Inter-Office Memorandum

To	Mesa Users	Date	May 31, 1978
From	Roy Levin	Location	Palo Alto
Subject	Mesa 4.0 Microcode Update	Organization	CSL

XEROX

Filed on: [IRIS]<MESA>DOC>MICROCODE40.BRAVO

This memo outlines the differences in the (Alto) Mesa microcode for release 4.0. The *OIS Processor Principles of Operation* [1] ("PrincOps") is scheduled to be revised soon, and some of the changes indicated below will be incorporated in that revision. Others are peculiar to the Alto implementation of Mesa and are indicated as such. This document is only a summary of the changes in Mesa 4.0; additional details may be found in the references cited at the end of this memo.

Definitions of New Notions

Two new instruction properties have been introduced in the Mesa 4.0 instruction set:

Alignment

An aligned 1-byte instruction must be the last significant byte in the word. Thus, if an aligned, 1-byte instruction appears in an even byte position, the microcode will ignore the contents of the odd byte in the same word. An aligned 2-byte instruction must have both bytes in the same memory word. An aligned 3-byte instruction consists of an aligned 1-byte opcode followed by a word containing the α and β bytes. In this case, the α byte must be in the *odd* byte of the word following the opcode. In the 2-byte case, padding is accomplished by use of the new instruction NOOP, which is discussed below.

Minimal stack

A minimal stack instruction expects its operands to be the only quantities on the stack, and leaves the stack empty, except for any results it explicitly supplies.

These properties are peculiar to the Alto implementation of Mesa and will *not* be included in the revision of the PrincOps.

Changes to the Instruction Set

Many of the changes in the Mesa 4.0 microcode bring the instruction set closer to the PrincOps. In some cases, however, constraints imposed by the Alto architecture have prevented an exact emulation of the PrincOps semantics. The following sections define the differences between the Mesa 3.0 and 4.0 instruction sets, and relate those differences to the PrincOps.

Mesa 3.0 bytecodes not present in Mesa 4.0

LGS, SGS, LLS, SLS
LGDS, SGDS, LLDS, SLDS
WSDS

Space constraints in the Alto implementation have forced the elimination of these bytecodes.

ADDL, ADDG

These instructions have been superseded by the PrincOps instructions LADRB and GADRB (see below).

RXL0-3, WXL0
RIG0-3, WIG0, WIL0

These instructions have been superseded by the PrincOps instructions RXLP, WXLP, RIGP, RILP, and WILP (see below). Note that RIL0 has been retained, because of its high static frequency.

RIL1-3

Space constraints in the Alto implementation have forced the elimination of these bytecodes. They will, however, remain in the PrincOps when it is revised. Note that RIL0 has been retained in the Alto implementation.

Jumps

Mesa 4.0 has adopted byte distances for specifying jump targets. As a result, the even and odd byte forms of the Mesa 3.0 jump instructions have been eliminated, and the entire set of jumps revised to conform almost completely to the PrincOps. JIB and JIW are the only Mesa 3.0 jumps that have been retained, though their semantics have been modified to agree with the notion of byte distances. Details appear below.

GFC0-15, GFCB

The GFCn instructions have been uniformly replaced with EFCn instructions, which have similar semantics but support destination links in either the global frame or the code segment. This substitution will occur in the revision of the PrincOps as well.

CVT

Space constraints in the Alto implementation have forced the elimination of CVT.

BLK

The Mesa 4.0 process machinery makes BLK obsolete, and it has been eliminated. It will be eliminated in the revision of the PrincOps as well.

Mesa 4.0 bytecodes not present in Mesa 3.0

NOOP

Introduced to accommodate alignment requirements of the Alto implementation. NOOP will not be incorporated in the PrincOps revision. See the section on "Interrupts" for additional requirements affecting the execution of NOOP. Note: this opcode is meaningful only when it appears in the odd byte of a word.

LADRB, GADRB

Behave as described in [1], except:
Both are aligned instructions.

This difference will not be included in the PrincOps revision.

ADD01

Identical to ADD as described in [1], except:
ADD01 is a minimal stack instruction, assuming precisely 2 elements on the stack.

It is possible that ADD01 will be included in the PrincOps revision.

DADD, DSUB, DCOMP

DADD and DSUB behave as described in [1], except:
Both require minimal stack.
No carry bit is left on stack above stack pointer.

DCOMP expects two double precision values on the stack. Call them A and B. DCOMP performs: Sign[DSUB[A,B]]. Sign(x) = {-1 if x<0, 0 if x=0, +1 if x>0}. The (single precision) result of the Sign is left on the stack. DCOMP has a *minimal stack requirement.* DCOMP will probably be included in the PrincOps revision.

RXLP, WXLP
RILP, RIGP, WILP

Behave exactly as described in [1].

RFS, WFS
RFC

RFS and WFS are aligned, 1-byte instructions that expect the top word of the stack to be an $\langle\alpha, \beta\rangle$ pair, with α in the left byte. RFS and WFS pop this word from the stack, then behave exactly like RF and WF, respectively, using the α and β values obtained from the stack word.

RFC is an aligned, 3-byte instruction that is identical to RF in all respects except that the code base register, C, is added to the computed address before the field is accessed.

All of these instructions use a new field descriptor format, which is described in the following section under RF, WF, and WFS. All of these instructions will be included in the PrincOps revision.

J2-J9, JB, JW

Behave as described in [1], except:
JB and JW are aligned instructions.
All jump distances are signed values, measured from the last byte of the jump instruction instead of the first (as in the PrincOps).

These differences will not be included in the PrincOps revision.

JEQ2-9, JEQB
JNE2-9, JNEB

Behave as described in [1], except:
JEQB and JNEB are aligned instructions.
All jump targets are signed, PC-relative distances in bytes, measured from the last byte of the jump instruction instead of the first (as in the PrincOps).

These differences will not be included in the PrincOps revision.

JLB, JGEB, JGB, JLEB
JULB, JUGEB, JUGB, JULEB
JZEQB, JZNEB

Behave as described in [1], except:
All are aligned instructions.
All jump targets are signed, PC-relative distances in bytes, measured from the last byte of the jump instruction instead of the first (as in the PrincOps).

These differences will not be included in the PrincOps revision.

DESCB, DESCBS

Behave as described in [1], except:
Both are aligned instructions.
The result left on the stack is (gfiword $\wedge$ 177B)+2*α+1, *where* gfiword *is word 0 of the global frame used by the instruction (see the section on "Global Frame Format", below).*

The difference in the result produced by these instructions will be included in the PrincOps revision.

EFC0-15, EFCB
LLKB

EFC0-15 and EFCB replace the GFC0-15 and GFCB instructions of Mesa 3.0. EFCn behaves identically to GFCn except in the way the destination link is determined. To locate the destination link, the microcode examines the low-order bit of the gfi word (word 0) of the global frame. If this bit is 0, the destination link is taken from location G-n-1, where G is the address of the current global frame. If the bit is 1, the destination link is taken from location C-n-1, where C is the address of the current code segment. EFC0-15 and EFCB will preplace GFC0-15 and GFCB in the PrincOps revision.

LLKB is an aligned, 2-byte instruction that computes a destination link in the same way that EFCB does. Instead of using it as the destination of an Xfer, however, LLKB simply pushes the destination link on the stack, and performs no additional actions upon it. LLKB, with the alignment requirement dropped, will be included in the PrincOps revision.

ME, MRE, MXW, MXD, NOTIFY, BCAST, REQUEUE

These are process-related opcodes, and are described separately below. All will be included in the PrincOps revision.

Bytecodes whose semantics have changed from Mesa 3.0 to Mesa 4.0

LGDB, SGDB, LLDB, SLDB
LIW

Identical to Mesa 3.0, except:
All are aligned instructions.

This difference will not be included in the PrincOps revision.

JIB, JIW

Identical to Mesa 3.0, except:
Both are aligned instructions.
The jump target for JIB is an unsigned distance in bytes, measured from the last byte of the JIB instruction instead of the first (as in the PrincOps).

The jump target for JIW is a signed distance in bytes, measured from the last byte of the JIW instruction instead of the first (as in the PrincOps).

These differences will not be included in the PrincOps revision.

RDB, WDB
WSB, WSDB

Identical to Mesa 3.0, except:
All are aligned instructions.

This difference will not be included in the PrincOps revision.

RSTR, WSTR

Identical to Mesa 3.0, except:
All are aligned instructions.

This difference will not be included in the PrincOps revision.

RF, WF, WSF

Identical to Mesa 3.0, except:
All are aligned instructions.
Field descriptor in β byte is <p,s>, where p = bits to the left of desired field and s = bits in field minus 1.

The difference in field descriptor format will be included in the PrincOps revision.

BITBLT

Identical to Mesa 3.0, except:
BITBLT is an aligned instruction.
BITBLT is a minimal stack, 2-argument instruction.
The first argument to BITBLT is unchanged from Mesa 3.0; the second is a word containing the value zero.
BITBLT may be interrupted and subsequently restarted.

These differences are not relevant to the PrincOps revision.

DST, LST, LSTF

Identical to Mesa 3.0, except:
All are aligned instructions.
Only the active portion of the stack is saved or loaded (including 2 words above the top of the stack).

Except for the alignment requirement, these differences will be included in the PrincOps revision.

RR, WR

All α byte interpretations have changed - see the section on Trap Handlers and Parameters, below.

These differences will not be included in the PrincOps revision.

Process Instructions and the Mesa/Nova Interface [4]

Entry points to the Mesa emulator

Three entry points are defined. Control is transferred to these entry points by means of the Nova JMPRAM instruction. The addresses of these entry points are unchanged from Mesa 3.0, but some of the necessary conditions at entry are different.

a) Entry point 'Mgo' (location 420B): Nova AC0 must contain the address of a process state vector. The emulator will load the process state from this address, then perform a control transfer using the destination link at <AC0>+11B and source link at <AC0>+12B.

b) Entry point 'Minterpret' (location 400B): Mesa emulation continues using the current state in the emulator's internal registers. It is the responsibility of the invoker to ensure that the proper process state has been loaded.

c) Entry point 'SWRET' (location 777B): Used only when the Mesa emulator resides in ROM1. See Alto Hardware Manual, section 9.2.4.

Exits to Nova code

The Mesa emulator may cease execution and transfer control to the Nova emulator for one of two reasons:

1) A pending interrupt must be serviced.

2) A bytecode whose actions are implemented by Nova code has been interpreted.

In both cases the Nova program counter is set to a fixed value (in page 0) before control is passed to the Nova emulator. (In the second case, the PC value is different for each distinct bytecode.) In addition, the following conditions hold:

a) Any parameters expected by the Nova code appear in consecutive ACs beginning with AC0. Any Nova accumulator whose content is not explicitly supplied by the microcode will have undefined contents. In particular, no parameters are passed for a case 1 exit (pending interrupts). The microcode does not interpret the values transferred to the Nova ACs.

b) The state of the current process has been dumped to a process state vector whose starting address is stored in main memory at location 'CurrentState' (location 23B).

c) Unless a pending interrupt is being serviced (case 1), Nova interrupts have been disabled.

d) Any results generated by the Nova code must be transferred to the saved process state vector *before* the Mesa emulator is restarted. The emulator supplies no explicit mechanism to the Nova code for returning results.

Nova dispatch vector

When the Nova emulator receives control, the PC reflects the intended action to be performed. The Mesa emulator uses a dispatch vector beginning at 'NovaDVloc' (location 25B), and indexes it by the particular action required. The entries in the dispatch vector may be any Nova instructions, but in most cases will be a JMP to a Nova program that implements the desired semantics. The particular indices and relevant parameters passed are given by the following table:

Index	Action	Parameters
0	interrupt	none
1	STOP	none
2-3	unused	
4	ME	AC0
5	MRE	AC0, AC1
6	MXW	AC0, AC1, AC2
7	MXD	AC0
10B	NOTIFY	AC0
11B	BCAST	AC0
12B	REQUEUE	AC0, AC1, AC2

Global Frame Format

The global frame overhead has been reduced to 3 words, which have the following contents:

<G>+0	bits 0-8 contain the GFI, bits 9-14 are used by software, bit 15 is the frame links/code links indicator.
<G>+1	is the code base (odd => swapped out, even => address of code segment).
<G>+2	not used by Mesa emulator microcode.

Xfer Traps

A mechanism to implement trapping of control transfers has been implemented in Mesa 4.0. It is described in detail in a separate document [8].

Interrupts

After any Xfer, regardless of the cause, the Mesa emulator guarantees that at least one "useful" instruction will be executed. "Useful" means an instruction which is not a NOOP. In Mesa 3.0, no padding instructions were necessary and thus every instruction was considered "useful". In Mesa 4.0 this is no longer true, and this guarantee is needed to preserve certain properties within trap handlers (see below).

Trap Handlers and Parameters [7]

Mesa 4.0 trap handlers obtain their arguments by means of the RR and WR instructions. Four internal registers have been assigned for trap parameters:

XTSreg:	holds Xfer trap state (see [8]).
XTPreg:	holds Xfer trap parameters (see [8]).
ATPreg:	holds allocation trap parameter.
OTPreg:	holds parameters for all other traps.

Thus, Xfer traps use XTSreg and XTPreg, allocation traps use ATPreg, and code-swapped-out and unbound procedure traps use OTPreg.

An additional constraint on trap handlers is that they cannot assume that interrupts have been disabled by the microcode (this was true for some trap handlers in Mesa 3.0). However, the microcode continues to guarantee that one instruction following a trap will be executed before any interrupts are taken. Therefore, if an IWDC instruction is the first instruction of a trap handler, no interrupts will occur before the trap handler has had the opportunity to obtain its parameter(s).

As a result of these changes, the α byte interpretations of RR and WR have been redefined. The new meanings are:

α	RR	WR	
1	WDC	WDC	
2	XTSreg	XTSreg	
3	XTPreg	---	
4	ATPreg	---	
5	OTPreg	---	
6	clock	---	(see below)
7	code-base	---	

The clock value returned by RR6 is the low-order 16 bits of the Alto clock, regardless of the model. Thus the format of this value will be different on the Alto I and Alto II.

Stack overflow and underflow are reported by the same trap mechanism used in Mesa 3.0, but may not occur immediately after the instruction that caused the stack error. However, the trap *will* occur before or during the next instruction that either manipulates the stack or terminates a straight-line execution sequence.

AV, SD, and GFT

The addresses of these tables were stored in internal registers by the Mesa 3.0 emulator and could be accessed and modified by RR and WR. In Mesa 4.0, fixed locations have been established for each of these tables, as follows:

AV	1000B
SD	1060B
GFT	1400B

References

[1] Thacker, Chuck. *OIS Processor Principles of Operation.* Version 2.0, April 9, 1977.

[2] Levin, Roy. **Comparison of Old and New Alto/Mesa Microcode.** November 4, 1977. Filed on [Maxc]<Levin>UCodeComparison.bravo.

[3] Levin, Roy. **Extensions to Alto/Mesa Microcode.** November 7, 1977. Filed on [Maxc]<Levin>UCodeExtensions.bravo.

[4] Levin, Roy. **Mesa/Nova Interface.** November 15, 1977. Filed on [Maxc]<Levin>MesaNovaInterface.bravo.

[5] Levin, Roy. **Incompatibilities in Alto/Mesa microcode, version 24.** November 23, 1977. Filed on [Maxc]<Levin>Incompatibilities24.bravo.

[6] Levin, Roy. **Extensions to Alto/Mesa Microcode for version 26.** December 5, 1977. Filed on [Maxc]<Levin>UCodeExtensions26.bravo.

[7] Wick, John. **Mesa 4.0 microcode (loose ends).** March 14, 1978. Filed on [Maxc]<Wick>MICRO40.bravo.

[8] Levin, Roy. **A Mechanism for Monitoring Transfers of Control in (Alto) Mesa.** March 29, 1978. Filed on [Ivy]<Levin>Mesa40>XferTrap.bravo.

Distribution
Mesa Users
Mesa Group

c: Belleville
Charnley
Lampson
Thacker