An Input/Output Processor for the XDS 930: An Exercise in Micro-programmed Design

A. I. Zygielbaum
Communications Systems Research Section

A micro-programmed stored logic input/output (I/O) processor has been developed to evaluate micro-programming as a digital design technique. This I/O processor can be used for investigating DSN standard computer/computer interfaces as well as for experimentation with external control of the XDS 930 in emulation of multiple computer systems.

I. Introduction

A micro-programmed stored logic I/O processor has been developed to evaluate micro-programming as a digital design technique. The processor was specifically designed as an I/O processor for an XDS 930 to minimize loading of the CPU when driving the digital video display system (Ref. 1). This flexible I/O processor can be used for investigating DSN standard computer/computer interfacing, and for experimentation with external control of the XDS 930 in emulation of the control of satellite minicomputers in a multiple-computer DSIF tracking station.

Micro-programming was found to provide an orderly straightforward design process, simplifying digital system debugging, maintenance, and documentation. This article will discuss micro-programmed stored logic and the minicomputer-like I/O processor (Mini-proc).

II. Definitions

To clarify the terms to be used, the following definitions are applicable. "Stored logic" shall imply a digital system where system states are contained in the address counter of a sequencing memory (control store) rather than in the conventional conglomeration of gates and flip-flops. "Micro-instruction" will refer to a particular system state transition function stored in this memory. A "micro-program" is a sequence of micro-instructions. An "instruction" is a word stored in a computer's memory for the

purpose of controlling that computer through its central processor. A "command" is a word stored in a computer's memory for the purpose of controlling an input/output processor.

III. Background

In 1951, M. V. Wilkes (Ref. 2) proposed a computer controlled by the output of a diode matrix memory. Each word of this memory controlled the gating structures of the computer's functional units (adders, registers, etc.) in some particular way. Each bit of the word controlled a unique gate in the structures. This scheme allowed the independent design of functional units and the straightforward sequencing of data transfers.

Using one bit per function in the micro-instruction is known as "horizontal format." A simple example is shown in Fig. 1. A simple micro-program can, for instance, subtract register 2 (R2) from register 1 (R1); then double the result. Assume a synchronizing clock for each step:

ADDRESS	WORD	
000	10011001	Subtract R2 from R1. Store result in R1 and go to NEXT INSTRUCTION.
001	00111011	Add R1 to itself, store result in R1 and Step
010	00000000	HALT (NO STEP, NO FUNCTION)

It is unlikely that gates G1, G2, G3, and G4 will need to be enabled at once. The only likely possibilities are G1, G2; G3, G4; G1, G4; and G2, G3. These four conditions can be encoded into two bits thus reducing the word size from 8 bits to 6 bits. This is an example of "vertical format."

Vertical formating breaks the micro-instruction into encoded fields. There are two primary disadvantages relative to horizontal format. First, the system loses flexibility because some logic states are disallowed. Second, decoding involves extra hardware and design time. However, the advantages of vertical format are a shorter micro-instruction, coupled with easier microprogram development and documentation, since mnemonics can be assigned for each field. Usual design involves a combination of both formats as shown in Fig. 2.

One can establish a hierarchy of levels where a "nano-instruction" mechanizes the execution of a micro-instruction just as a micro-instruction mechanizes a computer instruction. The unique vertically formatted micro-instruction of the Nanodata QM-1, for instance, is essentially an address which selects a nano-instruction in a nano-memory (Ref. 3). The nano-instruction, which has a horizontal format, controls gate level functions. Consequently, the QM-1 offers an extremely flexible combination of both formats.

One must note that, no matter what the format, the present state of the system is contained in the current micro-instruction address, and the next state is easily determined. In a conventional system the current state is spread over many locations.

The states of the stored logic machine need not be sequential. Branching on condition and subroutining can be easily implemented. Repetitious or conditional functions are a simple extension to the stored logic. This offers additional logic efficiency to the digital system. In the example of Fig. 1, branching can be accomplished by leaving the low-order control store address bit unspecified and allowing that bit to be set by some condition, like adder carry. The higher order bits would merely step sequentially.

The development of integrated circuits and medium/large-scale integrated (MSI/LSI) circuits encourages the modularization of digital system design. Modularization lends itself quite readily to stored logic which offers a simple way to control data transfers between MSI/LSI modules. Further, MSI/LSI read-only memories (ROMs) and random-access memories (RAMs) offer attractive storage for micro-programs. MSI/LSI circuits make stored logic an economically feasible choice for digital system design.

Current use of micro-programming in computers is well known. Most popular mini-computers as well as the IBM System/360 are micro-programmed. The 360 uses a vertically formatted micro-instruction to implement a fixed computer instruction set which reaches across a compatible line of computers. As the System/360 is a business data machine, IBM will not support a user micro-programmed machine. This restriction is necessary to insure software compatibility between installations. Minicomputers can be micro-programmed by the user to specialize machine capabilities as was done in the DSN data decoder assembly (Ref. 4). In this application, an

Interdata 4 mini-computer was microprogrammed to perform sequential decoding of the Pioneer 10 telemetry at data rates adequate to support the Pioneer Program.

IV. Rationale for Stored Logic

On the XDS 930, direct memory access is typically through the data multiplex system (DMS) which operates through the multiple access to memory (MAM). The DMS can transfer a block of up to 512 words without central processing unit (CPU) intervention. The CPU must, however, update word count and data location between each block.

The digital video display system (DVDS, Ref. 1) uses blocks on the order of 30 words. For this block length the CPU is 50% occupied with the job of updating the data multiplex system (DMS) control words. To reduce this load, an I/O processor was designed for the XDS 930. This I/O processor, controlled by a command program in memory, must transfer data at full memory bandwidth (570,000 words/s) and perform system status tests, both independent of the CPU.

Initial design proceeded in the usual manner. Several block diagrams were tried. Each worked but each had its limitations. Typically a design which was optimal for the DVDS was not efficient for other I/O. Compromises and the inability to include all contingent I/O requirements lead to the abandonment of conventional logic in favor of stored logic.

V. Functional Units

The Mini-proc is made up of independent functional units (Fig. 3). These units are interconnected by a single 8-bit bus. This two-way path allows the output of any functional unit to be connected to any combination of functional unit inputs.

The functional units were designed independently. The only specifications were: (1) logic level delays from enable to output of 100 ns maximum, (2) transistor-transistor logic (TTL) compatible 5-V logic levels, and (3) 8-bit wide data input and output. The control store word was not defined until after the basic design was completed.

The manipulator is the data modifier in the Mini-proc. Referring to Fig. 4, the primary operand enters the manipulator through a shifter into the arithmetic logic unit (ALU). The shifter can shift data one bit right cyclic, left cyclic or left noncyclic. The shifter's output supplies the "A" input of the ALU. The second operand is stored in the arithmetic logic buffer which is routed to the B input of the ALU. In the ALU, two Texas Instrument SN74181 (arithmetic logic units) provide 16 logic functions and two sets of 16 carry dependent arithmetic functions for eight bit bytes. The ALU can also be used to compare two numbers. A special A = B signal is available to the status functional unit. The output carry, saved in a flip-flop for the status unit, can be routed back into the ALU to facilitate multi-byte arithmetic. The output of the ALU is stored in the accumulator for distribution on the bus. Sixteen 8-bit words of storage are available in a semiconductor scratch pad. The scratch pad address is supplied by the control store.

Direct communication to the Mini-proc from the XDS 930 CPU is by way of the parallel output (POT) channel. The CPU can start or stop the processor. A *channel status dump* is available to the CPU on request. The *interface control* decodes the necessary XDS 900 series EOMs and signals the status functional unit as required. Two EOMs override control store. The *unconditional activate* forces the Mini-proc to reinitialize and start. The *stop immediate* halts the processor no matter its current status. The interface control can interrupt the CPU using two interrupt channels. In case the CPU tries to activate an already active Mini-proc, one interrupt channel provides a warning. The second interrupt is under the control of control store.

Primary data and command transfers between the XDS 930 memory and the Mini-proc is by way of the MAM. The interface control synchronizes all data and address transfers on this channel. The unit also generates word parity to the XDS 930 and checks parity from the XDS 930. The address buffer supplies the memory location address to the MAM. Sixteen bits of storage are available although only fifteen are used for the address. Two Mini-proc cycles are required to transfer address data in or out by way of the 8-bit bus. The data buffer assembles 24 bits of data in three Mini-proc cycles for transfer to the XDS 930 via the MAM. Conversely the buffer unpacks 24 bits into 8-bit bytes in three cycles for distribution on the bus.

Communication to the outside world is through the external control functional unit. Besides providing synchronizing signals for XDS 930 memory access, this unit

has four special-purpose timing lines available to control store. These lines can be used for arbitrary control line simulation.

The Mini-proc's single bus can be gated to the outside world for direct I/O. This facilitates simulating XDS 930 EOM-type instruction. Special status conditions can be input in this manner. The external control also provides a test line which is routed to the status functional unit.

The status unit keeps track of various internal and external conditions. Its primary function is the control of conditional jumps. Any of the following three specific conditions can be tested: Manipulator A = B, Manipulator carry, and External test. The status unit can also provide an interrupt-like jump, that is, a jump to a specific address if any condition is set. The conditions are tested on a priority scheme and are reset once handled. The jump addresses, one for each condition, are defined by an eight diode dual in-line package (DIP) carrier on the status board.

The control address buffer selects the active micro-instruction. This micro-instruction is stored in the control buffer. The address buffer can be reset to zero or incremented by one. To facilitate micro-program branching, the control address buffer can also copy jump addresses from the bus.

In the initial version of the Mini-proc, the control store is a diode matrix memory with a 100-ns access time. Diode matrices, though cumbersome, are easily changeable and therefore amenable to experimentation. The control store operates in parallel with the functional units. The next micro-instruction is fetched while the current one is being processed. A conflict occurs with a jump micro-instruction. Here the fetched word is not the next one to be processed. The usual solution is to ignore the fetched word and thus lose a machine cycle. The Mini-proc, however, handles a jump by allowing one sequential micro-instruction to be executed after the jump micro-instruction occurs. The programmer would then put a jump instruction one step earlier than necessary or follow the jump with a clear word (no-operation).

The format of the micro-instruction is a combination vertical/horizontal format. An explanation of the format appears in Figs. 5 and 6. Basically the output of any functional units can be connected to any combination of inputs under control of the control field.

VI. Micro-Programming

The sample micro-program (Fig. 7) is the start up routine for the Mini-proc. The XDS 930 stores a command program starting address in location 0234 of its memory. The Mini-proc is enabled and accesses that location. The micro-program then jumps to the routine called for by the command in the starting address location. This micro-program was written with the aid of a Meta-symbol procedure deck written for the XDS 930. The assembly language has fields as shown in Table 1.

The card image corresponds to the input data for the assembler. The label field is an address locator. The device field corresponds to the micro-instruction device field. (AORG sets the absolute origin. END implies an end to the program.) The control/input field uses the input mnemonics previously given in Figs. 5 and 6. A control function is written by giving a name and its corresponding value such as ('L', 234). This places the value 234 in the label subfield of the control field. Additional field definitions are found in Table 2. The assembler produces a binary tape of the 27 bit micro-instruction words for later implementation.

Each command of the processor program calls a specific micro-programmed routine. The initial command list will consist of the following:

LOAD	IMMEDIATE	BLOCK COUNT
LOAD	IMMEDIATE	WORD COUNT
LOAD	IMMEDIATE	DATA STARTING ADDRESS

JUMP

DECREMENT, JUMP IF LOC. < 0

INPUT ONE WORD

OUTPUT ONE WORD

TRANSFER DATA (according to word, block count)

VII. Summary

Design of the Mini-proc has demonstrated the value of stored logic. Micro-programming offers the digital system designer an orderly method to produce logical controls. The efficiency of design, along with simplified documentation and debugging, suggests that stored logic is an attractive and viable alternative to ad hoc design.

References

1. Zygielbaum, A. I., "Information Systems: Range Doppler Display System," in *The Deep Space Network*, Space Programs Summary 37-60, Vol. II, pp. 22–28. Jet Propulsion Laboratory, Pasadena, Calif., Nov. 30, 1969.
2. Wilkes, M. V., "The Best Way to Design an Automatic Calculating Machine," presented at the Computer Inaugural Conference, Manchester University, North Manchester, Ind., July 16–18, 1951.
3. Rosen, R. F., et al., "An Environment for Research in Microprogramming and Emulation," presented at the 4th Annual Workshop on Microprogramming, ACM-Sigmicro-IEEE, Santa Cruz, Calif., September 13–15, 1971. Also appears in *Communications of the ACM*, Vol. 15, No. 8, pp. 748–760, August 1972.
4. Grauling, C. R., "Data Decoder Assembly," in *The Deep Space Network*, Technical Report 32-1526, Vol. IV, pp. 170–177. Jet Propulsion Laboratory, Pasadena, Calif., Aug. 15, 1971.

Bibliography

Leis, C., "Microprogramming Features of a 16-bit Mini," Preprint of the 4th Annual Workshop on Microprogramming, ACM-Sigmicro-IEEE, Santa Cruz, Calif., Sept. 13–15, 1971.

McDermott, J., "Suddenly, Everybody is Building Micro-Programmed Computers," *Electronic Design*, Vol. 24, pp. 23–28, Nov. 25, 1971.

Redfield, S. R., "A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor," *IEEE Transactions on Computers*, Vol. C-20, No. 7, pp. 743–750, July 1971.

Rosin, R. F., "Contemporary Concepts of Microprogramming and Emulation," *Computing Surveys*, Vol. 1, No. 4, pp. 197–212, December 1969.

Wilkes, M. V., "The Present States and Potential of Microprogramming," *IEEE Convention Record*, pp. 131–133, March 25, 1971.

Table 1. Assembly language format

Relative location	Assembled code			Card number	Card image		
	Tag	Device	Control/input		Label	Device	Control/input
XXXXX	X	XX	XXXXXXX		(INIT)	(RT B)	('L', O), BTADB

Table 2. Field definitions

Mnemonic	Corresponding 'field'	Remarks/sub-class
MCM	B	ALU Control
BTSP	A	Scratch Pad
SP	A	Used with SPTB
TIC	D	HALT – Stop RIN – Request Input RAC – Request Access RDCS – Request Dump Channel Status ENINT – Fire Interrupt
TAG	TAG	
ST	E	STATUS
X	C	EXTERNAL
L	F	LITERAL

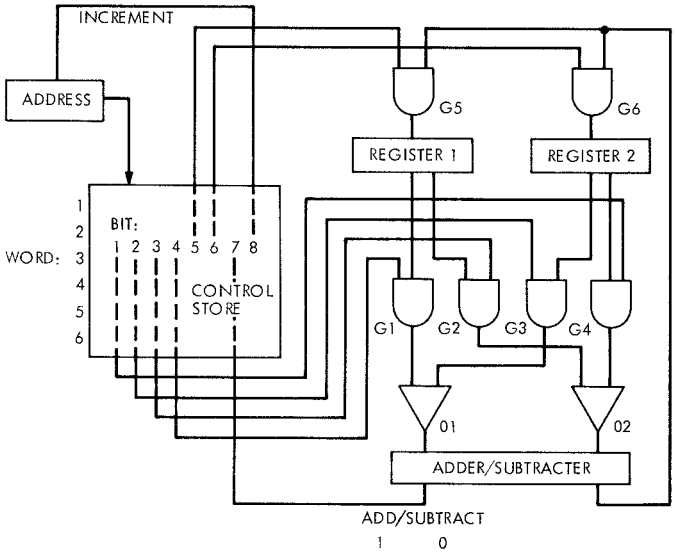


Fig. 1. Simple digital system, horizontal format

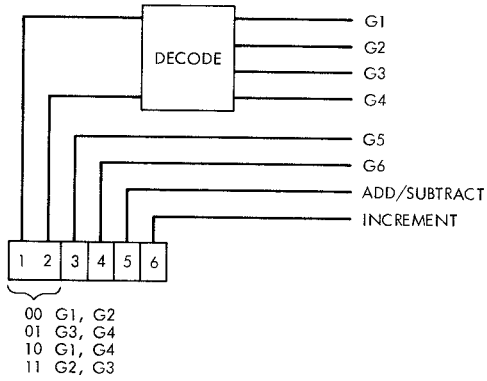


Fig. 2. Simple digital system, vertical format

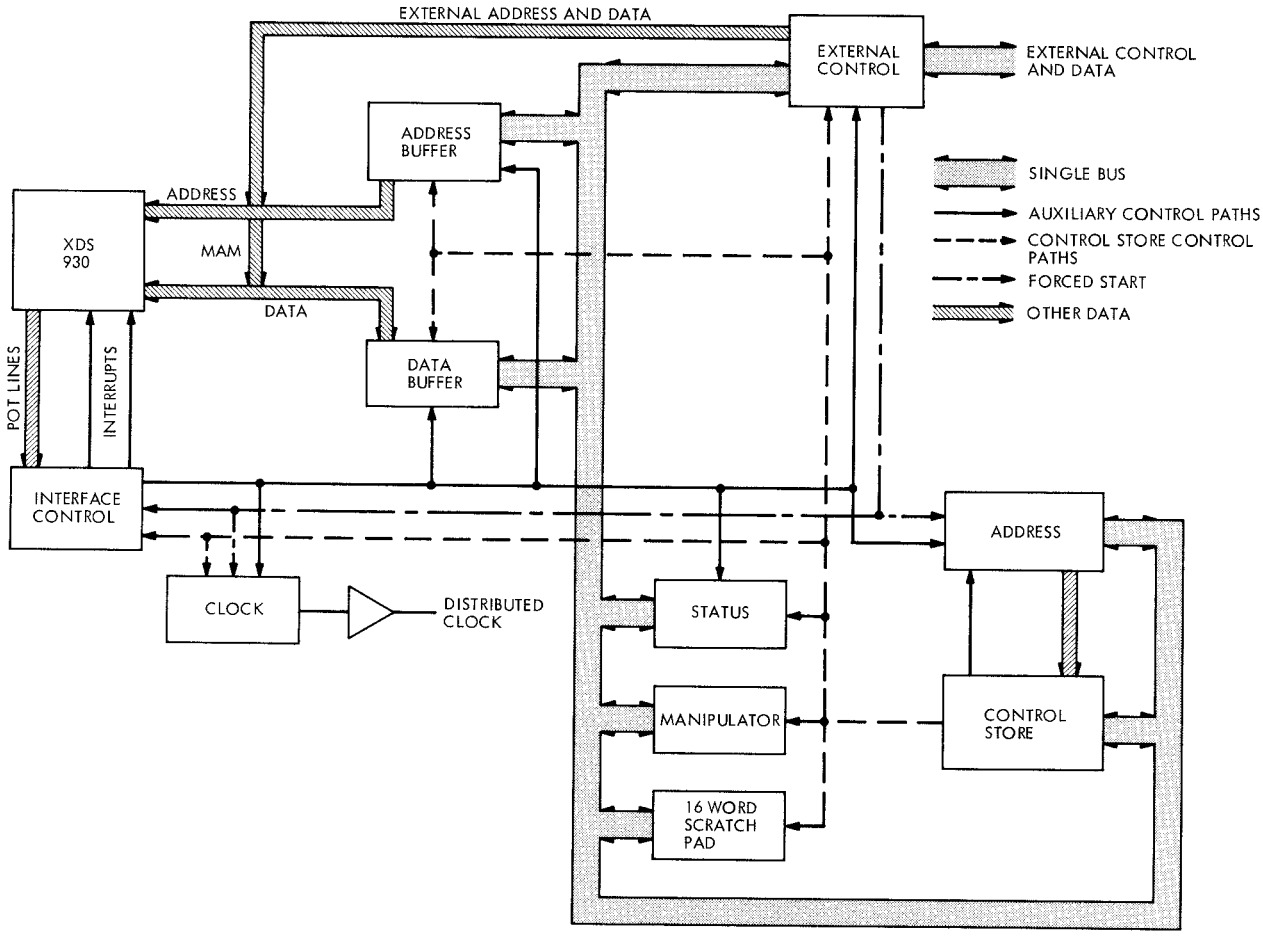


Fig. 3. Block diagram of Mini-Proc

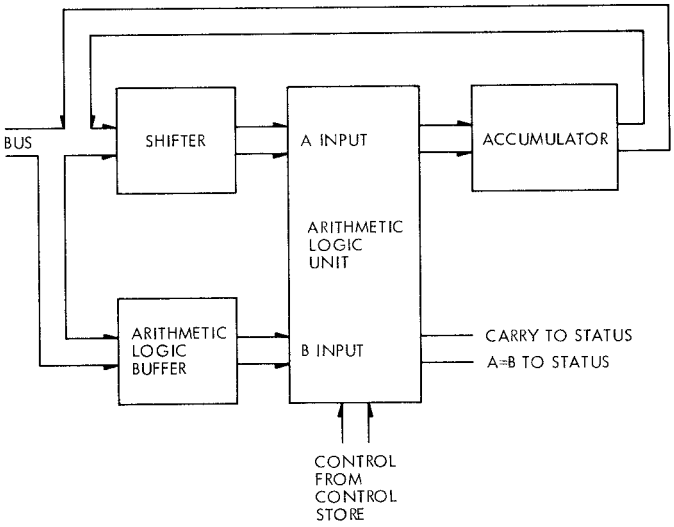
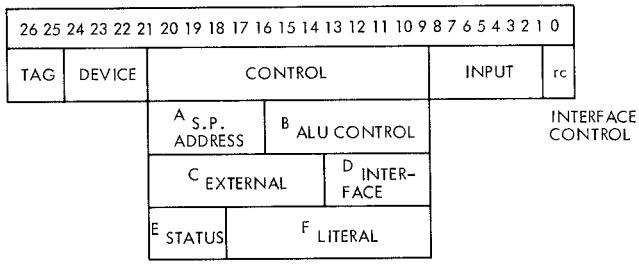


Fig. 4. Manipulator



TAG FIELD

VALUE	MNEMONIC	ACTION
01	LRC	LOAD CONTROL MODIFY
10	RC	MERGE CONTROL MODIFY

DEVICE, CONTROL

VALUE	MNEMONIC	CONTROL FIELD ENABLED	ACTION
0000	NOB		NO ACTION
0001	RA↑B		CONTROL ADDRESS TO BUS
0010	AC↑B		ACCUMULATOR TO BUS
0011	DB↑B		DATA BUFFER TO BUS
0100	C↑DB		COMPUTER TO DATA BUFFER
0101	DB↑C		DATA BUFFER TO COMPUTER
0110	ADB↑B		ADDRESS BUFFER TO BUS
0111	SP↑B	A	SCRATCH PAD TO BUS
1000	ST↑B	E	STATUS TO BUS
1001	EX↑B		EXTERNAL TO BUS
1010	EXTCTL	C	EXTERNAL CONTROL
1011	R↑B	F	CONTROL STORE LITERAL TO BUS
1100	EXTRST		OPTIONAL TIMING RESET
1101	STOP		DEACTIVATE MINI-PROC
1110	EXPY		SUPPLY EXTERNAL PARITY

INPUT

BIT	MNEMONIC	CONTROL FIELD ENABLED	ACTION
8	JP		JUMP (BUS TO CONTROL ADDRESS)
7	XJP	E	JUMP ON CONDITION
6	MCM	B	MANIPULATOR CONTROL
5	B↑ALB		BUS TO ARITHMETIC LOGIC BUFFER
4	B↑DB		BUS TO DATA BUFFER
3	B↑ADB		BUS TO ADDRESS BUFFER
2	B↑SP	A	BUS TO SCRATCH PAD
1	B↑EX		BUS TO EXTERNAL

INTERFACE CONTROL

INTERFACE CONTROL FIELD (D) ENABLED IF BIT 0 (RC) IS A ONE.

Fig. 5. Control store micro-instruction format

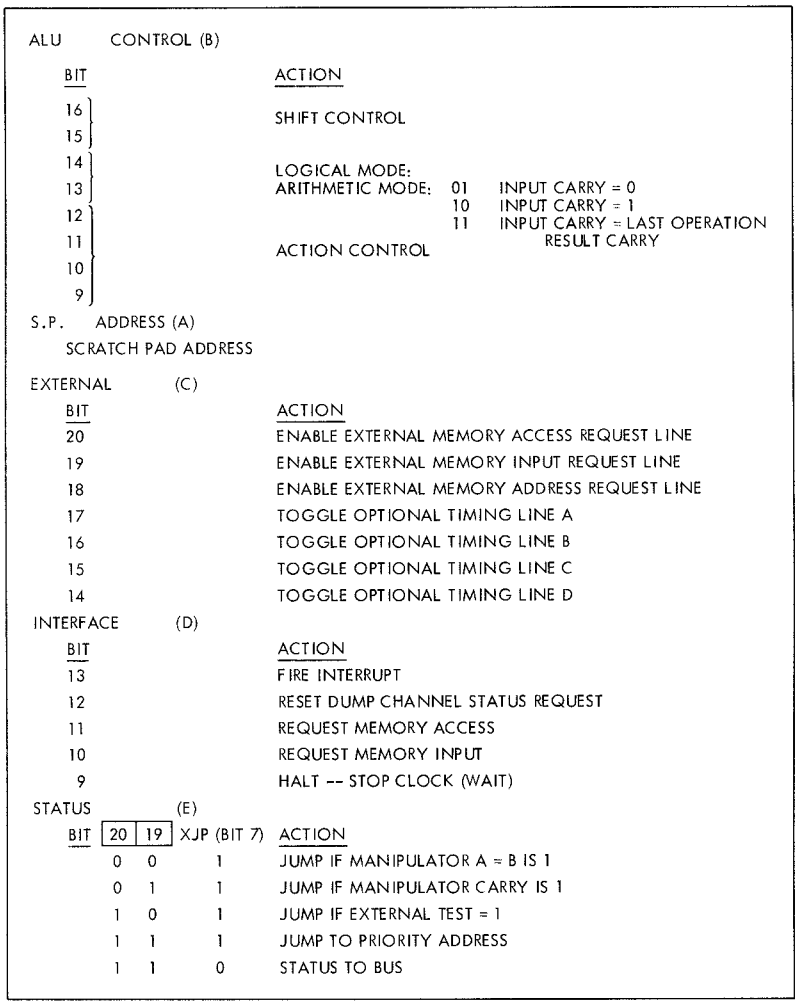


Fig. 6. Control fields

```
                                    79          PAGE
                                    80 * START MINIPROC ROM PROGRAM
00000                               81          ABRG    0
00000   0 13 0000010                82 INIT     RTB     ('L',0),BTADB
00002   0 13 0352010                83          RTB     ('L',234),BTADB
00004   0 00 0005001                84          NBB     ('TIC',RAC,HALT)
00006   0 04 0000000                85          CTDB
00010   0 10 6000200                86          STTB    ('ST',3),XJP
00012   0 03 0057104                87          DBTB    ('BTSP',0),('MCM',0,2,15)
00014   0 03 0400014                88          DBTB    ('BTSP',1),BTADB
00016   0 07 0004011                89          SPTB    ('SP',0),BTADB,('TIC',RAC)
00020   0 02 0000004                90          ACTB    ('BTSP',0)
00022   0 07 0477100                91          SPTB    ('SP',1),('MCM',0,3,15)
00024   0 02 0401005                92          ACTB    ('BTSP',1),('TIC',HALT)
00026   0 04 0000000                93          CTDB
00030   0 10 6000200                94          STTB    XJP,('ST',3)
00032   0 13 7432004                95          RTB     ('BTSP',15),('L',$)
00034   0 03 0000400                96          DBTB    JP
        00000000                    97          END     INIT
```

Fig. 7. Sample micro-program