AN EMULATION ORIENTED, DYNAMIC MICROPROGRAMMABLE PROCESSOR *

by Charles Neuhauser

ABSTRACT

The design of a low cost, emulation oriented, dynamic microprogrammable processor is proposed. This machine is being built at Johns Hopkins University for use both as a research tool and an instructional aid. Features of this machine include: field isolation instructions, flexible access to microprogram writable control memory and convenient use of immediate data.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

* This work was supported by the U. S. Atomic Energy Commission under contract AT (11-1 3288)

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

fy

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

1. GENERAL SYSTEM DESCRIPTION	1
1.1 Purpose and Objectives	1
1.2 Description of Laboratory Facilities	1
1.3 Description of Support Software	1
2. DESCRIPTION OF THE MICRO MACHINE	3
2.1 Data Transformation Resources	3
2.2 Register Resources	3
2.3 Memory Resources	3
3. MICROINSTRUCTION DESCRIPTION	5
3.1 Binary Instruction Formats	5
3.2 Shifter/Auxiliary Instruction Format	5
3.3 Conditional Instruction Format	5
3.4 Insert/Extract Instruction Format	6
3.5 AI Field Format	6
3.5.1 Auxiliary Opaeration Commands	6
3.5.2 AI Field Used as Immediate Data	7
4. DESCRIPTION OF MICRO MACHINE UNITS	8
4.1 Busses	8
4.2 Control Signals	8
4.3 Binary/select Units	9
4.4 Shift/Rotate Unit	9
4.5 Insert/Extract Unit	9
4.6 Conditional Unit	10
4.7 Conditional Exasmination Unit	10
4.8 Field Expansion Unit	10
4.9 Auxiliary Operation Unit	10
4.10 Registers	11
5. SYNTACTIC DEFINITION OF THE JHU-MICRO ASSEMBLY LANGUAGE	12

1. GENERAL SYSTEM DESCRIPTION

1.1 Purpose and Objectives

The JHU-microprogramming laboratory system is being designed to provide a research oriented microprogramming facility at low cost. Primarily emulation oriented, these facilities will be employed for both research in computer architecture and for student projects in microprogramming and computer organization. Our basic system objectives for the design of the JHU-micromachine are as follows:

- 1) A cost of approximately \$5,000,
- 2) Emulation oriented instruction set,
- 3) Writable micromemory, and
- 4) Straight forward design to allow easy implementation.

From our experience with other micromachine designs [1],[2], we felt that this design should include the following capabilities:

- 1) Operations with immediate data,
- 2) Flexible shift and rotate facilities,
- 3) Ability to select fields from words,
- 4) Low level parallelism where possible, and
- 5) Ability to access the microaddress register.

1.2 Description of Laboratory Facilities

Figure 1 shows the system organization of the microprogramming laboratory. There are four basic components:

- 1) JHU-micromachine,
- 2) 2200/JHU-micro interface,
- 3) Datapoint 2200 system, and
- 4) Peripheral devices.

The Datapoint 2200 is a self-contained, byte organized, processing system consisting of a processor, a CRT display, a keyboard and two cassette tape drives. Components of the 2200 system are connected by the processor I/O bus system. One member of the I/O bus system is the 2200/JHU-micro interface which handles communication between the 2200 and the JHU-micromachine. This interface will access the JHU-micromachine in two ways. First, it will have access to the device memory of JHU-micro in order to send and receive data, and second, it will have direct access to the microsequencing logic of JHU-micro in order to exercise direct control.

Communications between the Datapoint 2200 and the laboratory peripheral equipment take place through the 2200-400 communication adaptor, which converts parallel byte streams into EIA (Electronic Industry Association) compatible serial bit streams. The output of the communication adaptor may be directed to a model 38 teletype or a remotely located PDP-11 system. The teletype is capable of communication over voice grade telephone lines independently of the 2200 using an internal Bell System compatible modem.

1.3 Description of Support Software

Initially the microprogramming laboratory will require the

following support software:

- 1) Microassembler (running on the PDP-11 system),
- 2) Microloader (running on the 2200 system), and
- 3) Micromachine control program (running on the 2200 system).

Typically a microprogramming run will begin by first accessing a microassembly language file, stored either on the 2200 cassette system or on the PDP-11 disk system. After editing the file from the 2200, the microprogrammer will transmit it to the PDP-11 system for micro assembly. Assembled microprograms will be loaded into the JHU-micromachine from the 2200. Facilities within the micromachine control program will allow the experimenter to control the microsequencing (e.g. run, stop, step) of the micromachine. The microprogrammer will also use the micromachine control program to read and change micromachine registers and memory locations.

[1] L. Yelowitz et al; "A Simulator for a Microprogrammed Computer, It's Microassembler, and an Emulation"; Hopkins Computer Research Report #1; April 1971

[2] J. Davison; "The JHU Host Machine Simulator"; to be published

2. Description of the Micromachine

The architecture of the JHU micro machine is illustrated in figure 2. There are three classifications of resources:

- 1) Data transformation resources,
- 2) Register resources, and
- 3) Memory resources.

2.1 Data Transformation Resources

Data transformation resources operate on data obtained from the registers or the immediate field of the microword being executed. After processing, the data transformation resources return results to the registers. Depending upon the type, a data transformation resource may require one, two or three words of input data and will produce one word of output data and in some case a control signal to influence processing in the register or memory resources. The four types of transformation resources are:

- 1) Conditional unit - tests machine conditions and influences sequencing of the auxiliary operation unit
- 2) Binary unit - performs operations between two words of data.
- 3) Shift/Rotate unit - performs general shifting and rotating of data.
- 4) Extract/Insert unit - this unit is capable of taking a selected field from one data word and inserting it into a selected field of a second word.

2.2 Register Resources

Eight registers are available to store intermediate results and to provide communications between the memory and transformation resources. Each register is 32 bits in length. Two registers, registers 0 and 1, are distinguished as the MicroAddress Register (MAR) and the status register (SR), respectively. The MAR contains the address from which the microinstruction is to be fetched. The status register contains various quantities which define the state of the micromachine.

When data transformation resources require immediate data they may receive it from the Field Expansion Unit (FEU). The FEU receives skeleton data from the current microinstruction and expands it to a normal 32 bit quantity.

2.3 Memory Resources

In addition to the registers there are three memory resources in the micromachine. These are:

- 1) Micromemory,
- 2) Main memory, and
- 3) Device memory.

All memory resources are 32 bits in width. Micromemory consists of one or two K words of fast (about 200 nsec) storage and will be used to store the microprogram and the virtual registers of the emulated machine. Main memory is slower (about 1.2 usec) and will be used

to hold the program of the emulated machine. Device memory is a group of main memory locations communicating directly with external devices.

The auxiliary operation unit (AOU) controls all communications between the various memory resources and between these resources and the registers. The AOU receives directions from the current micro-instruction word.

3. MICROINSTRUCTION DESCRIPTION

Figure 3 shows the five basic formats of the microinstruction words used by the JHU micromachine. A microinstruction word may be divided into two sections. The first 14 bit group is called the CODE field and the remaining 18 bit group is referred to as the AI field. The CODE field is used to specify the operation of the data transformation resources. The AI field, depending upon the instruction type, may be used to specify the operation of the auxiliary operation unit or be expanded by the field expansion unit for use as immediate data.

3.1 Binary Instruction Formats (FMT=00)

There are two types of binary instruction format:

- 1) Binary/Auxiliary
- 2) Binary/Immediate

Bit 7 of the CODE field determines the type. A Binary/Auxiliary format specifies a binary operation between two registers and a AOU operation to take place simultaneously. In a Binary/Immediate format the CODE field specifies a binary operation to take place between a single register and immediate data which is supplied by the AI field, encoded for FEU expansion.

Four fields are defined for binary formats:

- BINOP - binary operation to be performed
- X - Binary/Immediate or Binary/Auxiliary
- REG1 - source of one operand and destination of result
- REG2 - source of second operand

Figure 4 gives more details of the binary instruction format.

3.2 Shifter/Auxiliary Instruction Format (FMT=01)

In the Shifter/Auxiliary instruction format the CODE field specifies an operation to be performed by the shift/rotate unit, while the AI field specifies a AOU operation to take place simultaneously. Figure 5 gives the details of the Shifter/Auxiliary format. The fields used are as follows:

- AMOUNT - specifies amount of the shift
- CONTROL - defines the type of shift/rotate operation to be performed (e.g. right shift logical)
- REG - specifies a register which is the source and destination of the data to be transformed

3.3 Conditional Instruction Format (FMT=10)

The AI field of the conditional instruction specifies an auxiliary operation. This operation will be performed only if the conditional unit finds the condition specified in the CODE field to be valid. To check validity the conditional unit compares the condition codes stored in the status register to the condition mask and type of testing given in the CODE field. If the condition is valid the the AOU operation will be allowed to proceed and the MAR will be gated to the register specified. Figure 6 gives the details of the conditional

instruction format. The fields are:

CONDITION MASK - specifies conditions to be tested
TYPE - specifies how the conditions are to be interpreted
REG - specifies destination of the MAR is the condition is found to be valid. This feature may be used to save the MAR when a CALL is performed.

3.4 Insert/Extract Instruction Format (FMT=00)

Figure 7 gives the details of the Insert/Extract instruction format. This instruction allows the programmer to right rotate a register, and from it insert a specified field into another register. The register which receives the field may be cleared before insertion if desired. With this instruction fields may be extracted from a data word, right justified and placed in a destination register. This instruction also allows the programmer to modify a selected field in a register, such as the status register.

The sub-fields of this instruction format are:

AMOUNT - amount by which the source data is to be rotated
I - indicates whether the destination register is to be cleared before insertion of masked data
D-REG - destination register
S-REG - source register

The AI field of the Insert/Extract instruction is used to specify the mask to be used for insertion.

3.5 AI Field Format

The AI field may be used for two purposes; to specify auxiliary operations or to specify immediate data for the field expansion unit. AI field formats for the specification of auxiliary operations are shown in figures 8 and 9. When the field is used as immediate data the format shown in figure 10 is used.

3.5.1 Auxiliary Operation Commands

Figure 8 diagrams the format of the AI field when it is used to specify auxiliary operations. Four types of operations may be specified:

- 1) Register to micromemory data transfers,
- 2) Immediate loading of registers,
- 3) Block movement of data between main and micromemory, and
- 4) Conditional branching relative to the current MAR address.

For the first three types of operation the format shown in figure 8 applies while for conditional branching operations the format in figure 9 applies.

Operation codes 0-3 specify the movement of data between registers and micromemory. For an LD operation the ADR field specifies the source of data in micromemory and the REG field specifies

the destination in the register file. For an SD operation the meanings of the REG and ADR fields are interchanged. LI and SI are indirect operations in which the register specified in the REG field is used to point to the actual destination or source. For indirect operations the contents of the register designated by REG will be considered as 11 bit quantities, that is, only micro-memory addresses may be accessed with LI and SI operations.

Operation codes 4 and 5 specify the immediate loading of the register specified by REG with data specified in the ADR field. IM0 specifies that the register will be left filled with zeros, while IM1 specifies that the register will be left filled with ones. This allows the loading of both positive and negative quantities. Since the ADR field is 11 bits wide the micro-programmer may use the IM0 operation to cause a jump to any micromemory location by an immediate load of the MAR.

Block transfers of data between micro and main memory are specified by the operation codes 8-15. Codes 8-11 specify block loading starting at the address specified by the contents of the register given in the REG field from the address specified in the ADR field. Codes 12-15 specify the movement of blocks of data from the starting address specified in the register given in the REG field to the address given in the ADR field. In both cases blocks of 1,2,4, or 8 words may be transmitted.

Auxiliary operation codes 6 and 7 are used to denote conditional branching. The format for these codes is shown in figure 9. The fields denoted by MASK and TYPE correspond exactly to the similarly name fields in the conditional instruction format (figure 6). Testing of the condition or indicator codes set by the previous microinstructions is made in exactly the same manner as described for the conditional instruction. If the test is valid then the MAR will be incremented (or decremented) by the amount specified in the AMOUNT field. This instruction allows the microprogrammer to make efficient use of conditional testing for short relative jumps.

3.5.2 AI Field Used as Immediate Data

For Binary/Immediate and Insert/Extract instructions the AI field is used to specify immediate data. The AI field, which consists of 18 bits, is expanded into a 32 bit quantity by the Field Expansion Unit. When used as immediate data the AI field contains two sub-fields EXP and DATA/MASK. The DATA/MASK field consists of 16 bits of data. The EXP (EXPansion) field specifies how the Field Expansion Unit will position this 16 bit DATA/MASK field and fill out the remaining 16 positions. The DATA/MASK field may be left justified, right justified or centered as shown in figure 10. If the field is right justified it may be filled with either zeros or ones (allowing negative numbers). If the field is centered or left justified the fill bits are zero.

4. DESCRIPTION OF MICROMACHINE UNITS

Figure 2 illustrates the structure of the JHU micromachine. This section will describe the function of the various units and their interconnection.

4.1 Busses

Within the data transformation resource sub-section of the micromachine there are four data busses used for communication between transformation resources and registers. These busses are:

- 1) Immediate - supplies immediate data from the FEU as input to the transformation resource units.
- 2) Operand 1 - supplies the primary operand from the registers to the transformation units.
- 3) Operand 2 - supplies the secondary operand from the registers to the transformation units.
- 4) Result - returns the output result from transformation units to the registers.

4.2 Control Signals

In addition to data inputs and outputs the various resource units in the micromachine receive control signals. There are four classes of control signals:

- 1) Select - cause a register to be connected to a bus
- 2) Enable - cause a unit to complete an operation
- 3) Operate - define a unit operation
- 4) Conditional - can prevent an enabled unit from operating

With the exception of the conditional control signals, all of the control signals are derived from the CODE portion of the Micro-Instruction Register (MIR) by the decode unit. Conditional control signals are generated by a resource unit when it is necessary for one unit to control another.

Below is a list of the control signals used.

Signal - # Bits - Type - Description

BOP	-	5	- OP	- Operation code for binary unit
COP	-	9	- OP	- Operation code for conditional unit
IOP	-	2	- OP	- Operation code for Ins/Ext unit
ROP	-	9	- OP	- Operation code for shift/rotate unit
SOP	-	1	- OP	- Operation code for 'A' input to binary unit
SRI1	-	3	- SEL	- Register select for result bus
SRO1	-	3	- SEL	- Register select for operand 1 bus
SRO2	-	3	- SEL	- Register select for operand 2 bus
BEN	-	1	- ENBL	- Gate binary unit to result bus
CEN	-	1	- ENBL	- Gate conditional unit to result bus

EEN	-	1	- ENBL	- Gate examination unit results to STATUS register
AEN	-	1	- ENBL	- Start Auxiliary Operation Unit
IEN	-	1	- ENBL	- Gate Ins/Ext unit to result bus
COND	-	1	- COND	- If 0 terminate AOU operation If 1 allow AOU operation to continue
CMP	-	1	- COND	- TRUE if binary operation is compare
GATE	-	1	- COND	- If 0 do not gate result bus to register If 1 gate result bus to register

4.3 Binary/Select Units

The binary unit provides the arithmetic and logical functions used in data transformation. This unit accepts two inputs, A and B, and returns a result. The B input is designated as the operand 2 bus while the A unit receives input from either the immediate operand bus or the operand 1 bus as determined by the select unit. The binary unit will be implemented with an MSI ALU which will allow at least the operations shown in figure 3. Operations will be divided into two classes, logical and arithmetic. All 16 binary, two valued logical operations will be supplied since they are implemented by the MSI ALU. These logical operations include register clearing, transferring, complementing and immediate loading.

Eight arithmetic operations and comparison are defined. Arithmetic will be either ones or twos complement depending upon a bit in the STATUS register. The eight arithmetic operations are:

AD	Addition
ADC	Addition with carry
SU	Subtraction
SUB	Subtraction with borrow
INCR	Increment by one
DECR	Decrement by one
AC0	Addition of zero with carry (carry propagate)
SU0	Subtraction of zero with borrow (borrow propagate)

A comparison operation is provided which is equivalent to an SU operation except that the CMP conditional control signal line is set to true this allows the results of the subtraction to be gated to the result bus for examination but prevents the results from being gated into the registers.

4.4 Shift/Rotate Unit

The shift/rotate unit accepts a single data word from the operand 1 bus and performs a shift or rotate of data as directed by the ROP control signal. Results are gated to the insert/extract unit. Shift/rotate unit operations are defined in figure 5.

4.5 Insert/Extract Unit

This unit accepts three inputs: MASK, a mask from the immediate bus, D, destination register data from the operand 2 bus, and S, rotated source data from the shift/rotate unit. Depending upon the IOP control signal the insert/extract unit may compute the

output result as follows:

- 1) Pass - RESULT <- S
- 2) Insert - RESULT <- (MASK and S) or (~MASK and D)
- 3) Extract- RESULT <- (MASK and S)

4.6 Conditional Unit

The conditional unit receives inputs from the operand 1 and operand 2 busses. Operand 1 bus contains data from the MAR and operand 2 bus contains condition code data from the status register. The COP control signal describes how the condition codes received from the status register are to be treated. This processing is outlined in figure 6. The conditional unit has two outputs. One is the contents of the MAR as received and the other is the conditional control line COND. Normally the COND line shows a 'true' signal which allows the AOU to operate as directed and gating of the register input unit R11 to take place. If the conditional unit finds that the conditions received from the status register do not fulfill the criteria set by the COP signal, then the COND line will be changed to 'false' inhibiting the AOU and R11 gating.

4.7 Conditional Examination Unit

The conditional examination unit is continually monitoring the data being sent on the result bus and setting the condition codes described in figure 12. When an EEN enable control signal is received the condition codes will be gated to the condition code segment of the status register. In general EEN will be sent on binary or shift/rotate unit operations.

4.8 Field Expansion Unit

The field expansion unit receives input from the AI field of the MIR. This field is expanded as illustrated in figure 10 and gated to the immediate bus for use by transformation resource units.

4.9 Auxiliary Operation Unit

The Auxiliary Operation Unit (AOU) has four functions: transferring data between memory resources, immediate loading of register data, relative conditional branching and interrupt handling.

There are two classes of data movement operations; those for moving single words between registers and micromemory and those for moving blocks of data between micro and main memory. For direct operations (LD, SD) the source and destination addresses are given in the AI field of the microinstruction. When indirect or block transfer operations are specified one of the address will be contained in a register designated by the AI field. Only block transfer instructions (BL1-8, BS1-8) may access main memory. Only one request from main memory access may be handled at a time. If more than one request is made to main memory the microinstruction fetching sequencer will halt. Micromemory operations specified in the AI field require one cycle in addition to the cycle required for execution of the CODE field specified operation. If both source and destination are registers then

no additional time is required.

Figure 11 is an illustration of the memory address scheme of the micromachine. Registers are assigned to overlap the highest eight addresses of micromemory, allowing increased addressing flexibility. The high limit of addressing is 377777 ($2^{18}-1$). Main memory and micromemory grow towards each other from these limits. The highest 1K of main memory addresses are assigned to the device memory. Device memory is accessed by addressing these memory locations. The structure of device memory has not been decided upon yet.

When performing an immediate load of a register the AOU expands the data given in the AI field with 0's (IM0) or 1's (IM1) as required. Immediate register loading will be done in parallel with the CODE field specified operation and will require no additional cycles.

Conditional branching allows the programmer to modify the MAR by incrementing or decrementing up to 31 if a given condition is fulfilled. This operation is performed in parallel with the transformation resource operation and requires no additional cycles.

Provisions are made for the micromachine to receive one external interrupt signal. Upon reception of this signal the micromachine will:

- 1) Replace the high 16 bits of the MAR with the low 16 bits of the MAR,
- 2) Replace the MAR with the address specified in the low 11 bits of the status register,
- 3) Disable interrupts, and
- 4) Jump to the new MAR address.

Interrupts are allowed only at the completion of a micro-instruction (except for block loads and stores). An interrupt requires one cycle to complete

4.10 Registers

Two of the eight registers are assigned special significance. Register 0 is designated as the MicroAddress Register (MAR) which will contain the address of the current microinstruction. Loading the MAR with an address is the equivalent of a jump. Register 1 is the status register. The format of this register is shown in figure 12. There will be four fields:

CONDITION	- (6 bits)	- contains the condition codes last stored by the conditional examination unit
INDICATOR	- (6 bits)	- contains information which can only be changed by the programmer
STATE	- (5 bits)	- contains machine state information (see figure 12)
INTRADDRESS	- (11 bits)	- contains the address that the micro-machine will jump to if an interrupt occurs

5. SYNTACTIC DEFINITION OF THE JHU-MICRO ASSEMBLY LANGUAGE

Primitives

```

<zero> ::= 0
<octit> ::= 01121314151617
<sub digit> ::= 11213141516171819
<digit> ::= <zero>|<sub digit>
<letter> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<character> ::= <letter>|<digit>|<space>|<underscore>
<underscore> ::= _
<null> ::=
<space> ::= (a space)
<sps> ::= <space>|<sps><space>
<sep> ::= <sps>,|<sps>|<sps>,<sps>
<eol> ::= <line feed><carriage return>

```

Elementary Forms

```

<octal constant> ::= <zero>|<octal constant><octit>
<decimal constant> ::= <sub digit>|<decimal constant><digit>
<name> ::= <letter>|<name><letter>|<name><digit>|<name><underscore>
<token> ::= <constant>|<name>
<label> ::= <name>
<register token> ::= %R<octit>

<source> ::= <token>
<destination> ::= <token>
<reg> ::= <token>
<adr> ::= <label>

```

General Forms

```

<assembly program> ::= <code line><end stm>|
                        <assembly program><code line><end stm>
<code line> ::= <specification part><comment part><eol>|
                *<comment part><eol>
<end stm> ::= <sps>END|<sps>END<label>

<specification part> ::= <label field><specification field>
<comment part> ::= <character>|<comment part><character>

```

Specification Forms

```

<label field> ::= <null>|<label>|<sps><label>:
<specification field> ::= <directive>|<instruction>|<macro>

<directive> ::= EQU<sps><token>|
                EQU<sps><register token>|
                SKIP<sps><constant>|
                CON<sps><constant>|
                SET<sps><constant>
<macro> ::= (not yet defined)
<instruction> ::= <simple instruction>|<double instruction>|
                <cond instruction>

```

Basic Instruction Forms

```

<simple instruction> ::= <binary immediate instruction>|

```



```

                                <extract instruction>
<double instruction> ::= <binary instruction><auxiliary instruction>|
                                <shifter instruction><auxiliary instruction>
<cond instruciton> ::= <cond specification><auxiliary instruction>

```

Specific Instruction Forms

```

<binary immediate instruction> ::=
    <binop><sps><data specification><sps><source><sep><destination>

<extract instruction> ::=
    SELECT<sep><source><sep>ROTATE<sps><token><sep><mask op><sps>
    THRU<sps><data specification><sps>INTO<sps><destination>

<binary instruction> ::= <binop><sps><source><sep><destination>

<shifter instruction> ::=
    <shift op><sps><control spec><sps><token><sep><reg>

<cond instruction> ::= IF<sps><cond spec><sps>THEN|
    IF<sps><cond spec><sps>SAVE IN<sps><reg><sps>THEN

```

Operation Forms

```

<binop> ::= CLEAR|NOR|SETZ|CXFR|COM|XOR|NAND|
    AND|EQU|NOP|XFRI|IMP|OR|ONE|
    ADI|ADC|SUI|SUB|INCR|DECR|AC0|SB0|CMP

<mask op> ::= INSERT|REPLACE

<shift op> ::= ROTATE|SHIFT

<control spec> ::= RL|R|A|LL|L|A|
    RL0|RA0|RL1|RA1|
    LL0|LA0|LL1|LA1

<cond spec> ::= ALL<sps>CONDI
    ANY<sps>CONDI
    ALL<sps>INDI
    ANY<sps>INDI
    NOT<sps>ALL<sps>CONDI
    NOT<sps>ANY<sps>CONDI
    NOT<sps>ALL<sps>INDI
    NOT<sps>ANY<sps>IND

<data specification> ::= <token>|(token,<exp spec>)
<exp spec> ::= R0|R1|C|L

```

Auxiliary Instruction Forms

```

<auxiliary instruction> ::= /<memory instruction>|/<branch instruction>

<memory instruction> ::= <memop><sps><reg><sep><adr>
<memop> ::= LD|LI|SD|SI|IM0|IM1|NOP|
    LI1|L2|L4|L8|S1|S2|S4|S8
<branch instruction> ::= BRANCH<sps><cond spec><sps>T0<sps><label>

```

SCHEMATIC OF MICROPROGRAMMING LABORATORY SYSTEM

FIGURE 1

BLOCK DIAGRAM OF JHU-4

FIG 2

10 OCT 1973

NOTES:

- ① AUXILIARY FIELD DEFINED IN FIGURE
- ② DATA FIELD DEFINED IN FIGURE
- ③ FORMATS DEFINED BY FMT FIELD, BITS 0 AND 1

INSTRUCTION WORD FORMATS

FIGURE 3

BINARY INSTRUCTION FORMAT

FIG 4 08/25/73

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0						X						

BINOP

- SPECIFIES OPERATION (SEE TABLE BELOW)

X

- SPECIFIES SOURCE FOR ALU INPUT

0

- USE OPERAND 1 BUS / ENABLE MOV

1

- USE IMMEDIATE BUS / DISABLE MOV

REG2

- SOURCE OF INPUT B

REG1

- SOURCE OF INPUT A ON BIN/MEM
DESTINATION OF RESULT

LOGICAL (BIT 2 = 0)

3	4	5	6	RESULT	NAME
0	0	0	0	0	CLEAR
0	0	0	1	$A+B$	NOR
0	0	1	0	$A\bar{B}$	SET2
0	0	1	1	$\bar{B}$	XFR
0	1	0	0	$\bar{A}\bar{B}$	—
0	1	0	1	$\bar{A}$	COM
0	1	1	0	$\bar{A}B+\bar{A}\bar{B}$	XOR
0	1	1	1	$\bar{A}\bar{B}$	NAND
1	0	0	0	AB	AND
1	0	0	1	$AB+\bar{A}\bar{B}$	EQU
1	0	1	0	A	NOP
1	0	1	1	$A+\bar{B}$	—
1	1	0	0	B	XFR
1	1	0	1	$\bar{A}+B$	IMP
1	1	1	0	$A+B$	OR
1	1	1	1	1	ONE

ARITHMETIC (BIT 2 = 1)

RESULT	NAME
$A+B$	AD
$A+B+C$	ADC
$A-B$	S4
$A-B-C$	S4B
$A+1$	INCR
$A-1$	DECR
$A+C$	ACB
$A-C$	SBB
$(A-B)$	CMP

C = CARRY BIT

(COMPARE)

NOTE 1: ALL 16 LOGICAL OPERATIONS ARE AVAILABLE.

NOTE 2: TYPE OF ARITHMETIC (1 OR 2'S COMPLEMENT) IS CONTROLLED BY A BIT IN THE STATUS REGISTER.

NOTE 3: COMPARE WILL BE EQUIVALENT TO AN SH INSTRUCTION EXCEPT THAT THE RESULT WILL NOT BE BATED INTO THE SOURCE REGISTER.

SHIFT/ROTATE INSTRUCTIONS

FIG. 5 08/25/73

AMOUNT

- SPECIFIES AMOUNT OF SHIFT/ROTATE

- SHIFT
 - ROTATE
 - LEFT
 - RIGHT
 - LOGICAL
 - ARITHMETIC
 - SHIFT FILL 0
 - SHIFT FILL 1
- } NOTE 1

REG

- SPECIFIES SOURCE REGISTER

NOTE 1: LOGICAL OPERATIONS INVOLVE ALL 32 BITS
 ARITHMETIC OPERATIONS INVOLVE BITS 1-31 LEAVING
 THE SIGN BIT UNCHANGED.

CONDITIONAL INSTRUCTION FORMAT

FIG 6 10/10/73

MASK

- SPECIFIES WHICH CONDITION OR INDICATOR CODES ARE TO BE TESTED. SEE STATUS REG FIGURE 12 FOR MEANING OF CONDITION CODES.

TYPE

- INDICATES TEST ACTION

0
1

- TEST CONDITION CODES
- TEST INDICATOR CODES

0
1

- VALID ON ANY
- VALID ON ALL

0
1

- COND = 0 ON VALID
- COND = 1 ON VALID

REG

- MAR DESTINATION IF TEST IS VALID

INSTRUCTION OPERATION:

IF THE TEST SPECIFIED IS TRUE THEN THE AOU OPERATION SPECIFIED BY THE AI FIELD WILL BE PERFORMED. IN ADDITION, THE MAR WILL BE GATED TO THE REGISTER SPECIFIED BY THE REG FIELD. THE COND CONTROL SIGNAL IS SET AS FOLLOWS

EXTRACT/INSERT INSTRUCTION FORMAT

FIG 7 08/25/73

AMOUNT - AMOUNT OF ROTATE TO BE PERFORMED

I - SPECIFIES INSERT OF EXTRACT

0 - INSERT ROTATED/MASKED RESULT INTO D-REG
1 - (EXTRACT) CLEAR D-REG BEFORE INSERTION

D-REG - DESTINATION REGISTER

S-REG - SOURCE REGISTER

IMMEDIATE EXPANDED FIELD IS USED AS A MASK

INSTRUCTION DESCRIPTION

- TEMP1, TEMP2 ARE TEMPORARY STORAGE REGISTERS
- MASK IS THE QUANTITY RESULTING FROM THE FIELD EXPANSION

14	17 18	20 21	31
OPER	REG	ADDRESS	

0 0 0 0	LD	LOAD REG DIRECT FROM 4 MEM
0 0 0 1	LI	LOAD REG INDIRECT FROM 4 MEM
0 0 1 0	SD	STORE REG DIRECT TO 4 MEM
0 0 1 1	SI	STORE REG INDIRECT TO 4 MEM
0 1 0 0	IM0	LOAD REG IMMEDIATE (0 FILL)
0 1 0 1	IM1	LOAD REG IMMEDIATE (1 FILL)
0 1 1 0	BRP	BRANCH POSITIVE
0 1 1 1	BRN	BRANCH NEGATIVE
1 0 0 0	BL1	BLOCK LOAD 1 WORD FROM MM
1 0 0 1	BL2	" " 2 WORDS " "
1 0 1 0	BL4	" " 4 " " "
1 0 1 1	BL8	" " 8 " " "
1 1 0 0	BS1	BLOCK STORE 1 WORD TO MM
1 1 0 1	BS2	" " 2 WORDS " "
1 1 1 0	BS4	" " 4 " " "
1 1 1 1	BS8	" " 8 " " "

USES SPECIAL FORMAT

FORMAT OF AUXILIARY OPERATION FIELD

SEMANTICS OF AUXILIARY OPERATIONS

NOTES: ① ADR' IS ADDRESS CONTAINED IN REGISTER DESIGNATED BY REG FIELD

- BRP - IF VALID BRANCH TO MAR + AMOUNT
 - BRN - IF VALID BRANCH TO MAR - AMOUNT

- MASK FOR BRANCH CODE TEST

- INDICATES TYPE OF
 CODE TEST TO BE DONE

- TARGET OFFSET FROM
 CURRENT ADDRESS

NOTE: TESTING FOR BRANCH OPERATIONS IS EXACTLY THE
 SAME AS THAT DESCRIBED FOR THE CONDITIONAL
 INSTRUCTION (FIGURE)

FORMAT OF AUXILIARY OPERATIONS BRP AND BRN FIGURE 9
10/11/73

OUTPUT OF FIELD EXPANSION UNIT

FIELD EXPANSION AS CONTROLLED BY AI FIELD FIGURE 10
10/11/73

MEMORY ADDRESS STRUCTURE

- BRP - IF VALID BRANCH TO MAR + AMOUNT
 - BRN - IF VALID BRANCH TO MAR - AMOUNT

- MASK FOR BRANCH CODE TEST

- INDICATES TYPE OF
 CODE TEST TO BE DONE

- TARGET OFFSET FROM
 CURRENT ADDRESS

NOTE: TESTING FOR BRANCH OPERATIONS IS EXACTLY THE
 SAME AS THAT DESCRIBED FOR THE CONDITIONAL
 INSTRUCTION (FIGURE)

FORMAT OF AUXILIARY OPERATIONS BRP AND BRN FIGURE 9
10/1/73

OUTPUT OF FIELD EXPANSION UNIT

FIELD EXPANSION AS CONTROLLED BY AI FIELD FIGURE 10
10/1/73

MEMORY ADDRESS STRUCTURE

C H M L P B

- CONDITIONS SET BY THE CONDITIONAL EXAMINATION UNIT

INDICATOR

- STORAGE BITS MODIFIED ONLY BY THE PROGRAMMER

STATE

- CURRENT MACHINE STATE

R C I A

INTR-ADDRESS

TARGET ADDRESS FOR
INTERRUPT PROCESSING

FORMAT OF STATUS REGISTER FIGURE 12