The Brown University Graphics System[1]

SIMALE Principles of Operation

The Brown University Graphics Project

Division of Applied Mathematics

Box F

Brown University

Providence, Rhode Island 02912

June 1975

Printed: June 13, 1978

---

[1]This Research is being supported by the National Science Foundation Grant GJ-41539, the Office of Naval Research, Contract N00014-75-C-0427, and the Brown University Division of Applied Mathematics; Principal Investigator Andries van Dam.

TABLE OF CONTENTS

1 OVERVIEW

The SIMALE is a processing element of the BUGS system capable of performing a varied and complex repetoire of graphics operations for the user. Its high speed gives BUGS the performance typical of graphics systems employing special purpose hardware but none of the inherent inflexibility. Although anyone could program the SIMALE to meet the needs of his application, it is presumed most users will use a standard set of SIMALE graphics operations embedded within a higher level language. The discussion that follows, therefore, is intended for the more serious user.

2 SIMALE COMPONENTS

The SIMALE's architecture departs radically from the more traditional ones of the Meta 4A and Meta 4B in several respects. These departures were dictated by the special nature of the graphics operations it performs. Speed, flexibility, simplicity, reliability, and cost were all important design considerations affecting its architecture.

The most noticable features of the SIMALE are its abundances of processors, stores, and data buses (see figure 1). The SIMALE has four parallel processors, quadrupling its effective speed, and unfortunately making it difficult to understand and program.

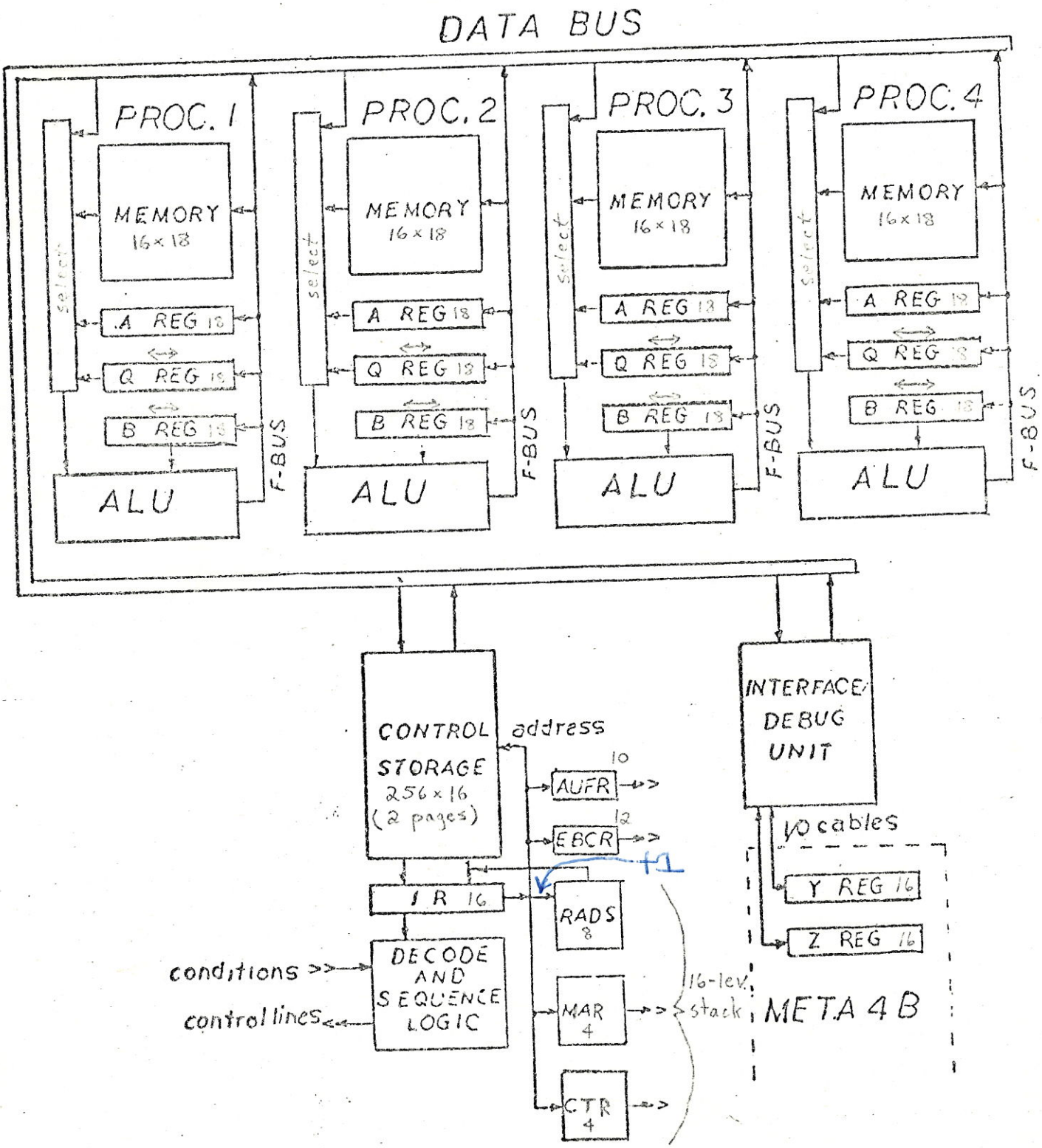
2.1 SIMALE STORES

The SIMALE has four general types of stores; two in the control and two in the processing units.

CONTROL STORE: This is a fully readable/writeable store in the control unit and is used primarily to hold instructions. It consists of 256, 16-bit words organized into two, 128-word pages which are loaded by the Meta 4B during execution of the ETC instruction. Control storage locations may be read or written under SIMALE program control, although the addressing mechanism makes doing so difficult.

CONTROL REGISTERS: The SIMALE makes heavy use of residual control; a technique where the loading of special control registers determines the nature of subsequent operations. The control unit has five such registers; the CounT Register (CTR), the Memory Address Register (MAR), the Enable and Bus Control Register (EBCR), the Request Code Register (RCR), and the ALU Function Register (AUFR). These registers control such things as program looping, the sources and destinations during data transfers, and the operands and operations performed by the four processors. Their existence greatly increases the power of the SIMALE's small instruction set while avoiding a

FIG 1 – SIMALE ARCHITECTURE



more expensive and inefficient "horizontal" architecture.

PROCESSOR MEMORY: Each processor possesses sixteen, 18-bit words of memory used as a "scratch-pad" to hold less frequently used operands. The memories are addressed by the MAR control register. All operand stores in the SIMALE are 18 bits to provide the precision necessary to avoid overflow problems arising during certain graphics operations. The eighteen bits of these stores are labeled (X0),(X1),(0),(1),...,(15) to emphasize their positional relationship with the sixteen bits of control storage and the Meta 4B.

PROCESSOR REGISTERS: In addition to the sixteen memory locations, each processor has three 18-bit registers; the A, Q, and B registers. Two of these registers are shiftable and all are used for intermediate results and frequently used data during computations.

2.2 SIMALE PROCESSORS

The SIMALE has five logical processing units: one processes instructions, and four process operand data.

INSTRUCTION PROCESSOR: All instructions in the SIMALE are processed by the control unit. The execution of an instruction can cause activity on all of the four operand processors simultaneously. This parallel execution of instructions makes the SIMALE an efficient machine for graphical operations.

OPERAND PROCESSORS: Each of the processing units has an 18-bit ALU (Arithmetic/Logic Unit) which selects operands from among the various stores and performs an arithmetical or logical operation upon them. The AUFR control register determines the operations performed by the ALU'S. The results are made available on the F-buses where they can be copied into a store or tested by subsequent instructions.

2.3 THE DATA BUS

The SIMALE transfers data among its various processing units and the Meta 4B via a bidirectional 18-bit data bus; the Z-bus. During a data transfer, one of eight sources and one of seven destinations are selected by the EBCR control register. Among them are the Meta 4B, control storage, the low order byte of the current instruction, and the processor F-buses. 16-bit sources (such as the Meta 4B and control storage) have their signs extended onto the bus. When the instruction register is source, bits(8-15) of the current instruction are copied onto the corresponding bits of the Z-bus and the rest are forced to zero. An additional data path serves to pass command, status, and debugging information between the Meta 4B and the SIMALE's control unit.

3 THE SIMALE INSTRUCTION SET

The SIMALE has sixteen instructions, all of which are 16-bits long. Almost all instructions are divided into three fields; an operation code in bits(0-3), a modifier or data field in bits(4-7), and an address field in bits(8-15). While most SIMALE instructions are simple register loads, they can be difficult to understand because several things modify their behavior.

First, the loading of residual control registers determines the operation of subsequent instructions. For example, the setting of the AUFR register might instruct the ALU's to perform addition on the B and Q registers. If an instruction then loaded the A registers, they would contain the result of this addition. But for other settings of control registers, this same instruction could zero the A register, perform a logical OR'ing of two registers, or complete a data transfer with the Meta 4B. In addition, many of the control registers are on top of stacks and can assume different values on different levels of program execution!

Besides being affected by residual control registers, the execution itself of many instructions is conditional. Condition codes referenced by these instructions test one of sixteen conditions throughout the SIMALE and modify execution behavior according to the result. See Section 3.4 for a further explanation of the condition codes.

In the instruction descriptions that follow, upper case symbols are predefined by the SIMALE assembler to the hexadecimal values annotated (#hex value) and optional symbols are enclosed in braces. All symbols are delimited by either blanks or commas. To produce object code, the assembler simply OR's together the values of the symbols appearing on a line of code. For more detail, see the SIMALE Assembler Manual.

3.1 PROCESSOR INSTRUCTIONS

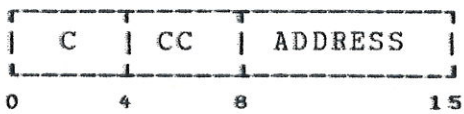
Instructions with operation codes 8 through 15 affect the operand stores of the four processors. All interpret bits(4-7) as a condition code and bits(8-15) as a branch address unless the condition code specified pre-empts taking this branch. The EBCR control register enables the processors, and no

action is ever taken by a disabled processor or by a processor which is not a bus destination during a data transfer (see the XFER condition code).

Write Memory from F-bus on condition
WM <condition code>[,<label>]

Load Q register from F-bus on condition
LQ <condition code>[,<label>]

Load A register from F-bus on condition
LA <condition code>[,<label>]



Load B register from F-bus on condition
LB <condition code>[,<label>]

For these instructions, each enabled processor takes the local value of the condition code specified and if true, copies the current ALU output from the F-bus into the specified 18-bit store. In the case of a Write Memory instruction, the exact memory location written is specified by the Memory Address Register (MAR).

Unless modified by the condition code, execution resumes at the address specified by <label>.

Left SHift Q register, entering condition value
LSHQ <condition code>[,<label>]

Right SHift Q register, entering condition value
RSHQ <condition code>[,<label>]

Left SHift B register, entering condition value
LSHB <condition code>[,<label>]

Right SHift B register, entering condition value
RSHB <condition code>[,<label>]

For these instructions, each enabled processor shifts the specified register one position in the specified direction, entering the local value of the condition code into the emptied end bit position.

Unless modified by the condition code, execution resumes at the address specified by <label>.

3.2 CONTROL REGISTER INSTRUCTIONS

The following instructions, with operation codes 2 through 6, load and modify the residual control registers.

Load Memory Address Register with immediate
LMAR <data>[,<label>]

This instruction takes the data specified in bits (4-7) and loads it into the 4-bit Memory Address Register (MAR).

Execution resumes at the address specified by <label>.

The MAR is a control register whose contents selects one of the sixteen locations in the processor memories. It sits atop a 16-level stack in which the current value of the MAR may be saved by a CALL instruction and from which the MAR may be restored by a RETURN instruction.

The MAR can be incremented by a MODR instruction (see Section 3.3).

Load CounT Register with immediate
LCTR <data>[,<label>]

This instruction takes the data specified in bits (4-7) and loads it into the 4-bit CounT Register (CTR).

Execution resumes at the address specified by <label>.

The CTR is a control register used to control program looping. It sits atop a 16-level stack in which the current value of the CTR may be saved by a CALL instruction or from which the CTR may be restored by a RETURN instruction.

The CTR can be decremented and tested by the CounT (CT) and Not CounT (NCT) condition codes (see Section 3.4).

Load ALU Function Register with immediate
LAUFR <function>[,INV][,RND][,+1]

This instruction takes the immediate data specified in bits(4-15) and loads it into the 12-bit ALU function register.

Execution resumes at the next sequential instruction.

The ALU function register is a control register selecting the operand stores and the operation performed by the ALU's of the processors. There are several fields within the register. The 5-bit field in bits(11-15) selects one of thirty-two arithmetic or boolean operations performed upon the two source stores selected by the fields in bits(4-7). A description of all thirty-two functions can be found in the SIMALE hardware Princiles of Operation manual; those currently defined for the assembler are:

<function>:

F=<s1> (#0000): This copies s1 onto the F-bus.

F=<s1> OR <s2> (#0001): This logically OR's the two sources.

F=<s1> OR_NOT <s2> (#0002): This OR's s1 with the complement of s2.

F= ONES (#0003): This sets the F-bus to all ones.

F=<s1> AND <s2> (#0004): This AND's the two sources.

F=<s1> XNOR <s2> (#0006): This is the complement of the exclusive OR of the two sources.

F=<s1> AND_NOT <s2> (#0008): This AND's s1 with the complement of s2.

F=<s1> XOR <s2> (#0009): This XOR's s1 with s2.

F= NOT <s2> (#000A): This complements s2.

F=<s1> NAND <s2> (#000B): This NAND's s1 and s2.

F= ZEROS (#000C): This sets the F-bus to all zeros.

F=<s1> NOR <s2> (#000E): This NORS s1 with s2.

F=<s1> +0 (#0010): This adds zero plus the carry in to s1.

F= NEG_ONE (#0013): This sets the F-bus to all ones plus the carry in.

F=<s1> MINUS <s2> (#0016): This sets the F-bus to s1 minus s2 minus one plus the carry in.

F=<s1> PLUS <s2> (#0019): This sets the F-bus to s1 plus s2 plus the carry in.

F=<s1> TIMES_2 (#001C): This sets the F-bus to s1 left shifted one place.

F=<s1> -1 (#001F): This sets the F-bus to s1 minus one plus the carry in.

<s1> : The first operand source for the above functions is selected by bits(4,5). the possible sources are:

F=MEM (#0000): The memory

F=A (#0400): The A register

F=Q (#0800): The Q register

F=B (#0C00): The B register

<s2>> : The second operand source for the above functions is selected by bits(6,7). The possible sources are:

MEM (#0000): The memory

A (#0100): The A register

Q (#0200): The Q register

B (#0300): The B register

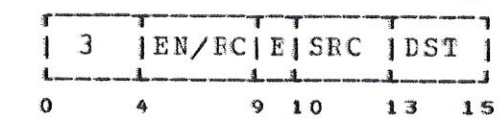
INV (#0080): Bit(8) is the invert bit (I) and inverts the carry in and function bits(12-15) seen by processors 2 and 3. This has the effect of making the ALU's of processors 2 and 3 perform the complement of the function performed on processors 0 and 1. Thus it is possible, for example, to do addition on

processors 0 and 1 and at the same time do subtraction on processors 2 and 3.

RND (#0040): Bit(9) is the round bit (R) and if set, the carry in to any processor that has right shifted a one bit out of its B register is inverted. This prevents roundoff errors during certain operations.

+1 (#0020): Bit(10) is the carry bit (C) into the ALU's during arithmetic operations. It may be inverted either by the round or the invert bits and has the effect of adding one to the results of arithmetic operations.

Load Enable, Bus Control Register
with immediate
LEBCR <enable or request><bus src><bus dst>



This instruction takes the immediate data specified in bits(4-15) and loads it into the Enable and Bus Control Register (EBCR).

Execution resumes at the next sequential address.

The EBCR is a control register which deals with enabling processors and with data transfers on the Z-bus. The LEBCR instruction does not actually perform a data transfer but rather determines what transfer will be performed by the XFER condition code.

<enable> : When a LEBCR instruction is executed with bit(9) (E) set, bits(4-7) are loaded into a 4-bit enable register each bit of which enables one of the processors. Usually all processors are left enabled, but when not, disabled processors are unaffected by instructions and cannot be tested by condition codes. The MODR instruction can rotate right the enable bits one place. The enable register bits are:

P0 (#0840): enables Processor 0

P1 (#0440): enables Processor 1

P2 (#0240): enables Processor 2

P3 (#0140): enables Processor 3

ALL (#0F00): enables Processors 0-3.

<request> : When a LEBCR instruction is executed and bit(9) (E) is off, bits(4-8) are loaded into the 5-bit Request Code Register. This register is used by the SIMALE to request an operation of the Meta 4B. When an XFER condition code causes the SIMALE to pause for I/O with the Meta 4, the Q-interpreter examines the request code register and the CTR. It then performs the requested operation, sometimes using the CTR as a modifier. Only sixteen of the thirty-two possible request codes are currently implemented in the Meta 4B firmware, and requesting illegal codes causes a SIMALE interrupt . The possible request codes are:

QUIT (#0000): causes termination of the current ETC instruction.

INTERRUPT (#0080): causes a SIMALE interrupt, then quits.

NEXT_BLOCK (#0010): causes the termination of the current ETC block and resumes execution at the next block.

NEXT_SUB_BLOCK (#0180): causes termination of the current ETC sub-block and resumes execution at the next sub-block.

GOTO_PAGE (#0200): The SIMALE gives the Meta 4B a SIMALE initialization halfword via the Z-bus. The Meta 4B then loads the requested control store page if necessary and starts SIMALE execution at the requested location. This is an inter-virtual page jump (see Section 16.3 of the Meta 4B manual).

GET_DATA (#0280): causes the Meta 4B to get the CTR number of halfwords from the current sub-block and give them one at a time to the SIMALE via the Z-bus. If the CTR is zero, an interrupt is caused and execution is halted.

SET_VG_MODE (#0300): causes the Meta 4B to issue a mode order to the VG which it reads from the SIMALE Z-bus.

SEND_VG_DATA (#0380): causes the Meta 4B to send the VG the CTR number of halfwords from the SIMALE Z-bus. If the CTR is zero, an interrupt is caused and execution is halted.

GET_REG (#0400), PUT_REG (#0480), GET_LS (#0500), PUT_LS (#0580), GET_MS (#0600), PUT_MS (#0680), GET_VG_REG (#0700), PUT_VG_REG (#0780): All of these request codes cause the Meta 4E to read an address from the SIMALE's Z-bus. It then transfers the CTR number of halfwords to or from the specified stores (ie. user REGisters, Local Store, Main Store, or VG REGisters). If the CTR is zero, an interrupt is caused and execution is halted.

<bus src>: Bits(10-12) of the FBCR select the Z-bus source. They sit atop a 16-level stack in which they may be saved by a CALL instruction and from which they may be restored by a RETURN instruction. The bus source bits may be incremented by the MODR instruction. Possible sources are:

FROM_M4 (#0000): Meta 4 is source

FROM_CS (#0008): Control Store is source

FROM_IR (#0010): The low order byte of the instruction is source

FROM_ALL (#0018): The F-buses of all processors are OR-ed together and are the source

FROM_P0 (#0020): The F-bus of Processor 0 is source

FROM_P1 (#0028): The F-bus of Processor 1 is source

FROM_P2 (#0030): The F-bus of Processor 2 is source

FROM_P3 (#0038): The F-bus of Processor 3 is source

<bus dst>: Bits(13-15) of the EBCR select the Z-bus destination. As with the bus source field, the bus destination bits are atop a 16-level stack and may be incremented by the MODR instruction. possible destinations are:

TO_M4 (#0000): Meta 4B is the destination

TO_CS (#0001): Control Store is the destination. Bits(8-15) of the instruction invoking the data transfer (ie. specifying xfer) is the address of the location loaded from the Z-bus.

TO_ALL (#0003): The Z-bus contents is copied onto the F-buses of all processors, which are not bus source, regardless of the AUFR setting.

TO_P0 (#0004): The Z-bus is copied onto processor 0's F-bus, if not bus source.

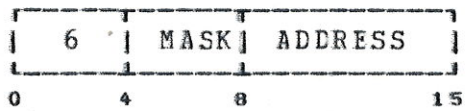
TO_P1 (#0005): the Z-bus is copied onto processor 1's F-bus, if not bus source.

TO_P2 (#0006): the Z-bus is copied onto processor 2's F-bus, if not bus source.

TO_P3 (#0007): the Z-bus is copied onto processor 3's F-bus, if not bus source.

MODify Registers

MODR [MAR][,ENABLE][,SRC][,DST][<,label>]



The Modify Registers instruction uses bits(4-7) as a mask to determine which of four control registers to modify as follows:

MAR (#0800): If bit(4) is set, the Memory Address Register is incremented.

ENABLE (#0400): If bit(5) is set, the Enable field of the EBCR bits(4-7), is rotated right one position. EBCR bit(7) is entered into EBCR bit(4) when this is done.

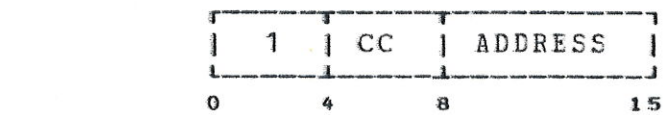
SRC (#0200): If bit(6) is set, the source field of the EBCR is incremented if the source specified is one of the processors. Processor 3 wraps around to Processor 0.

DST (#0100): If bit(7) is set, the Destination field of the EBCR is incremented if the destination specified is one of the processors. Processor 3 wraps around to processor 0.

3.3 FLOW OF CONTROL INSTRUCTIONS

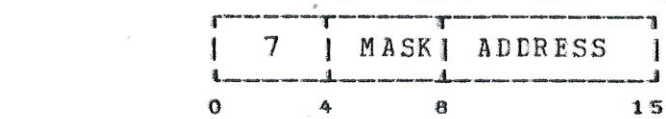
The SIMALE has three instructions which alter the flow of control in programs. All of them interpret bits(8-15) as an address. These instructions are:

BRanch on condition true
ER <condition code>[<,label>]



The Branch instruction tests the global value of the condition code specified (see Section 3.4). If it is true, execution resumes at the address specified by <label>, otherwise execution resumes at the next sequential instruction.

CALL subroutine
CALL [MAR][,CTR][,SRC][,DST][<,label>]



The call instruction saves the next sequential address in a 16-level stack internal to the control unit. This serves as link information used by the RETURN instruction.

Execution is resumed at the address specified by <label>.

The CALL instruction also affects those control registers that are on stacks. The values of these control registers are constantly being copied into the present level of their stacks. When a CALL is executed, the stack frame pointer is

incremented, accessing the next level of the stacks, then bits(4-7) determine the subsequent setting of these control registers as follows:

MAR (#0800): If bit(4) is set, the Memory Address Register's current value remains unchanged. If bit(4) is not set, the MAR is loaded from the new stack level and thus assumes the value it had previous to the last RETURN .

CTR (#0400): If bit(5) is set, the Count Register remains unchanged. If not set, the CTR assumes its previous value.

SRC (#0200): If bit(6) is set, the source field of the EBCR remains unchanged. If not set, it assumes its previous value.

DST (#0100): If bit(7) is set, the destination field of the EBCR remains unchanged. If not set, it assumes its previous value.

RETURN from subroutine on condition true.
RETURN <condition code>[<,label>]

The RETURN instruction tests the global value of the condition code specified. If it is true, the stack frame pointer is decremented and the control registers are restored to their values previous to the last CALL instruction. Execution is resumed at the saved address. If the condition code is not true, execution is resumed at the address specified by <label>.

3.4 CONDITION CODES

The SIMALE conditional instructions specify a condition code in bits(4-7). All codes select one of sixteen conditions tested in five places; in the control unit, and in each of the four processors. The value tested in the control unit is called the global value, and the values tested in the processors are called the local values. In many cases, the global value is simply the logical OR'ing of the four local

values. Disabled processors make no contribution to the global values so obtained.

Four of the condition codes can modify the operation of processor-type instructions. This modification usually involves invoking special cycle sequences or altering where execution is resumed upon completion. The condition codes are:

FALSE (#0000): This doesn't modify instruction sequencing

local value: always false

global value: always false

TRUE (#0100): This doesn't modify instruction sequencing

local value: always true

global value: always true

TREJ (#0200): This is a windowing rejection test for a line whose end point window coordinates are in the A and Q registers. If both of these registers are negative on any one processor, then the line is absolutely outside the window. TREJ doesn't modify instruction sequencing.

local value: (A(X0) AND Q(X0)) from all enabled processors OR'd together

global value: same as local value.

REJ (#0300): This is a windowing rejection test for a line whose window coordinate end points are in the A register and on the F-bus. If both are negative on any one processor, the line is absolutely outside the window. REJ doesn't modify instruction sequencing.

local value: (A(X0) AND F(X0)) from all enabled processors OR'd together

global value: same as local value

DELAY (#0400): This inserts a one cycle delay before the execution of a processor instruction to aid fixing possible program timing problems.

local value: always true

global value: always true

XFER (#0500): When referenced by a processor instruction, XFER triggers a several cycle sequence causing a data transfer along the Z-bus. The transfer is from the source to the destination as specified in the EBCR. If the Meta 4B is involved, the SIMALE pauses until the Meta 4 can signal the transfer is complete. Only those processors which are destinations can be affected by an instruction specifying XFER, and only those which are sources pay attention to the setting of the AUFR during the transfer.

Execution is always resumed at the next sequential instruction.

local value: always true

global value: always true

CT (#0600): The CounT condition code always causes the CTR to be decremented. If it is specified by a processor instruction, and the CTR was one before being decremented, execution is forced to resume at the next sequential instruction; otherwise it resumes at the address specified. This gives processor instructions a BCT ability.

local value: always true

global value: CTR not = 1 (i.e. goes false when the count is exhausted)

NCT (#0700): The Not Count condition code always causes the CTR to be decremented. If it is specified by a processor instruction, and the CTR was one before being decremented, execution is forced to resume at the next sequential instruction; otherwise it resumes at the address specified. This gives processor type instructions a BCT ability.

local value: always false

global value: CTR = 1 (i.e., goes true when the count is exhausted)

COUT (#0800): This doesn't modify instruction sequencing.

local value: Carry OUT from ALU bit(X0)

global value: carry out from all enabled processors OR'd together.

NF(X0) (#0900): This doesn't modify instructions sequencing.

local value: the complement of the sign bit of the F-bus

global value: NF(X0)'s of all enabled processors OR'd together

LSIG (#0A00): This doesn't modify instruction sequencing. LSIG is a test for left significance of a processor's B register (i.e., either its left most two bits are different or it is all ones or zeros)

local value: (B(X0) XOR B(X1)) OR (B NOT MIXED)

global value: LSIG's of all enabled processors OR'd together.

BM (#0B00): This doesn't modify instruction sequencing. B mixed is a test for a processor's B register not being all ones or all zeros.

local value: B register mixed

global value: BM of all enabled processors OR'd together.

Q(X0) (#0C00), Q(15) (#0D00), B(X0) (#0E00), B(15) (#0F00): These don't modify instruction sequencing they are used to test the end bits of the B and Q register.

local value: the bit specified

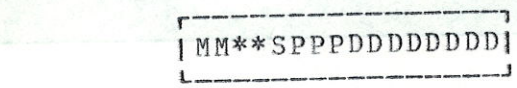
global value: the bit specified of all enabled processors are OR'd together.

1 APPENDIX I -- SIMALE INTERFACE REGISTERS

The Meta-4B is provided with two registers, named SY and SZ. The SY register is used to pass control information between the SIMALE and the "B", and the SZ register is used to pass data between the machines. The contents of the SZ register are controlled by the source and destination fields in the SY register.

1.1 SY REGISTER

1.1.1 ON OUTPUT



1.1.1.1 Mode (MM):

00 --> Read and go (useless)
01 --> Load and go
10 --> Control Storage Read
11 --> Control Storage Write

1.1.1.2 Step (S):

If this bit is on, the SIMALE steps the number of cycles in the clock counter.

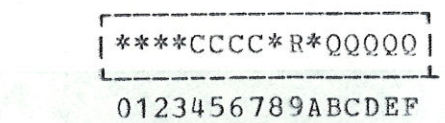
1.1.1.3 Source/Destination (PPP):

000 --> normal execution
001 --> read,write backplane pins
010 --> read,write backplane pins
011 --> backplane on input, no output
100 --> backplane on input, no output
101 --> backplane on input, no output
110 --> backplane on input, no output
111 --> backplane pons in input, clock counter on output

1.1.1.4 Data Byte (DDDDDDDD):

Address or data

1.1.2 INPUT IN NORMAL MODE



1.1.2.1 Request Code Field (QQQQQ):

0 --> Quit
1 --> Interrupt (Count field contains code)
2 --> Next Block
3 --> Next Sub-block
4 --> Goto Page[0]
5 --> Get Data [1]
6 --> Set VG Mode[0]
7 --> Send VG Data[1]
8 --> Get Register[2]
9 --> Put Register[2]
10 --> Get Local Store[2]
11 --> Put Local Store[2]
12 --> Get Main Store[2]
13 --> Put Main Store[2]
14 --> Get VG Register[2]
15 --> Put VG Register[2]
16 --> CALL

17 --> RETURN
18 --> PUSH[1]
19 --> POP[1]

([0]) count ignored, data in SZ register
([1]) count valid, data transfer thru SZ
([2]) count valid, SIMALE writes addr. thru SZ before data

1.1.2.2 Count Field (CCCC):

Indicates no. of words to transfer.

1.1.2.3 Run Flag (R):

If the run flag is off, SIMALE is pausing for a request.