addition of further microprogrammed routines for specialised functions. Facilities are included for additional routines to be activated either through the program, or through an external device. Thus the machine may be specialised to act most effectively as a controller, signal processor, data-communication system, or similar system, whilst retaining a standard general-purpose instruction set.

This document is intended as a general introduction and users' handbook for a MINIC computer with the MINIC I microprogram.

6.3 Rotate Instructions

6.3.1 Rotate X left (ITR 16, RXL)
6.3.2 Rotate X right (ITR 17, RXR)
6.3.3 Rotate double register XY left (ITR 20, RDL)
6.3.4 Rotate double register XY right (ITR 21, RDR)

6.4 Half-word Instructions

6.4.1 Exchange half words of X (ITR 27, EHX)
6.4.2 Unpack X and Y (ITR 31, UNP)

6.5 Sign Operations

6.5.1 Zero to sign of X (ITR 24, ZSX)
6.5.2 Unity to sign of X (ITR 25, USX)
6.5.3 Copy sign of X to all of X (ITR 26, ASX)

6.6 Push-down Operations

6.6.1 Push-down retaining X (ITR 22, PDX)
6.6.2 Push-down and clear X (ITR 23, CLX)

6.7 Logical Operations

6.7.1 Complement X (ITR 30, CMX)
6.7.2 AND Y to Z (ITR 32, AND)
6.7.3 EXCLUSIVE-OR Y to X (ITR 33, EOR)

6.8 Arithmetic Operations

6.8.1 Add X to YZ and circulate up (ITR 34, ADD)
6.8.2 Subtract X from Y and circulate up (ITR 35, SUB)
6.8.3 Multiply X by Y and add to Z (ITR 36, MUL)
6.8.4 Divide ZY by X (ITR 37, DIV)

## 7. Compound Instructions

7.1 Load complement of literal to X (COM 0, LCX)

7.2 Double Unpack (COM 1, DUN)

7.3 Double AND (COM 2, DAN)

7.4 Double EXCLUSIVE-OR (COM 3, DEO)

7.5 Double Add (COM 4, DAD)

7.6 Double Subtract (COM 5, DSU)

7.7 Double Multiply (COM 5, DMU)

7.8 Double Divide (COM 7, DDI)

7.9 Load literal in E (COM 10, LDE)

7.10 Load literal in F (COM 11, LDF)

7.11 Load literal in G (COM 12, LDG)

7.12 AND X to literal and skip if zero (COM 13, ASX)

7.13 Increment in program page and skip if result is zero (COM 13, ISP)

7.14 Increment in data page and skip if result is zero (COM 14, ISD)

7.15 Extended instruction set (COM 16, EXT)

7.16 Change environment (COM 17, ENV)

# MINIC SPECIFICATION

## MINIC I INSTRUCTION SET

### 1. Introduction

MINIC I is a byte-orientated machine in which an 8-bit "byte" is both the unit of data and of programming. The majority of instructions in MINIC I are single-length and occupy one byte. All data operations which can be specified operate on one or two bytes and produce a one or two byte result.

The range of instructions within the exceptionally short 8-bit word-length of MINIC is made possible by the use of hardware indexing and indirect addressing techniques for access to structured data, together with direct access to a small workspace. Double-length instructions are also made available for special cases not covered by the standard address techniques.

The data-structure addressing techniques are such that any arithmetic or logical operation on single byte words can be extended to multiple byte words up to 256 bytes. Thus the machine may be thought of either as a single-byte machine with an exceptionally compact instruction set, or a variable word-length machine with an extremely flexible instruction set.

The power of MINIC I depends on the large number of hardware registers accessible to the programmer together with its sophisticated micro-programmed instruction set. The availability of a number of hardware registers is matched by special instructions for loading the registers and operating on them, such as the EXCHANGE ENVIRONMENT, instruction for moving between programs.

The following sections describe the programmer-accessible registers of MINIC I, the instruction set of the machine and its input/output operations.

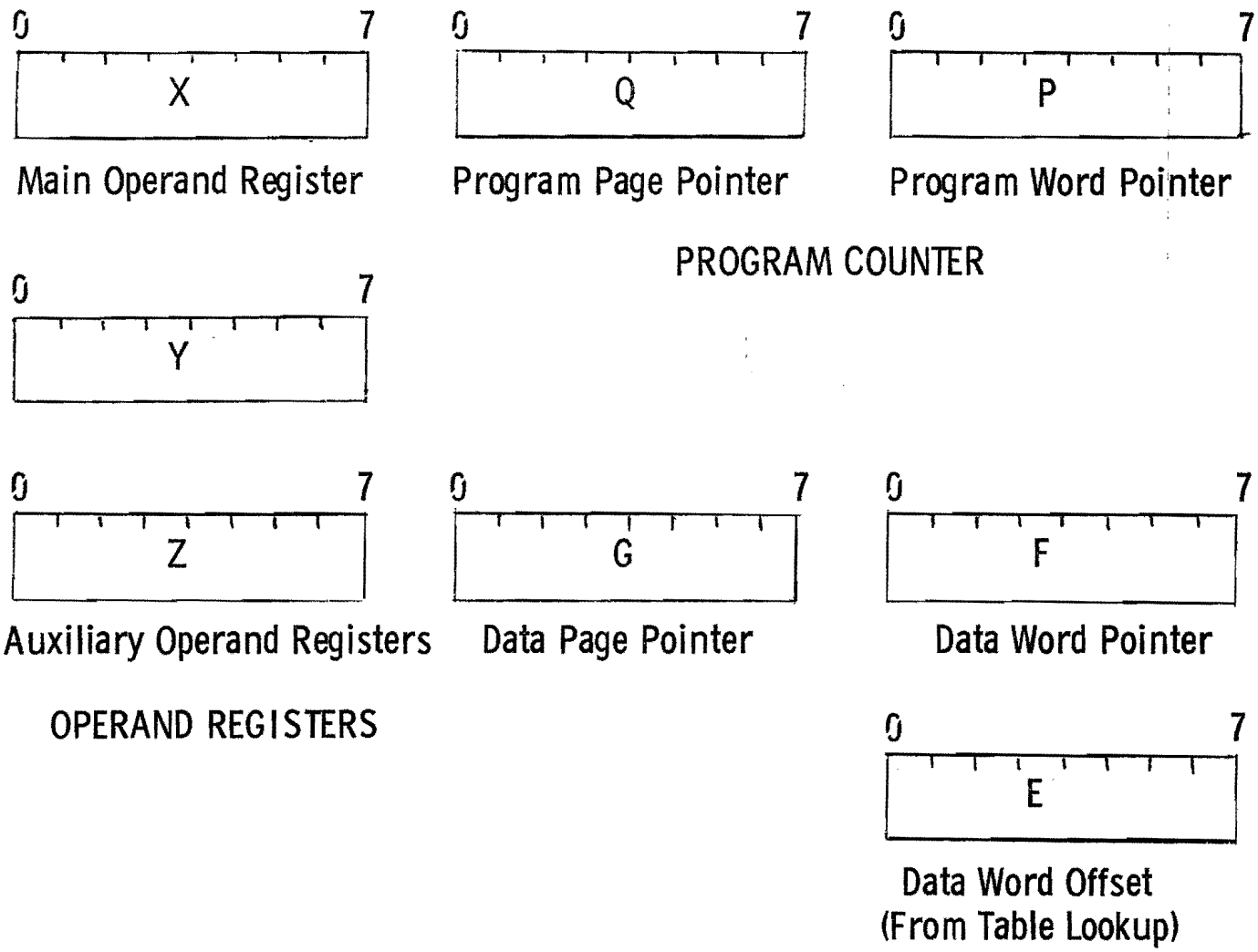



Figure 1 Registers in MINIC 1

## 2.5 The Core Store

With its capability of generating 16-bit addresses (as two bytes), the MINIC I instruction set is able to control a 65,536 word core-store. Since 4-bit, 8-bit, 12-bit (through "double word" load/store and jump instructions) and 16-bit (through use of hardware registers) addresses may be generated, it is convenient to divide the core into:

"blocks" of 16 words

"pages" of 16 blocks, 256 words

"chapters" of of 16 pages, 256 blocks, 4096 words

The maximum size of core store is then 16 chapters, 256 pages, 4096 blocks 65,536 words.

Since the double-length JUMP and JUMP to SUBROUTINE instructions (Section 5) generate a twelve-bit address which, together with the upper four bits of Q, gives a 16-bit instruction pointer, it is convenient to think of these four bits of Q pointing to the current program chapter. Similarly the double-length load/store instructions concatenate a 12-bit address with the upper four bits of G which point to the current data chapter.

The 256-word current program page pointed at by Q, and the 256-word current data page pointed at by G, are also natural units of storage in MINIC I, since operations incrementing or adding to P, F and E do not propagate carries into Q or G, and hence the program and data pages are completely "circular". Thus, separate pages are not functionally related, and contiguity of pages in either program or data chapters has no effect.

The zeroth block of the current program space is used as workspace, and the zeroth block of the current data page is used as a table of indirect addresses to that page. The block structure of MINIC is the only sub-structure not fixed to power-of-two boundaries since in the JUMP RELATIVE and LOAD/STORE INDEXED instructions carries are propagated from the lower

## 3.1. Instruction Formats and Mnenomics for Assembly Languages

Since each instruction in MINIC I is either a single 8-bit word, or two 8-bit words stored in consecutive locations the basic representation of a MINIC program is a succession of 8-bit numbers. It is convenient to represent these in octal notation separated by appropriate delimiters, and translate these to 8-bit words by an assembly program. Three alternative delimiters are accepted by the assemblers: carriage return, newline or semicolon (;). Line feeds and spaces are ignored so that a legible layout may be obtained. The assemblers convert the octal number to binary taking the 8 least significant bits of the result as the instruction. This is the zeroth-level representation of programs for MINIC I.

This basic representation is conveniently extended since, as shown in Table 1, most instructions split naturally into two 4-bit fields, the first field denoting the instruction type, and the second field having a variety of interpretations. To take advantage of this, the assemblers may also accept a two-component instruction format, consisting of two octal numbers separated by a colon. The assemblers multiply the first component by 20 (octal) and add the second, regarding the result as a octal number and converting to binary as previously described. If no colon is present then the first component is assumed to be absent, thus preserving compatibility with the zeroth level representation. Because of the mode of interpretation of this instruction, the second component can represent a 4-bit, 5-bit or greater length, field, and hence the RELATIVE JUMP and INTER-REGISTER instructions are conveniently represented. This is the first-level representation of programs for MINIC I.

The previous formats are conveniently extended by allowing the octal number of a single-component instruction, or the first component of a two-component instruction, to be represented by symbols. Three-letter mnenomic symbols are used to represent the standard instruction set of MINIC I. These are translated into octal numbers by the assemblers, and then the previously described stages of assembly are entered. Because the symbols

The LOAD instruction causes the contents of the memory location addressed to be copied into register X, the previous contents of X to be loaded into register Y, and the previous contents of Y to be loaded into register Z; that is, the stack is "pushed down" and the previous contents of Z are lost. The contents of the memory location addressed remain unchanged.

The STORE instruction causes the contents of register X to be loaded into the memory location addressed. The previous contents of register Y are loaded into X; the previous contents of register Z are loaded in Y; and the previous contents of register X are loaded into Z. That is, the stack "pops up" and the contents of X circulate into Z. This circulation is not a normal stack operation, but is convenient in data handling, and is used in many inter-register operations.

### 4.2 Address Computation

There are four modes of address computation determined by the first two bits of the address designated by the last six bits of the instruction:

#### 4.2.1 Load and Store in Workspace (LDW and STW)

The first 16 words of the 256-word active program page pointed to by register Q are allocated as "workspace" and may be addressed directly in the workspace mode by the last four bits of instruction word.

#### 4.2.2 Load and Store Double (LDD and STD)

In the double-length instruction mode the word following the instruction in the store is regarded as an 8-bit extension of the address denoted by the last four bits of the instruction, giving an effective 12-bit address directly addressing any of the 4096 words in a "chapter" of 16 x 256 word pages; this is the maximum size of core directly addressable in MINIC I. The P register is incremented by one when a double-length load/store is executed so that the extension word is skipped over.

bc = 10: Indexed: address word j relative to word pointed at by F in current data page

bc = 11: Table: address word pointed at by word j in block zero current data page relative to E.

The mnenomic forms of the load and store instructions as given in Table 1. They consist of the two letters, LD or ST for load and store respectively, followed by the single letter, W, D, I or T, for workspace, double, indexed or table respectively.

## 5. The JUMP Instructions

There are three modes of instruction for changing the sequence of a program through a "jump":

### 5.1 Relative Jump (JMR)

In a relative jump the last five bits of the instruction are regarded as a signed twos-complement number in the range (-16, +15) and are added to contents of register P. Since P, at that time is pointing to the next instruction, this enables a relative jump to be made of up to 16 steps forward and 15 steps backward. No carry is propagated from P to Q so that the current program page cannot be changed by a relative jump and is regarded as circular.

### 5.2 Double Jump (JMD)

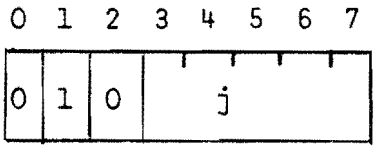
In a double-length jump the last four bits of the instruction word are loaded into the lower half of the Q register to select the new active program page and the word following the instruction is loaded into the P register to point to the next instruction in that page. Thus a transfer to any one of the 4096 words in the current program chapter may be made through a double-length jump.

### 5.3 Subroutine Jump (JMS)

A subroutine jump is identical in its mode of operation to a double jump except that a pointer to the return address, in the form of a double

## 6. Inter-Register Instructions

There are 32 inter-register instructions in MINIC I, involving no calls in the memory and performing interchange, arithmetic and logical operations, and so on. The format of the inter-register instructions is:




Inter-register instruction

From Table 1, it will be seen that the two-component representation of these instructions is either:

04:j or
ITR:j

where j is an octal number in the range 0 through 37 denoting the instruction. Since each instruction is different and distinctive in operation, they are also each given an individual 3-letter mnemonic which forms a single component (see second example, Section 3.2); Table 2 gives a complete listing of all inter-register instructions and mnemonics. The instruction split naturally into sub-sets of a similar type, and the exact operation of each instruction within a sub-set is described in the following sections:


6.1 Skip Instructions.
6.2 Register Exchanges and Circulation.
6.3 Rotate Instructions.
6.4 Half-Word Instructions.
6.5 Sign Operations.
6.6 Push-down Operations.
6.7 Logical Operations.
6.8 Arithmetic Operations.

| ITR Octal | Mnenomic | Effect |
|---|---|---|
| 33 | EOR | EXCLUSIVE -OR Y to X |
| 34 | ADD | Add X to ZY and circulate up |
| 35 | SUB | Subtract X from ZY and circulate up |
| 36 | MUL | Multiply X by Y and add to Z |
| 37 | DIV | Divide ZY by X |

## 6.1. Skip Instruction

The skip instructions are used to change the sequence of a program according to the state of operands. They cause P to be incremented by 1 to skip over the following instruction if a certain condition is satisfied, otherwise the next sequential instruction is executed; no carries are propagated from P. Note that the following instruction should be single-length if it is to be skipped over correctly. Skips may be combined with increment operations on registers E and F for use in loop testing and iteration through data structures.

### 6.1.1. Skip if register X contains zero (ITR 0, SZO)

P is incremented by 1 if and only if the contents of X are zero.

### 6.1.2. Skip if register X does not contain zero (ITR 1, SNZ)

P is incremented by 1 if and only if the contents of X are not zero.

### 6.1.3. Skip if X is positive (ITR 2,SPO)

P is incremented by 1 if and only if the most significant bit of X, $X_0$, is a zero. If X contains a number in twos-complement notation, this implies that it is positive or zero.

### 6.2.3. Exchange X and F (ITR 6, EXF)

The contents of X are placed in F and the previous contents of F are placed in X.

### 6.2.4. Exchange X and G (ITR 7, EXG)

The contents of X are placed in G and the previous contents of G are placed in X.

### 6.2.5. Exchange F and E (ITR 14, EFE)

The contents of F are placed in E and the previous contents of E are placed in F. This instruction is used to enable two E-type, or two F-type, pointers to be kept in E and F and interchanged as required. It is used typically in indexing through two data structures at different rates.

### 6.2.6. Exchange F and G (ITR 12, EFG)

The contents of F are placed in G and the previous contents of G are placed in F. This instruction is used to access one of two data pages, the active one being pointed to by G, the inactive one by F, and the exchange operation being used to interchange them. It is used typically in the interchange of data between pages.

### 6.2.7. Circulate up (ITR 10, CUP)

This instruction causes the XYZ stack to pop up and the contents of X to circulate round into Z. Its effect is that the contents of Z are placed in Y, the previous contents of Y are placed in X, and the previous contents of X are placed in Z.

one bit, so that bits of both words are shifted right one position, the least significant bit of X circulates into the most significant position of Y, and the least significant bit of Y circulates into the most significant position of X.

## 6.4. Half-word Instructions

Two instructions enable the operand registers of MINIC to be used as two 4-bit registers rather than a single 8-bit register. These are used mainly in decimal arithmetic, binary/decimal conversion, and operations on character codes.

### 6.4.1. Exchange half words of X (ITR 27, EHX)

The first four bits of X are interchanged with the last four bits of X.

### 6.4.2. Unpack X and Y (ITR 31, UNP)

The last four bits of X are interchanged with the first four bits of Y. This operation enables a two-digit BCD number in Y to be split in half, and the two halves to be placed in the lower four bits of X and Y respectively. If X initially contains zero, a literal multiply by 10 (decimal) following the unpack converts the BCD number to binary. If X initially contains 273 (octal) the two-digit BCD number is converted to two ASCII characters in X and Y. Equally, two ASCII characters may be packed by the same operation into two-digit BCD, and a binary number may be converted to the same form by division followed by "unpack".

## 6.5. Sign Operations

In signed arithmetic, the sign of an operand is normally held in the most significant bit of its most significant byte. The MINIC I instruction set contains three operations relating to the most significant bit of X, enabling it to be set, reset and copied into the whole of X.

### 6.7.1. Complement X (ITR 30, CMX)

The logical complement of the operand in X is placed in X, so that each bit of X that is zero becomes one and vice-versa.

### 6.7.2. AND Y to X (ITR 32, AND)

The logical AND function of the contents of X and Y is formed and placed in X. Each bit of X becomes a one if and only if the corresponding bits of X and Y were previously both ones.

### 6.7.3. EXCLUSIVE-OR Y to X (ITR 33, EOR)

The logical EXCLUSIVE-OR function of the contents of X and Y is formed and placed in X. Each bit of X becomes a one if and only if the corresponding bits of X and Y were previously not equal.

## 6.8. Arithmetic Operations

The four arithmetic operation of addition, subtraction, multiplication, and division are available in MINIC I. Each involves one double-length and one single-length operand, and hence uses all three operand registers. The operands are taken to be unsigned integers, but equivalent operations on signed numbers may be performed by using twos-complement or sign-and-magnitude representations.

### 6.8.1. Add X to YZ and circulate up (ITR 34, ADD)

The unsigned 8-bit integer in X is added to the unsigned 16-bit integer whose least significant half is in Y and whose most significant half is in Z (no carry is propagated out of Z). The result is circulated up so that the least significant half is in X, the most significant half is in Y, and Z contains the original contents of X.

### 6.8.2. Subtract X from Y and circulate up (ITR 35, SUB)

The unsigned 8-bit integer in X is subtracted from the unsigned 16-bit integer whose least significant byte is in Y and whose most significant byte is in Z (no borrow is propagated out of Z). The result is circulated up so that the least significant byte is in X, the most significant byte is in Y, and Z contains the original contents of X.

Where j is an octal number in the range 0 through 17 denoting the instruction. As with the inter-register instructions, each compound instruction is given an individual 3-letter mnenomic which forms a single component preceeding the literal double-word extension.

Seven of the compound instructions are literal argument forms of the inter-register, add, subtract, multiply, divide, and, exclusive-or, unpack, and are equivalent to "load X with the literal constant, pushing down the previous contents of X into Y, and the previous contents of Y into Z", followed by the inter-register instruction. The other compound instructions give literal loading of X, E, F and G, "increment-and-skip" operations to words in core, mask and skip operations to test bits of X, extensions of the instruction set, and a complete change of the 8-register current environment.

Table 3 lists the sixteen compound instructions, their octal formats, and mnenomics. The following sections describe the operation of the instructions in detail, except for the exchange environment instructions which are described separately in Section 9.

### 7.1. Load complement of literal to X (COM 0, LCX)

The 8-bit literal byte of the compound instruction is complemented and loaded into the X register. The previous contents of X are placed in Y, and the previous contents of Y are placed in Z; the previous contents of Z are lost. This instruction is intended to be used as a literal load of X, and the complementation is purely fortuitous, a convenience in the micro-program.

### 7.2. Double unpack (COM 1, DUN)

The 8-bit byte of the compound instruction is loaded into the X register, pushing down the previous contents of X into Y, and those of Y into Z. The lower half of the contents of X is then exchanged with the upper half of the contents of Y. This instruction is used for binary/decimal operations and character manipulation as described in Section 6.4.2, for example, the double length instruction:

```
DUN
273
```

expands a two-digit BCD number in X to two corresponding ASCII characters in X and Y.

### 7.3. Double AND (COM 2, DAN)

The 8-bit literal byte of the compound instruction is loaded into the X register, pushing down the contents of X into Y, and those of Y into Z. The logical AND function is taken between the contents of X and Y, and the result is placed in X.

### 7.4. Double EXCLUSIVE-OR (COM 3, DEO)

The 8-bit literal byte of the compound instruction is loaded into the X register, pushing down the previous content of X into Y, and those of Y into Z. The logical EXCLUSIVE-OR function is taken between the contents of X and Y, and the result is placed in X.

## 7.11. Load literal in G (COM 12, LDG)

The 8-bit literal byte of the compound instruction is loaded into register G.

## 7.12. AND X to literal and skip if zero (COM 13, ASX)

The logical AND function of the contents of the X register and the 8-bit literal byte of the compound instruction is formed. The contents of register P are incremented by 1 if the result is zero. Note that no registers except P are affected by this operation. The instruction is used to test whether bits of the word in X are set.

## 7.13. Increment in program page and skip if result is zero (COM 13, ISP)

The 8-bit literal byte is taken as a pointer to a word in the current program page, pointed at by Q. The contents of this memory location are incremented by 1, and the contents of P are also incremented by 1 if the result is zero.

## 7.14. Increment in data page and skip if result is zero (COM 14, ISD)

The 8-bit literal byte is taken as a pointer to a word in the current data page, pointed at by G. The contents of this memory location are incremented by 1, and the contents of P are also incremented by 1 if the result is zero.

## 7.15 Extended instruction set (COM 16, EXT)

This instruction performs no operation in the 128-step microprogram basic version of MINIC I. It is used to control extensions of the microprogram, for example, to drive a disc, synchronous line, or other high-speed peripheral. It may be used to implement families of special-purpose instructions, and may itself be extended to triple, or greater, length. This facility enables extensions of MINIC I to be downwards compatible with MINIC I and to use the full MINIC I program and subroutine library. Section 9 contains a specification of the operation of this instruction

## 8. Input/Output Instructions

The input-output system of MINIC I is hiway structured with separate 8-bit input and output data hiways and three control lines. The channel address is carried on the output hiway, and in a single-length input-output instruction 16 channels may be addressed. On execution of an IOT instruction, data may be output from the X and Y registers, data may be input to the X register, and any number of instructions may be conditionally skipped. The same input-output hiways are also used for the interrupt system, the key and lamp console, and for operation of fast peripherals, such as discs, under micro-program control. The input-output hiways are buffered extensions of internal hiways and operate synchronously at the main clock rate of the computer (nominally 2MHz).

### 8.1 Input and output hiways and control lines

The 8-bit input hiway, IN, of MINIC I consists of 8 lines which are normally high corresponding to zero data, and may be pulled low by external devices to signify one data. In an IOT instruction data coming in on this hiway may be read into register X, added to the program word pointer P to cause a skip, or used under interrupt control as the environment pointer to an EXCHANGE ENVIRONMENT compound instruction. The 8-bit output hiway, OUT, of MINIC I consists of 16 lines driven by internal bus drivers, 8 in the normal sense that high corresponds to one data, and low corresponds to zero data, and 8 in the inverse sense; the availability of both senses of output is particularly useful in device address decoding. In an IOT instruction data on the OUT bus may be a device channel address, or the contents of registers X or Y.

The three control lines in MINIC are MICRORUPT input, HESITATE input, and IO output. The MICRORUPT line is used to interrupt the instruction sequence of the main program, and is a single-wire hiway which is normally high. When the MICRORUPT is pulled low the current instruction is completed and, instead of fetching the next, the micro-program enters a sequence in which it addresses the interrupt system on OUT, inputs a word on IN, and executes an EXCHANGE ENVIRONMENT instruction using this word as the environment pointer. Input/output instructions cannot be interrupted in this way. The HESITATE line is another single-wire hiway used to freeze the microprogram for synchronization purposes when working with fast peripherals. The IO line

## 8.3 Device Control Logic

Figure 3 is a block diagram of the device control logic for a device which receives data or commands, causes a skip, and interchanges data, all in one IOT instruction cycle. Often only one or two of these operations will be performed, and only part of the device control logic will be required.

The "Device Address Detection Gates" effectively form a 9-way AND gate whose output is ON only when the IO line is ON and a prescribed 8-bit address is present on the OUT hiway. The output from this gate drives a 3-bit shift-register whose delayed outputs provide strobing levels corresponding to:

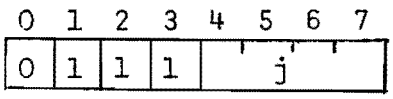
(1) Data available from X - the $READY_1$ line is activated to gate data or commands on the OUT bus into a device register.

(2) Increment program counter - a set of bus gates, the "skip generate gates", corresponding to the increment required, are driven to pull down the IN hiway according to the increment required. It is possible for a single device to have several sets of "skip generate gates" driven by different SKIP lines, according to conditions in the device.

(3) Data input to X - The $READY_2$ line is activated to gate data on the OUT bus into a device register and data from the device is gated onto the IN hiway through a set of bus gates.

## 8.4 Format of Input/Output Instructions

The format of the input/output instructions is:




Input/Output Instructions

where j is the channel number of the external device.

The whole instruction is actually transmitted as the channel address so that out of the 256 device channels, those from 160 (octal) through 177 (octal) may be addressed by an IOT instruction. The other channels are accessible only through the microprogram, for example in interrupt and halt operations, although the number of directly accessible channels may be increased by appropriate use of the EXTend compound instruction (Section 7.16). Normally these other input/output channels will be used with special peripheral control microprograms,such as disc interfaces, direct memory access channels, and so on.

## 9. Change Environment, Idle and Extended Instructions

The double-length CHANGE ENVIRONMENT instruction described in Section 7.16 is the software form of the response to an external interrupt in MINIC I. When the MICRORUPT input hiway is pulled low by some device on the input-output hiway, execution of the current instruction is completed and the word on the IN hiway is taken to be the double-word extension of a CHANGE ENVIRONMENT instruction. Execution of this instruction enables the computer to enter IDLE or PAUSE states, to change to a new program environment, or to enter a procedure in the microprogram extension, all under the control of the external device. The following sections describe in detail the effect of interrupts, the "change environment" operation, the action of the computer when it "pauses" or "idles", and the extension of the microprogram.

### 9.1 Interrupts

The MICRORUPT line may be pulled to ground at any time but no action is taken by the computer until execution of the current instruction is completed, or, if the instruction is an IOT, until execution of the following instruction is completed. At the completion of the current instruction cycle, the IO line comes on and the interrupt system device address (157) is put out on the OUT hiway. This signifies that the computer is ready to service the interrupt and is used to read-in the new environment specification, reset interrupt masks, and so on. At the end of the following clock interval the word on the IN hiway is read into the computer and a CHANGE ENVIRONMENT instruction is executed taking the word read in as the double-length extension of this instruction.

The interrupt system itself consists of flip-flops set by external devices to call for an interrupt, and associated flip-flops set up by the programmer through IOT instructions to enable or disable particular interrupt channels. The interrupt system hardware itself has priority logic built-in to prevent two channels putting in environment information at the same time. However, this effectively only allocates priority within an instruction cycle, and the main control over priority of interrupts is through the programmer-set enable/disable mask.

## 9.3 Microprogram Extension

When a change environment to 377 (binary change environment extension word 11 11 11 11) is executed or caused by an external interrupt, if an extension to the 128-word microprogram of the basic MINIC I is present, then control is transferred to the first word (location 200 (octal)) of the microprogram extension. If there is no extension to the microprogram then the effect of this instruction is the same as that of change environment to zero (i.e. computer enters PAUSE or IDLE states).

Similarly when the EXT instruction of Section 7.15 is executed, if a microprogram extension is present control is transferred to its second word (location 201 (octal)). Otherwise the effect of the instruction is that no operation takes place.

Either of these instructions may be used to enter procedures in the microprogram extension, and, in particular the change environment to 377 instruction may be used by a peripheral device, such as disc, or autonomous-transfer channel, to call peripheral control programs associated with the device.

(b) Data keys to set up an 8-bit word to be deposited in memory or in one of the 8 programmer-visible registers.

(c) Examine key to display in the memory data register the contents of the core location addressed by the memory address keys.

(d) Deposit key to store the setting of the data keys in the core location addressed by the memory address keys.

(e) Set key to store the setting of the data keys into a selected programmer-visible register.

(f) Run key to initiate execution of a stored program.

(g) Idle/pause key to stop execution of a stored program and display the contents of the hardware registers.

(h) Reset key to reset all internal hardware registers to zero.

(i) IO reset key to initialise all input/output devices.

(j) Register select switch to select which of the registers, X, Y, Z, E, F, G, P, Q, is displayed on the register display, and set into by depression of the set key.

(k) Mode select switch to select the single-instruction, idle, pause, or pause-and-set, modes of the computer. In the single-instruction mode, the computer is in the idle state and depression of the run key causes one instruction to be executed, and then the computer re-enters the idle state (since input/output instructions cannot be interrupted, the computer will not stop immediately after an IOT instruction and hence more than one instruction may be executed if an IOT is involved). In the idle mode execution of a change to environment zero instruction or depression of the idle/pause key causes the computer to enter the idle state in which the contents of registers are displayed and data may be set into