ALTO: A PERSONAL COMPUTER SYSTEM

Charles P. Thacker

Edward M. McCreight

Xerox Palo Alto Research Center
3180 Porter Drive
Palo Alto, California 94304

December 1974

Contents

1.0 INTRODUCTION

This document is a description of the Alto, a small personal computing system designed at PARC CSL.

By 'personal computer' we mean a non-shared system containing sufficient processing power, storage, and input-output capability to satisfy the computational needs of a single user.

An Alto system includes the following subsections:

a) An 875 line television monitor, oriented with the long tube dimension vertical. This monitor provides a 606 by 808 point display which is refreshed from main memory at 60 fields (30 frames) per second. It has programmable polarity, a low resolution mode which conserves memory space, and a cursor whose position and content are under program control.

b) An undecoded keyboard

c) A mouse (pointing device) and five-finger keyset.

d) A Diablo Model 31 or Model 44 disk file.

e) Optionally, a Diablo Hy Type printer.

f) Up to 64K 16 bit words of 850ns semiconductor memory.

g) A microprogrammed processor which controls the disk and display, and emulates a virtual machine whose characteristics are approximately those of the Data General Nova.

The processor, disk, and their power supplies are packaged in a small cabinet. The other I/O devices may be a few feet away, and are pleasingly packaged for desk top use.

The remaining sections of this document will discuss the hardware and microcode of the standard configuration Alto.

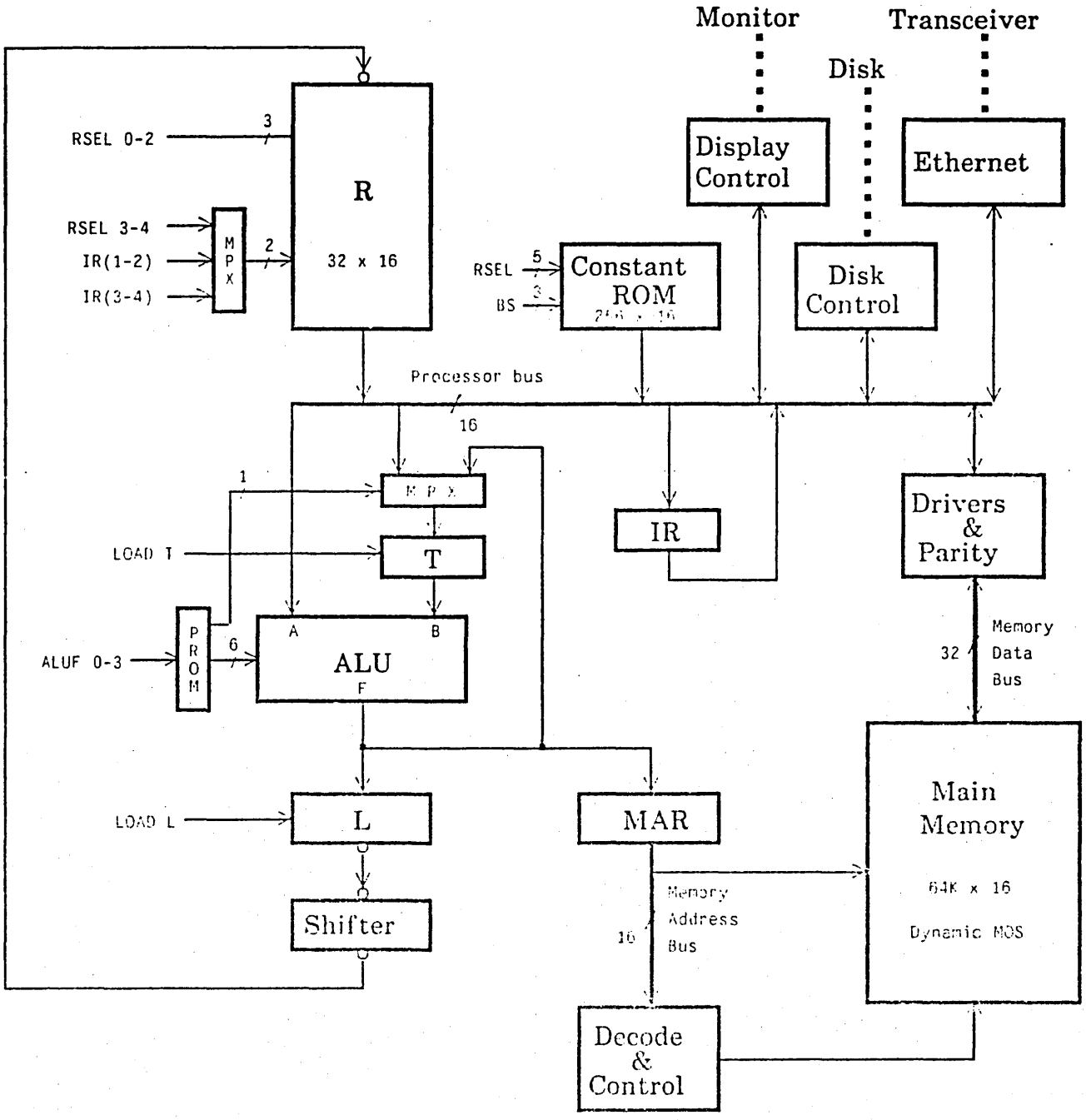


Figure 1
Processor Data Paths

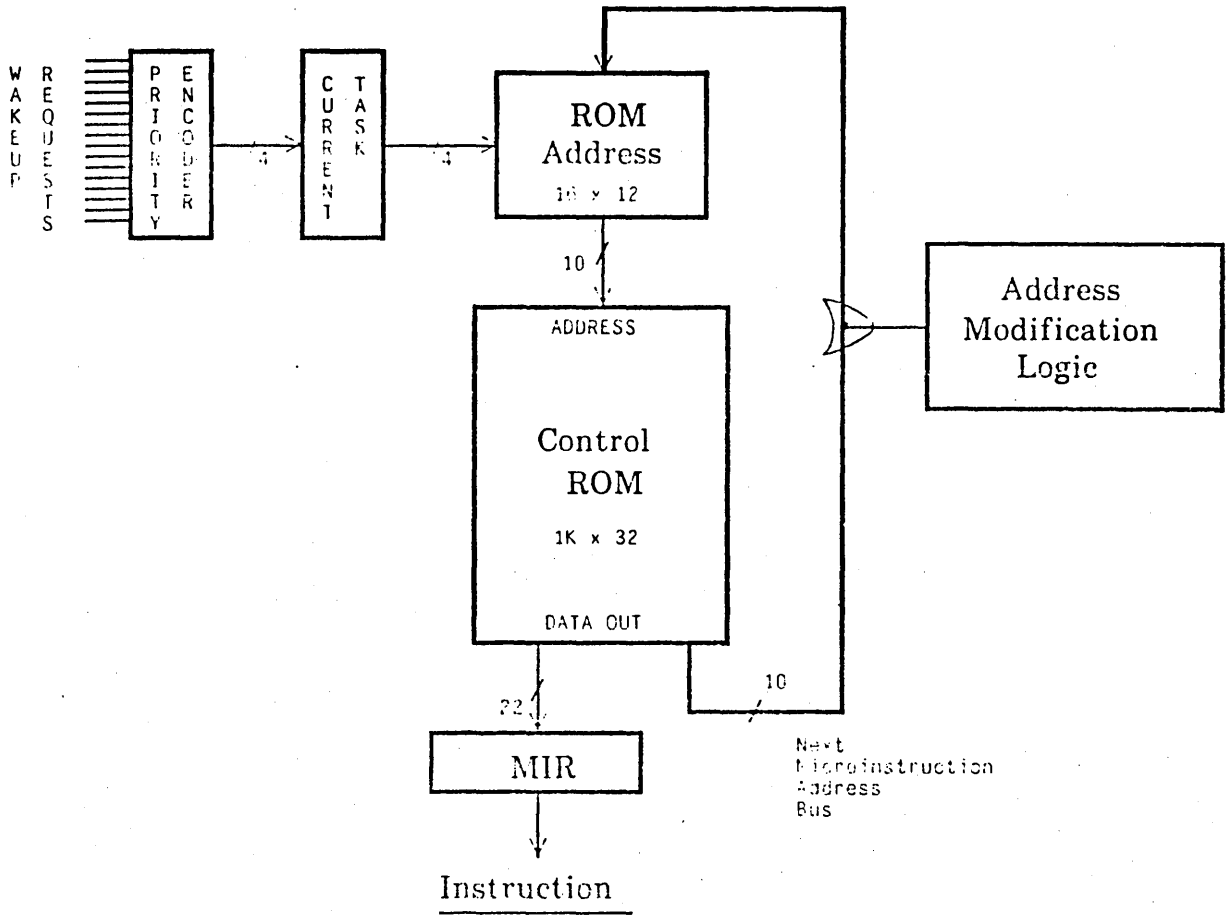


Figure 2
Processor Control

2.0 Microprocessor

The microprocessor is shown schematically in figures 1 and 2. A principal design goal in this system was to achieve the simplest structure adequate for the required tasks. As a result, the central portion of the processor contains very little application-specific logic, and no specialized data paths. The entire system is synchronous, with a clock interval of 170nsec. Microinstructions require one cycle for their execution.

A second design goal was to minimize the amount of hardware in the I/O controllers. This is achieved by doing most of the processing associated with I/O transfers with microprograms. To allow devices to proceed in parallel with each other and with CPU activity, a control structure was devised which allows the microprocessor to be shared among up to 16 fixed priority tasks, Switching between tasks requires very little overhead, and occurs typically every few microseconds.

2.1 Arithmetic Section

The arithmetic section of the processor consists of a 32-word by 16-bit register file R, and four registers, T,L, MAR, and IR. The registers are connected to the memory and to an ALU with a 16-bit parallel bus.

The ALU is a SN74181 type, restricted so that it can do only 16 arithmetic and logical functions. The ALU output feeds the L and MAR registers. T may also be loaded from the ALU output under certain conditions. L is connected to a shifter capable of left and right shifts by one place, and cycles of 8. It has a mode in which it does the peculiar 17-bit shifts of the Nova, and a mode which allows double-length shifts to be done.

The IR register is used exclusively by the Nova emulator to hold the current CPU instruction.

Attached to the bus is a 256-word read only memory (ROM) which holds arbitrary 16-bit constants.

The microprocessor executes instructions from a 1K word by 32-bit programmable read-only memory (PROM). The fields of the microinstruction are:

BIT	MEANING
0-4	R Register Select
5-8	ALU Function
9-11	Bus Data Source
12-15	Function 1
16-19	Function 2
20	Load L
21	Load T
22-31	Next microinstruction address (subject to modifiers)

R Select

The R select field specifies one of the 32 R cells to be loaded or read under control of the bus source field, or, in conjunction with the bus source field, one of the 256 locations to be read from the constant ROM.

The low order two bits of the R address (but not the constant ROM address) may be taken from fields in IR under control of the functions. This allows the emulator to address its central registers easily.

ALU Functions

The ALU function field controls the SN74181 ALU. This device can do a total of 48 arithmetic and logical operations, most of which are relatively useless. The 4-bit field is mapped by a PROM into the 16 most useful functions:

ALU FUNCTION	FIELD FUNCTION	S3,S2,S1,S0,M,C INPUTS TO SN74181
0	BUS	1111 1 0 (A)
1	T	1010 1 0 (B)
2	BUS OR T*	1110 1 0 (A+B)
3	BUS AND T	1011 1 0 (AB)
4	BUS XOR T	0110 1 0 (A XOR B)
5	BUS + 1*	0000 0 0 (A PLUS 1)
6	BUS - 1*	1111 0 1 (A MINUS 1)
7	BUS + T	1001 0 1 (A PLUS B)
10	BUS - T	0110 0 0 (A MINUS B)
11	BUS - T - 1	0110 1 0 (A MINUS B MINUS 1)
12	BUS + T + 1*	1001 0 0 (A PLUS B PLUS 1)
13	BUS+SKIP*	0000 0 SKIP' (A PLUS 1)
14	BUS.T* (AND)	1011 1 0 (AB)
15-17	UNDEFINED	

*If T is loaded during an instruction which specifies this function, it will be loaded from the ALU output rather than from the bus.

Bus Sources

The bus data source field specifies one of 8 data sources for the bus:

VALUE	SOURCE
0	Read R
1	Load R*
2	Nothing (-1)
3	Kstat (disk control status bits)**
4	Kdata (16 bits of disk data)**
5	Memory data
6	Mouse data (4 bits, remainder of word is 1)
7	Disp (low order 8 bits of IR, sign extended)

*This is not logically a source, but since R is gated to the bus during both reading and writing, it is included in the source specifiers. LOAD R forces the BUS to 0, so that T← F(0,T) may be simultaneously executed.

**By convention, these bus sources are task specific, i.e., their meaning depends on the currently active task. KSTAT and KDATA are the interpretations used during the disk sector and word tasks.

Special Functions

The two function fields specify the address modifiers, register load signals (other than those for R, L and T), and other special conditions required in the processor. The first eight conditions specified by each field are interpreted identically by all tasks, but the interpretation of the second eight depends on the active task. The task-independent functions are given below, the task-specific functions are included with the task descriptions.

FUNCTION 1:

VALUE	NAME	MEANING
0	----	No Activity
1	MAR←	Load MAR from ALU output; start main memory reference.
2	Task	Switch tasks if higher priority wakeup pending.
3	Block	Disable current task wakeup until reenabled by hardware generated condition.
4	←L lsh 1	Left shift L (one place)*
5	←L rsh 1	Right shift L (one place)*
6	←L lcy 8	Cycle L (8 places)*
7	←CONSTANT	BUS←constant ROM location addressed by RSELECT.BUS SOURCE.

*modified by DNS (do Nova shifts) function, and MAGIC function

FUNCTION 2:

VALUE	NAME	MEANING
0	----	No Activity
1	BUS=0	NEXT←NEXT OR (BUS=0)
2	SH<0	NEXT←NEXT OR (SHIFTER OUTPUT <0)
3	SH=0	NEXT←NEXT OR (SHIFTER OUTPUT =0)
4	BUS	NEXT←NEXT OR BUS(06)-BUS(15)
5	ALUCY	NEXT←NEXT OR LALUC0*
6	STORE	
7	←CONSTANT	SAME AS F1=7

*The carry used is that produced by the ALU function which last loaded the L register.

2.2 Constant Memory

The constant memory is a 256 x 16 PROM which holds arbitrary constants. The constant memory is gated to the bus by F1=7, F2=7, or BS>4. The constant memory is addressed by the (8 bit) concatenation of RSELECD and BS. The intent in enabling constants with BS>4 is to provide a masking facility, particularly for the MOUSE and DISP bus source. This works because the processor bus ANDs if more than one source is gated to it. Up to 32 such mask constants can be provided for each of the 4 bus sources >4. Note that it is not possible to use a constant other than -1 with the ←MD bus source, since memory parity is calculated on the bus, and a parity error will result if bits are marked off in a word fetched from memory.

2.3 Main Memory

A memory reference is initiated by executing F1=6, MAR←. The program partially controls memory timing, and must observe certain rules to insure correct operation:

a) There may be a minimum of one and a maximum of three microinstructions executed between starting the memory and the data transfer.

b) During the fourth cycle after MAR has been loaded, if F2=6, a store of bus data into the word addressed by MAR will occur.

c) During the fourth cycle of a reference, if BS=5, the reference is a fetch to the word addressed by MAR.

d) During the fifth cycle of a reference, if BS=5, the odd word of the doubleword addressed by MAR is delivered. The memory cycle is extended by one cycle if both words of a doubleword are fetched. If MD is referenced during the fifth cycle, it must have also been referenced during the fourth.

e) If MD is referenced before the fourth cycle of a reference, the processor will be suspended until a total of 4 cycles have elapsed.

f) If RSELECT = 37B during the instruction which starts the memory, a refresh cycle is assumed and all memory cards are activated. This is used by the refresh task.

g) The memory checks parity on all fetches, unless the cycle is a refresh cycle, or the address is > 177000B, in which case an I/O device is being referenced. Parity errors result in activation of a task whose purpose is to deal with the error.

2.4 Microprocessor Control

The control section of the microprocessor is straightforward. Instructions are taken from a 1K word by 32-bit ROM which is addressed by one of the cells of a 16 by 10 RAM. This RAM is addressed by the current task register. The instruction memory produces an instruction and the address of its successor (NEXT (Ø)-NEXT (9)). This successor address may be modified by merging into its bits under control of the function fields of the current microinstruction. This limited branching capability makes coding more difficult than with a more general scheme, but not seriously so, as examples of microcode demonstrate.

While only 10 bits of the PC ram are used in the standard Alto, the hardware contains 12 bits to allow for future expansion. An available option is a P.C. card which contains an additional 1K of control memory implemented with RAM. This memory may be loaded or read by special CPU instructions, and provisions exist for causing any of the 16 tasks to execute instructions from it.

At the end of each cycle, the microinstruction register (MIR) and the PC are loaded, and the cycle repeats. There is only one phase of the system clock. It is true during the last 25 ns. of every instruction.

Tasks

If the processor executes the 'task' function (F1=2) during an instruction, the current task register is loaded (at the end of the instruction) with the number of the current highest priority task as determined by the priority encoder. This causes the next instruction to be fetched from the ROM location specified by the saved task's PC. One additional instruction is executed before the switch becomes effective. A version of the current task register which is delayed from the PC RAM address by one cycle exists so that this instruction can execute task-specific functions, but these functions must do no address modification, since any modification would affect the new task. The situation for two streams of instructions A-F and J-M in two different tasks is shown below:

Instruction Being Executed	Instruction Being Fetched	Address Stored PC at End of Cycle
A	B	C
B	C	D
C *	D	E
D	J	K
J **	K	L
K ***	L	M
L	E	F
E	F	G

*Instruction C allows task switching. New task's PC = J.
**Instruction J does an operation which removes its task's wakeup request.
***Instruction K allows task switching, and the original task is now highest priority.

The task request lines which drive the priority encoder are hardware generated and are not accessible to the microprogram. When a running task executes the TASK function, control will switch to another task only if a

higher priority task has a wakeup request pending, or if the current task no longer has a wakeup request pending. In the latter case, control goes to a lower priority task. The lowest priority task is the CPU emulator, which is always requesting wakeup.

The TASK function should be executed only at times when the current task has no state in L or T, and has no main memory operations in progress, since there is no provision in the hardware for saving this information.

Initialization

The only way in which the microprogram can affect the task structure is to request a task switch. In particular, it cannot affect the PC's of tasks other than itself. This presents an initialization problem which is solved by having each task start at the location which is its task number. Task numbers are written into the PC RAM during a reset cycle, which may be initiated manually or by a CPU instruction

3.0 Emulator

The lowest priority task is a microprogram which implements an instruction set which is similar to that of the Data General Nova, with the following differences:

1) Addresses are 16, rather than 15, bits. The principal implications of this are that multi-level indirection is not possible, and that all 16 bits of an AC used for indexing are significant, so that the sign bit cannot be used as a flag.

2) There are no auto-index locations.

3) I/O class instructions are not implemented. They have been replaced with a number of additional instructions.

4) An entirely different interrupt system has been implemented.

3.1 Augmented Instruction Set

Opcodes above 60000B, which are I/O instructions in the Nova, have been reassigned to instructions which augment the standard instruction set. Bits 3 through 7 of the instruction determine 32 opcodes, each of which may use the displacement field. One of these opcodes is used to represent up to 256 instructions which do not require a displacement or a parameter as part of the opcode.

Currently, only a small number of the available extra instructions have been implemented. When an unimplemented opcode is executed, the microcode stores the PC (which points one location beyond the instruction which caused the trap) in location TRAPPC, and simulates a JMP@ TRAPVEC ! IR(3-7). TRAPPC, and the 32 word trap vector are all reserved locations in page 1 (see Appendix B).

The currently assigned extra instructions and their operations are:

CYCLE (60000): Left cycle (rotate) the contents of AC0 by the amount specified in instruction bits 12-15, unless this value is zero, in which case cycle AC0 left by the amount specified in AC1. Leaves AC1 = cycle count mod 20B.

JSRII: (64400): JSR double indirect, PC relative:
AC3←PC+1
PC←rv(rv(PC+DISP))

JSRIS (65000): JSR double indirect, AC2 relative:
AC3←PC+1
PC←rv(rv(AC2+DISP))

CONVERT: (67000) The convert instruction does scan conversion of characters, i.e., it transfers data between an area of main memory containing a font and an area of memory containing a bit map to be displayed on the TV monitor.

Convert takes a number of arguments:

AC0 contains the address of the destination word into which the upper left corner of the character is to be placed, offset by NWRDS, the number of words to be displayed on each scan line (AC0←DWA-NWRDS).

AC3 points to a character pointer in the font for the character to be displayed (AC3=FONTBASE+CHARACTER CODE).

AC2+Displacement points to a two word table:

word 0: NWRDS (number of words per to scan line)

word 1: DBA, the destination bit address corresponding to the left hand edge of the character. Convert interprets this bit address reversed from the normal convention, i.e., bit 0 is the LSB, bit 15 the MSB.

Convert requires that a 16 word mask table be set up starting at MASKTAB in page 1. MASKTAB!N=(2**N+1)-1 ($0 \leq n < 16$).

The format of a standard Alto font is:

FONTBASE-2 The height of a character in scan lines.

FONTBASE-1 The width of a character (and surrounding spaces) in raster points. If the font is proportionally spaced, this word contains the width of the widest character, and bit 0 is set.

FONTBASE to FONTBASE+377B: Self-relative pointers to word XH of the character descriptor block for the character codes 0-377B.

FONTBASE+400B to FONTBASE+400B+EXTCNT-1: These locations contain self-relative pointers to word XH of the character descriptor blocks for extensions, i.e., portions of characters which are wider than 16 bits.

FONTBASE+400B+EXTCNT to end: Contains a number of character descriptor blocks of the form:

word 0 to word XH-1: The bit map for the character and surrounding spaces. The bit map does not include 0's at the top and bottom of the character, as the character will be vertically positioned by convert. The upper left-hand bit of the character is in the MSB of word 0.

word XH: If the character is less than 16 bits wide, this word contains (2*width)+1. If the character requires an extension, this word contains 2* a pseudo-character which is used as a character code to index the font. If this is the last extension block of a character, this word contains (2* the width of the final extension), rather

than the total width. The pointer indexed by the character code points to this word.

word XH+1: In the left byte, HD. In the right byte, XH. HD is the number of scan lines to skip before displaying the character, XH is the height of the bit map.

The convert instruction ORs the bitmap contained in one descriptor block into the display area. If the character does not require an extension, convert skips, with the following information in the AC's:

AC0: unchanged
AC1: DBA and 17B
AC2: unchanged
AC3: the width of the character in bits

If the character requires an extension, convert returns normally. AC3 contains the pseudo-character code for the extension, and AC0-2 are as above.

DIR (61000) Disable interrupts: See <3.2>

EIR (61001) Enable interrupts: See <3.2>

BRI (61002) Branch return from interrupt: See <3.2>

RCLK (61003) Read Clock:

The microcode maintains a 26 bit real time clock which is incremented by the memory refresh task at 38.08 microsec. intervals. The high order 16 bits of this clock are maintained in location RTC in page 1, the low order 10 bits are kept in the high order bits of R37. R37 is incremented by 100B each 38.08 microsec. The low order 6 bits of R37 contain state information unrelated to the time.

RCLK loads AC0 with the contents of location RTC, and loads AC1 with the contents of R37. If the program then zeros bits 10-15 of AC1, it will have a clock value in units of .595 microseconds. AC0 alone is in units of 39ms. The period of the clock is about 40 minutes.

SIO (61004) Start I/O:

Start I/O is included to facilitate I/O control. It places the contents of AC0 on the processor bus, and executes the STARTF function (F1=17B). If the Ethernet hardware is installed, the serial number of the machine (0-377B) is loaded into AC1.

BLT (61005) Block transfer:

BLKS (61006) Block store:

Block transfer and block store take the following arguments:

AC0: Address of the first source word-1 (BLT), or data to be stored (BLKS).

AC1: Address of the last word of the destination area.

AC3: Negative word count.

Since these instructions (1) are potentially time consuming, and (2) keep their state in the AC's, they are interruptible. If an interrupt occurs, the PC is decremented by one, and the AC's contain the intermediate state. On return, the instruction continues. On completion, the AC's are:

AC0: Address of last source word+1 (BLT), or unchanged (BLKS).

AC1: unchanged.

AC2: unchanged.

AC3: 0.

The first word of the destination area (AC1 + AC3 + 1) is the first to be stored into.

SIT (G1007) Start interval timer:

The microcode implements an interval timer which has a resolution of 38 microsec., and a maximum period of 10 bits. As the principal application for this timer is to do bit sampling for a serial EIA-RS232 compatible communications line, the timer is specialized for this purpose. It uses three dedicated locations in page 1:

ITTIME: Contains the time at which the next timer interrupt should be caused. This is a 10 bit number, left justified in the 16 bit word. The low order 6 bits are not interpreted.

ITIBITS: This word contains one or more bits specifying the channel or channels on which the timer interrupt is to occur.

ITQUAN: When the interval timer interrupt is caused, the microcode stores a quantity in this location which depends on the mode.

The SIT instruction ORs the contents of AC0 into R37. The high 13 bits should be 0, the low order 2 bits determine the interval timer mode:

R37(14,15)	Mode
00	Off
01	Normal mode. Each 38 microsec., compare R37(0-9) with ITTIME(0-9). If they are equal, cause an interrupt on the channel specified by ITIBITS. Store the current state of the EIA interface in ITQUAN, and set R37(14,15) to zero. The state of the EIA interface is bit 15 of location EIALOC in page 177. This bit is 0 if the line is spacing, 1 if it is marking.

10	Same as 00
11	Every 38 microsec., check the state of the EIA line. If the line is marking, do nothing. If the line is spacing, cause an interrupt on the channel specified by ITIBITS. Store the current value of R37 in ITQUAN, and set R37(14,15) to zero.

The intention is that a program which does EIA input can use mode 3 to monitor the line for the arrival of a character, and can then use mode 2 to time the center of each bit. By storing the state of the line, the interrupt latency can be as much as 1 bit time without errors.

61010	RDRM	See <8.4>
61011	WTRM	See <8.4>
61012	JMPRM	See <8.4>
61020	MUL	
61021	DIV	

3.2 Interrupts

The emulator microcode implements an interrupt structure which allows both I/O devices and programs to interrupt the main program. The interrupt system provides 15 channels of vectored interrupts with adjustable priority. The interrupt system uses one register in R (NWW, new wakeups waiting), and a number of fixed locations in page 1:

ACTIVE:	This word contains 1's for the channels which are currently active. The highest priority channel is associated with bit 15, the lowest is associated with bit 1. Bit 0 is not used, and should not be set by any program.
WWLOC:	This word contains bits for channels on which interrupts are pending. Bit 0 is not used.
PCLOC:	During an interrupt, the PC is saved here.
INTVEC to INTVEC+14	Contains pointers to the service routines for the 15 interrupt channels. The first word corresponds to the highest priority interrupt channel (bit 15), the last corresponds to the lowest priority channel (bit 1).

The main loop of the emulator checks NWW during the fetch of each emulated instruction. If NWW is greater than zero, the microcode computes (NWW OR WW) AND ACTIVE. If this quantity is nonzero, an interrupt is caused. If not, NWW OR WW is stored in WW, NWW is cleared, and the instruction is restarted.

If the interrupt is caused, the microcode stores the program counter in PCLOC, sets bit 0 of NWW to disable further interrupts, clears the bit in NWW corresponding to the interrupt channel about to occur, and loads the PC with rv(INTVEC+CHANNEL).

Interrupts are caused by ORing into NWW or into WW. I/O device microcode usually has a dedicated location in which the program places a bitword for the interrupt(s) to be caused upon completion of I/O activity.

Only one interrupt channel is permanently assigned: the highest priority channel (bit 0) is triggered when a main memory parity error is detected.

The interrupt system uses three instructions:

DIR (61000) Disable interrupts: Sets bit 0 of NWW. Since NWW is negative, the check made at the start of every instruction will not process any new wakeup requests.

EIR (61001) Enable interrupts: Clears bit 0 of NWW, and ORs WW into NWW to detect any interrupts which were requested (by ORing into WW) while interrupts were off.

BRI (61002) Branch and return from interrupt: This instruction clears bit 0 of NWW, ORs WW into NWW, and restores PC from PCLOC.

3.3 Hardware

There is a small amount of special hardware which is used exclusively by the emulator. This hardware is controlled by the task specific F2's, and by the ←DISP bus source.

The IR register is used to hold the current instruction. It is loaded with IR← (F2=14). IR← also merges bus bits 0,5,6 and 7 into NEXT, which does a first level instruction dispatch. The high order bits of IR cannot be directly read, but the displacement field of IR (8 low order bits (sign extended)), may be read with the ←DISP bus source.

There are two additional F2's which assist in instruction decoding. The IDISP function (F2=15) does a 16 way dispatch under control of a 256x4 PROM. The inputs of the PROM are bits 1-7 of IR. ←ASOURCE (F2=13) has two roles. It does an instruction dispatch based on IR, and it replaces the two low order bits of the R select field with IR(1,2), allowing the emulator to address its accumulators (which are assigned to R0-R3). The dispatch done depends on IR(00); if it is false, IR bits 1-7 drive the PROM mentioned above. If IR(00) is true, the shift field of IR (bits 8 and 9) is gated to NEXT. The 8th bit of the PROM input is F2(02) (to differentiate between IDISP and ←ASOURCE).

F2=13, ACDEST, causes bits 3 and 4 of IR to be used as the low order two bits of the RSELECT field. This addresses the accumulators from the destination field of the instruction. The selected register may be loaded or read.

The emulator has two additional bits of state, the SKIP and CARRY flip flops. CARRY is identical to the Nova carry bit, and is set or cleared as appropriate when the DNS← (do Nova shifts) function is executed. DNS also addresses R from IR(3-4), and sets the SKIP flip flop if appropriate. The PC is incremented by 1 at the beginning of the next emulated instruction if SKIP is set, using ALUF 13. IR← clears SKIP.

Note that the functions which replace the low bits of RSELECT with IR affect only the selection of R; they do not affect the address supplied to the constant ROM.

The two additional emulator specific functions, BUSODD and MAGIC, are not peculiar to Nova emulation, but are included for their general usefulness. BUSODD merges BUS(15) into NEXT(09), and MAGIC is applied in conjunction

with LSH and RSH to allow double length shifts. It shifts the high order bit of T into the low order bit of R on left shifts, and shifts the low order bit of T into the high order bit of R on right shifts.

The STARTF function (F1=17) is not associated with any special hardware. It is used by the SIO instruction, and is to be used to define commands for (as yet unspecified) I/O hardware.

4.0 Display Controller

4.1 Programming Characteristics:

The display controller handles transfers between the main memory and the CRT. The CRT is a standard 875 line raster-scanned TV monitor, refreshed at 60 fields per second from a bit map in main memory. The CRT contains 606 points horizontally, and 808 points vertically, or 489,648 points total. Thirty-eight 16 bit words are required to represent each scan line; 30704 words are required to fill the screen.

The display is defined by one or more display control blocks in main memory. Control blocks are linked together starting at location DASTART in page 1, and have the following format:

DASTART: Pointer to first DCB word Ø, or Ø if display is off. All DCBs must start on even word boundaries.

DASTART+1: Vertical field interupt bit mask.

DCB word Ø: Pointer to next DCB, or 0 if this is the last.

DCB word 1:
- Bit Ø: Ø=high resolution mode, 1=low resolution mode
- Bit 1: Ø=black on white background presentation, 1=white on black background
- Bits 2-7:HTAB: On each scan line of this block, wait HTAB* 16 bits before displaying information from memory.
- Bits 8-16:NWRDS: Each scan line in this block is defined by NWRDS 16 bit words. (NWRDS must be even).

DCB word 2: SA: Bit map starting address (This address must be even)

DCB word 3: SLC: This block defines 2*SLC scan lines, SLC in each field.

At the start of each field, the display controller inspects DASTART and DASTART+1. An interrupt is initiated on the channel specified by the bit(s) in DASTART+1. The controller then executes each DCB sequentially until the display list or the field ends. At normal resolution, the first scan line of the first (even) field of a block is taken from location SA to SA+NWRDS-1, the first scan line of the odd field is taken from locations SA+NWRDS to SA+2*NWRDS-1. During each field, the bit map address is incremented by NWRDS between each scan line. Thus, although the display is interlaced, its representation in memory is not. In low resolution mode, the video is generated at half speed, and each scan line is displayed twice (once in each field). During each field, the bit map address is not incremented between the display of adjacent scan lines. This makes the format of the bit map in memory identical for both modes--only the size of the presentation is affected by the mode.

4.2 Hardware

The display controller consists of a sync generator, a data buffer and serializing shift register, and three microcode tasks which control data handling and communicate with the Alto program. The hardware is shown in block form in Figure 3. The 16 word buffer is loaded from the Alto bus with the DDR← function (F2=10, specific to the display word task DWT). The purpose of the intermediate buffer is to synchronize data transfers between the main buffer, which is synchronous with the 170ns. master clock, and the shift register, which is clocked with an asynchronous bit clock. The sync generator provides this clock and the vertical horizontal synchronization signals required by the monitor.

The bit clock is disabled by vertical and horizontal blanking, and its rate can be set by the microcode to either 50 or 100 ns. by the function SETMODE (F2=11, specific to the display horizontal task DHT). This function examines the two high order bits of the processor bus. If bit 0=1, the bit clock rate is set to 100ns period (at the start of the next scan line), and a 1 is merged into NEXT(9). SETMODE also latches bit 1 of the processor bus and uses the value to control the polarity of the video output. A third function, EVENFIELD (F2=10, specific to DHT and to the display vertical task DVT), merges a 1 into NEXT (9) if the display is in the even field.

The display control hardware also generates wakeup requests to the microprocessor tasking hardware. The vertical task DVT is awakened once per field, at the beginning of vertical retrace. The display horizontal task is awakened once at the beginning of each field, and thereafter whenever the display word task blocks. DHT can block itself, in which case neither it nor the word task can be awakened until the start of the next field. The wakeup request for the display word task (DWT) is controlled by the state of the 16 word buffer. If DWT has not executed a BLOCK, if DHT is not blocked, and if the buffer is not full, DWT wakeups are generated. The hardware sets the buffer empty and clears the DWT block flip-flop at the beginning of horizontal retrace for every scan line.

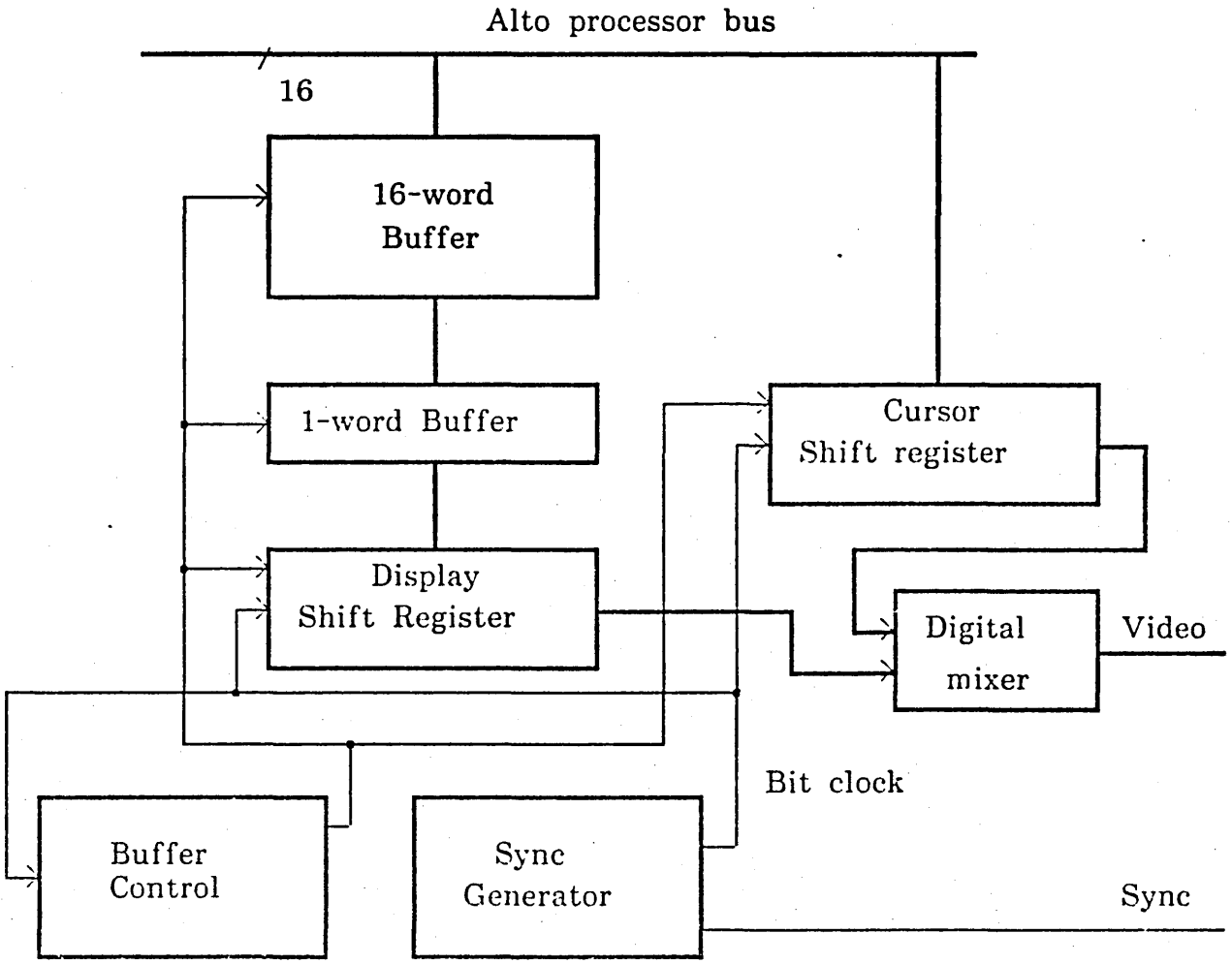


Figure 3
Display Control

4.3 Display Controller Microcode

The display controller microcode is divided into three tasks. The highest priority task is DVT, the display vertical task, the next is DHT, the horizontal task, and the third is DWT. The display controller uses 6 registers in R:

CBA	holds the address of the currently active DCB+1
AECL	holds the address of the end of the currently active scan line's bit map in main memory
SLC	holds the number of scan lines remaining in the currently active DCB
HTAB	holds the number of tab words remaining on the current scan line
DWA	holds the address of the bit map doubleword currently being fetched for transmission to the hardware buffer.
MTEMP	is a temporary cell.

The vertical task initializes the controller by setting SLC to 0 and CBA to DASTART+1. It also merges the contents of DASTART+1 into NWW, which will cause an interrupt if the specified channel is active. DVT also sets up information required for the cursor (see below), TASKs and becomes inactive until the next field.

DHT starts by initiating a fetch to the word addressed by CBA. It checks SLC, and if it is zero, the controller is finished with the current DCB, and the link word of the DCB is fetched. If this word is non-zero, it replaces CBA and processing of a new block is begun. If the link word is zero, DHT blocks until the start of the next field.

If the check of SLC indicates that more scan lines remain in the current DCB, SLC is decremented by one and the fetch of (CBA) is used to obtain the second word of the DCB, rather than the link word. The contents of this word are used to set the display mode and polarity, and the tab count is extracted and put into HTAB. NWRDS is extracted, and used to increment DWA and AECL by the appropriate amount, depending on the mode and field. All the registers required by DWT have now been set up, and DHT TASKs and becomes inactive until DWT blocks.

If a new DCB is required, DHT fetches all four words of the new DCB, and initializes all the registers. During all scan lines of a DCB except the first, DHT only accesses the first doubleword of the block.

DWT has the sole task of transferring words from memory to the hardware. When it first awakens during horizontal retrace, it checks HTAB. If it is non-zero, it enters a loop which outputs HTAB 0's to the display. When HTAB is zero, a second loop is entered which fetches a doubleword from the location specified by DWA. DWA is compared with AECL, and if they are equal, DWT blocks until the next scan line. DWA is incremented by 2, in preparation for the fetch of the next doubleword. If DWA≠AECL, DWT continues to supply words to the buffer whenever it becomes non-full.

4.4 Cursor

Because of the difficulty of inserting a cursor at the appropriate place in the display bit map at reasonable speed, a hardware cursor is included in the Alto. The cursor consists of an arbitrary 16x16 bit patch, which is merged with the video at the appropriate time. The bit map for the cursor is contained in 16 words starting at location CURMAP in page 1, and the x,y coordinates of the cursor are specified by location CURLOC and CURLOC+1 in

page 1. The coordinate origin for the cursor is the upper left hand corner of the screen. The cursor presentation is unaffected by changes in display resolution. Its polarity is that of the current DCB, or the last DCB processed if it is located on an area of the screen not defined by a DCB, The cursor may be removed from view in a number of ways. The most efficient in terms of processing time is to set the x coordinate to -1.

The cursor hardware consists of a 16 bit shift register which holds the information to be displayed on the current scan line, and a counter which is incremented by the bit clock, and determines the x coordinate at which the shift register begins shifting.

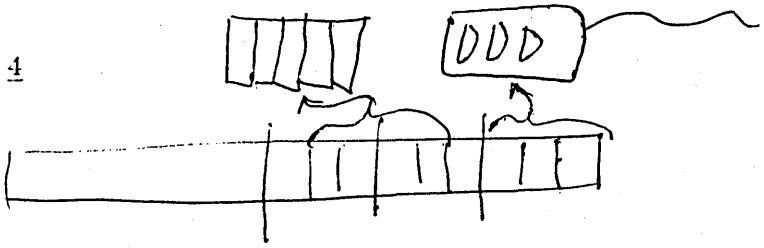
The hardware is loaded during horizontal retrace by the cursor task microcode, which simply copies the x coordinate and bit map segment from the R memory into the hardware.

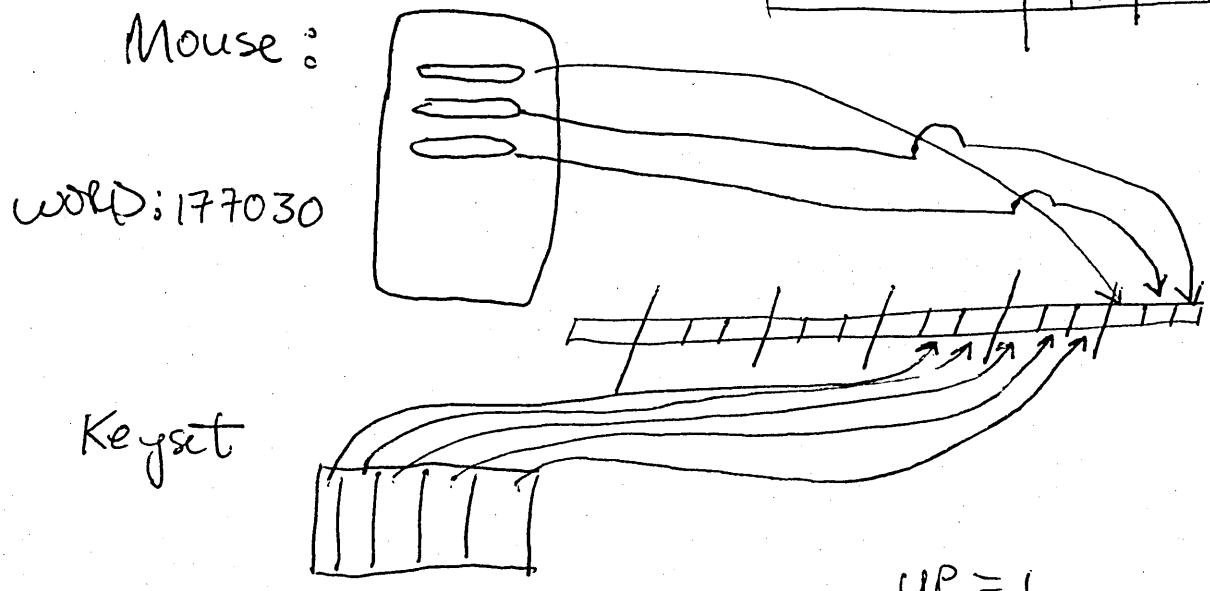
The values of x and the bit map are set up in R by a section of the memory refresh task, whose wakeup and priority are arranged so that it runs during every scan line after DWT has done all necessary output and DHT has set up the information required by DWT for the next scan line. MRT checks the current y position of the display, and if it is in the range in which the cursor should be displayed, fetches the appropriate bit map segment from CURMAP. When the cursor y position is exceeded by the display, a flag is set in MRT to disable further processing. The x and y coordinates of the cursor are fetched from CURLOC and CURLOC+1 at the beginning of each display field by a section of the display vertical task microcode.

Cursor processing is distributed as it is to minimize the amount of processing which must be done during the monitor's horizontal retrace time. This time is approximately 6 microsec, and it must include the worst case latency imposed by tasks at lower priority than the display, plus the worst case disk word processing time (the disk word task is at higher priority than the display), plus the time necessary for DWT to partially fill the display buffer, plus cursor processing time.

Bit	Word KBDAD	Word KBDAD+2	Word KBDAD+1	Word KBDAD+3
0	5	3	1	R
1	4	2	ESC	T
2	6	W	TAB	G
3	E	Q	F	Y
4	7	S	CTRL	H
5	D	A	C	8
6	U	9	J	N
7	V	I	B	M
8	0(zero)	X	%	LOCK
9	K	O	SHIFT(left)	SPACE
10	-	L	.	[
11	P	;	+	
12	/	'	RETURN	SHIFT(right)
13	\	]	←	SPARE3
14	LF	SPARE2	DEL	xxx
15	BS	SPARE1	xxx	xxx

Figure 4





5.0 Miscellaneous Peripherals

The Alto can have a number of slow peripherals which appear to programs as memory locations in the range 177000-177777B. The standard peripherals are described here and in Appendix C, which describes each word in detail.

5.1 Keyboard

The Alto keyboard contains 61 keys. It appears to the program as four 16 bit words in 4 adjacent locations starting at KBDAD. Depressed keys correspond to Ø's in memory, idle keys correspond to 1's. Figure 4 is a layout of the keyboard, showing the keytops and the word number, bit number corresponding to each key.

5.2 Mouse

The mouse is a hand-held pointing device which contains two encoders which digitize its position as it is rolled over a table-top. It also has three buttons which may be read as three bits of a memory location, in the manner of the keyboard.

The mouse coordinates are maintained by the MRT microcode in locations MOUSELOC (x) and MOUSELOC+1 (y) in page 1. These coordinates are relative, i.e., the hardware only increments and decrements them. The resolution of the mouse is approximately 100 points per inch.

5.3 Keyset

The standard Alto includes a five-finger keyset which is presented to the program as 5 bits of a memory location, similar to the keyboard.

5.4 Diablo Printer

The Alto includes an interface to a Diablo HyType printer. The printer uses a portion of one memory location to report status, and another location into which the Alto program can store to send signals to the printer. None of the timing signals required by the printer are generated automatically--all must be program generated. For detailed information on the printer, refer to the Diablo manual.

6.0 Disk and Controller

The disk controller is designed to accommodate one of a variety of DIABLO disk drives, including models 31 and 44. Each drive accommodates one or two disks. Each disk has two heads, one per side. Information is recorded on each disk in a 12-sector format on each of up to 408 (depending on the disk model) radial track positions. Thus, each disk contains up to 9792 recording positions (2 heads x 12 sectors x 408 track positions). Figure 6 tabulates various useful information about the performance of the disk drives.

The disk controller records three independent data blocks in each recording position. The first is two words long, and is intended to include the address of the recording position. This block is called the Header block. The second block is eight words long, and is called the Label block. The third block is 256 words long, and is the Data block. Each block may be independently read, written, or checked, except that writing, once begun, must continue until the end of the data block.

When a block is checked, information on the disk is compared word for word with a specified block of main memory. During checking, a main memory word containing 0 has special significance. When this word is encountered, the matching word read from the disk is stored in its place and does not take part in the check. This feature permits a combination of reading and checking to occur in the same block. (It also has the drawback of making it impossible to use the disk controller to check for words containing 0 on the disk.)

The Alto program communicates with the disk controller via a four-word block of main memory beginning at location KBLK. The first word in interpreted as a pointer to a chain of disk command blocks. If it contains 0, the disk controller will remain idle. Otherwise, the disk controller will commence execution of the command contained in the first disk command block. When a command is completed successfully, the disk controller stores in KBLK a pointer to the next command in the chain and the cycle repeats. If a command terminates in error, a 0 is immediately stored in KBLK and the disk controller idles. At the beginning of each sector, status information, including the number of the current sector, is stored in KBLK+1. This can be used by the Alto program to sense the readiness of the disk and to schedule disk transfers, for example. When the disk controller begins executing a command, it stores the disk address of that command in KBLK+2. This information is later used by the disk controller to decide whether seek operations or disk switches are necessary. It can be used by the Alto program for scheduling disk arm motion. If the Alto program stores an illegal disk address (like -1) in this word, the disk controller will perform a seek at the beginning of the next disk operation. (This is useful, for example, when the operating system wants to force a restore operation.) The disk controller also communicates with the Alto program by interrupts (see Section 3.2). At the beginning of each sector interrupts are initiated on the channels specified by the bits in KBLK+3.

KBLK:	Pointer to first disk command block
KBLK+1:	Status at beginning of current sector
KBLK+2:	Disk address of most-recently started disk command
KBLK+3:	Sector interrupt bit mask

Device	Diablo 31	Diablo 44	
Number of drives/Alto	1 or 2	1	
Number of packs	1 removable	1 removable 1 fixed	
Number of cylinders	203	406	
Tracks/cylinder/pack	2	same	
Sectors per track	12	same	
Words per sector	2 header 8 label 256 data	same	
Data words/track	3072	3072	
Sectors/pack	4872	9744	
Rotation time	40	25	ms
Seek time (approx.)	15+8.6*sqrt(dt)	8+3*sqrt(dt)	ms
min-avg-max	15-70-135	8-30-68	ms
Average access to 1 megabyte	80	32 (using both packs)	ms
Transfer rate:			
peak/avg	1.6/1.22	2.5/1.9	MHz
peak/avg	10.2/13	6.7/8.	ns/word
per sector	3.3	2.1	ms
for full display	.460	.266	sec
for big memory	1.03	.6	sec
whole drive	19.3	44 (both packs)	sec

Figure 6

A disk command block is a ten-word block of memory which describes a disk transfer operation to the disk controller, and which is also used by the controller to record the status of that operation. The first word is a pointer to the next disk command block in this chain. A 0 means that this is the last disk command block in the chain. When the command is complete, the disk controller stores its status in the second word. The third word contains the command itself, telling the disk controller what to do. The fourth word contains a pointer to the block of memory from/to which the header block will be transferred. The fifth word contains a similar pointer for the label block. The sixth word contains a similar pointer for the data block.

The seventh and eighth words of the disk command block control the initiation of interrupts when the command block is finished. If the command terminates without error, interrupts are initiated on the channels specified by the bits in DCB+6. However, if the command terminates with an error, the bits in DCB+7 are used instead.

The ninth word is unused by the disk controller, and may be used by the ALTO program to facilitate chained disk operations. The tenth word contains the disk address at which the current operation is to take place.

DCB:	Pointer to next command block
DCB+1:	Status
DCB+2:	Command
DCB+3:	Header block ptr
DCB+4:	Label block ptr
DCB+5:	Data ptr
DCB+6:	Command complete no-error interrupt bit mask
DCB+7:	Command complete error interrupt bit mask
DCB+10:	Currently unused
DCB+11:	Disk address

A disk address word A contains the following fields:

Field	Range	Significance
A[0-3]	0-11D	Sector number
A[4-12]	0-405D for Model 44 0-202D for Model 31	Track number
A[13]	0-1	Head number
A[14]	0-1 for Model 44 0 for Model 31	Disk number (see also C[15]) (0 is removable pack on Model 44)
A[15]	0-1	0 normally, 1 if track 0 is to be addressed via a hardware "restore" operation.

A disk command word C contains the following fields:

Field	Range	Significance
C[0-7]	110	Checked to verify that this is a valid disk command
C[8-9]	0-3	0 if Header block read 1 if Header block to be checked 2 or 3 if Header block to be written
C[10-11]	0-3	0 if Label block to be read 1 if Label block to be checked 2 or 3 if Label block to be written
C[12-13]	0-3	0 if Data block to be read 1 if Data block to be checked 2 or 3 if Data block to be written
C[14]	0-1	0 normally 1 if the command is to terminate immediately after the correct track position is reached (before any data is transferred)
C[15]	0-1	XOR'ed with A[14] to yield hardware disk number

A disk status word S has the following fields:

Field	Values	Significance
S[0-3]	0-11D	Current sector number
S[4-7]	17	(One can tell whether status has been stored by setting this field initially to 0 and then checking for non-zero).
S[8]	0-1	1 means seek failed, possibly due to illegal track address.
S[9]	0-1	1 means seek in progress
S[10]	0-1	1 means disk unit not ready
S[11]	0-1	1 means data or sector processing was late during the last sector. Data and current sector number unreliable.
S[12]	0-1	1 means disk interface was not transferring data last sector
S[13]	0-1	1 means checksum error. Command allowed to proceed.
S[14-15]	0-3	0 means command completed correctly 1 means hardware error (see S[8-11]) or sector overflow. 2 means check error. Command terminated instantly. 3 means disk command specified illegal sector.

Several clever programming tricks have been suggested to drive the disk controller. For an initial program load, KBLK should be set to point to a disk command block representing a read into location STRT. Before setting KBLK, the Alto program should put a JMP STRT instruction in STRT; afterward it should jump to STRT. The disk controller transfers data downward, from high to low addresses, so that when location STRT is changed the reading of the block is complete.

Another trick is to chain disk reads through their label blocks. That is, the label block for sector n contains part of the disk command block for reading sector n+1, and so on.

Disk Controller Implementation

The disk controller consists of a modest amount of hardware and two microcode tasks (the sector task and the word task). Communication with the CPU program is via four special main memory words (KBLK to KBLK+3) and disk command blocks, as described earlier.

The sector task is awakened by a sector signal from the disk. When awakened, it stores the status of the disk and controller in the special disk status word (KBLK+1). In addition, if this sector signal terminates a disk command (for example, a data transfer during the previous sector), the status of the disk and controller are stored in the status word of the disk command block containing the terminated command, and the command block pointer (KBLK) is advanced. If a command was terminated with an error, KBLK (DCB pointer) is set to 0 and KBLK+2 (current disk address) is set to -1. The effect of this is to cause the disk controller to abandon the current disk command chain and to forget where the disk arm is positioned.

Next, the sector task considers the first command on the disk command block chain (by using KBLK). If there is none, or if the disk unit is not ready to accept a command, the sector task goes to sleep until the next sector pulse. Otherwise, the sector specified in the new command is verified to be less than 13 decimal. Then, the disk and cylinder specified in the new command are compared with those stored in KBLK+2 (current disk address), and then the new disk address is stored in KBLK+2 and in the disk controller hardware. Part of the new command is also stored in the hardware. If the comparison is unequal, a seek is initiated and the sector task goes to sleep until the next sector pulse.

If the comparison was equal, the SEEKOK hardware flag is tested. If that is OK, then the no-transfer bit of the disk command (bit 14 of the command word of the current disk command block) is tested to see whether a data transfer is required. If not, the sector task goes to sleep such that the command will terminate at the next sector pulse. If a data transfer is required, the specified sector number and the current disk sector number are compared. If unequal, the sector task goes to sleep until the next sector pulse. If sector numbers are equal, awakening of the word task is enabled and the sector task goes to sleep such that the command will terminate at the next sector pulse.

The word task awakens when a word has been processed by the disk controller hardware and the word task has been enabled by the sector task. First, a starting delay is computed, based on whether the current record is to be read or written. Second, control is dispatched based on the current record number. A record length and main memory starting address are computed based on the record number. In addition, special starting delays are

computed for record number 0. The disk unit is set into the delay mode appropriate for the operation (read/write) and the word task goes to sleep the appropriate number of times.

Then a sync word is written (if writing) or awaited (if reading). Finally the main transfer loop is entered. Here the word count is decremented, a memory operation is started, and control is dispatched on the transfer type. If read, the disk word is stored in memory. If write, the memory word is sent to the disk. If check, the memory word is compared with 0. If non-zero, the disk and memory words are compared. An unequal compare here terminates this sector's operation with an error immediately. If the memory word is 0, it is replaced by the disk word. In any case, the checksum is updated and control returns to the main transfer loop. Due to the ALU functions available, the main transfer loop moves in sequence from high to low main memory addresses.

After the wordcount reaches 0, the checksum is written or checked. A checksum error will be noted in the status word, but will not terminate this sector's operation. A finishing delay is computed, based on the current operation, the disk unit is set into a delay mode appropriate to the operation, and the delay happens. Finally, all disk transfers are shut off, the record number is incremented, and control returns to the beginning of the word task.

To accomplish all this, the disk controller hardware communicates with the microprocessor in four ways: first, by task wakeup signals for the sector and word tasks; second, by five task-specific function 2's which modify the next microinstruction address; third, by seven task-specific functions 1's, four of which activitate bus destination registers, and the remaining three of which provide useful pulses; and fourth, by two bus sources. The following tables describe the effects of these.

F1 Value	Name	Effect
17	KDATA←	The KDATA← register is loaded from BUS(0-15). This register is the data output register to the disk, and is also used to hold the disk address during KADR← and seek commands. When used as a disk address it has the format of word A on page 29.
16	KADR←	This causes the KADR← register to be loaded from BUS(8-14). This register has the format of word C on page 30. In addition, it causes the head address bit to be loaded from KDATA←(13).
15	KCOM←	This causes the KCOM← register to be loaded from BUS(1-5). The KCOM← register has the following interpretation: (1) XFEROFF =1 inhibits data transmission to/from the disk. (2) WDINHIB =1 prevents the disk word task from awakening. (3) BCLKSRC =1 means take bit clock from disk input or crystal clock,

		as appropriate. =0 forces use of crystal clock.
	(4) WFFO	=0 holds the disk bit counter at -1 until a 1-bit is read. =1 allows the bit counter to proceed normally.
	(5) SENDADR	=1 causes KDATA←(4-12) and KDATA←(15) to be transmitted to disk unit as seek address. =0 inhibits such transmission.
14	CLRSTAT	Causes all error latches in disk controller hardware to reset.
13	INCRECNO	Advances the shift registers holding the KADR← register so that they present the number and read/write/check status of the next record to the hardware.
12	KSTAT←	KSTAT(12-15) are loaded from BUS(12-15). (Actually, BUS(13) is 'OR'ed into KSTAT(13).) This enables the microcode to enter conditions it detects into the status register.
11	STROBE	Initiates a disk seek operation. The KDATA← register must have been loaded previously, and the SENDADR bit of the KCOMM← register previously set to 1.

F2 Value	Name	Effect
10	INIT	NEXT←NEXT OR (IF WDTASKACT AND WDINIT) THEN 37 ELSE 0)
11	RWC	NEXT←NEXT OR (IF current record to be written THEN 3 ELSE IF current record to be checked THEN 2 ELSE 0)
12	RECNO	NEXT←NEXT OR MAP (current record number) where MAP(0) ← 0 MAP(1) ← 2 MAP(2) ← 3 MAP(3) ← 1
13	XFRDAT	NEXT←NEXT OR (IF current command wants data transfer THEN 1 ELSE 0)
14	SWRNRDY	NEXT←NEXT OR (IF disk not ready to accept command THEN 1 ELSE 0)
15	NFER	NEXT←NEXT OR (IF fatal error in latches then 0 ELSE 1).
16	STROBON	NEXT←NEXT OR (IF seek strobe still on then 1 else)).

7.0 Ethernet

The Ethernet is the principal means of communications between an Alto and the outside world. It is a broadcast, multi-drop, packet-switching, bit serial communications network. The goals of the design were to connect up to 256 nodes (Altos, Novas, etc), separated by as much as 1 mile, with a moderate bandwidth (2.94 megabits/sec) channel.

Alto Ethernets come in three pieces: The transceiver, the interface, and the microcode. The transceiver is a small device which taps into the passing ether inserting and extracting bits under the control of the interface while disturbing the ether as little as possible. The same device is used to connect all types of ethernet interfaces to the ether, so the transceiver design is not specific to the Alto, and will not be described here.

7.1 Programming Characteristics:

Programs communicate with the microcode and the interface via 10 reserved locations in page 1 and the emulator instruction SIO. Word counts, buffer addresses, etc. are put in the appropriate locations and then SIO is executed with an Ethernet command in AC0. The main memory locations and their functions are:

EPLOC: Post location. Receives status information when a command completes.
EBLOC: Interrupt bit location. Contents are ORed into NWW when a command completes causing an interrupt if non zero.
EELOC: EOT count location. Receives the number of words remaining in the buffer at command completion.
ELLOC: Load location. Used by the microcode to hold a mask of 1's shifted in from the right for generating random countdowns during retransmissions.
EICLOC: Input count location. Size of the input buffer.
EIPLOC: Input pointer location. Pointer to the beginning of the input buffer.
EOCLOC: Output count location. Size of the output buffer.
EOPLOC: Output pointer location. Pointer to the beginning of the output buffer.
ESLOC: Serial number location. Holds the serial number that the microcode should use for deciding which messages to receive.
ESPARE: Spare location. Reserved for future use.

SIO does two things involving the Ethernet: (1) It passes commands encoded in bits 14 and 15 of AC0 to the interface and (2) returns the serial number of the Alto in the right byte of AC0 and all ones in the left byte. The serial number is set by wirewraps on the backplane. The commands corresponding to the 4 combinations of bits 14 and 15 are:

AC0[14-15]		
	00	Do nothing
	01	Start transmitter
	10	Start receiver
	11	Reset interface and microcode

Upon completion of a command, EPLOC contains the status of the microcode in the left byte and the status of the hardware in the right byte. The possible values of the microcode status byte (left) and their meanings are:

EPLOC[0-7]	0	Hardware condition terminated input main loop. Whether this is good or bad depends on the hardware status.
	1	Hardware condition terminated output main loop. See hardware status byte.
	2	Buffer space exhausted during input operation.
	3	Load overflow. 16 collisions while trying to transmit.
	4	Command specified a zero length buffer
	5	Hardware condition aborted current activity. Generally indicates a reset command.
	6	Microcode branch conditions that should never happen cause this code to be posted if they do happen. Call a repairman.

Bits in the hardware status byte (right) are low true. If zero, their meanings are:

EPLOC[8-9]	Unused. These should always be one.
EPLOC[10]	Input data late. Interface did not get enough processor cycles.
EPLOC[11]	Collision.
EPLOC[12]	Input CRC bad
EPLOC[13]	Input command issued.
EPLOC[14]	Output command issued.
EPLOC[15]	Incomplete transmission - the received packet did not end on a word boundry.

Command completion can be detected in two ways: (1) zero EPLOC and wait for it to go non-zero, and (2) set bits in EBLOC corresponding to the channels on which interrupts are desired at command completion. A program can determine the size of an input message (and though not too useful, the size of an output message) by subtracting the contents of EELOC from the original buffer count in ExCLOC. The microcode never modifies the buffer count or pointer locations. To keep the receiver listening as often as possible, if EICLOC is non-zero when an output command is issued, the microcode will start the receiver 'under' the transmitter: while the transmitter is counting down a random retransmission interval after a collision, the receiver is listening. If a message arrives addressed to the receiver, the transmission attempt is aborted and the incoming message is received. The transmit command is not executed in this case, and must be reissued. The microcode status byte in EPLOC tells what happened.

The first word of all Ethernet packets must contain the address to which the packet is destined in the left byte, and the address of the sender (or 'source') in the right byte. Receivers examine at least the destination byte, and in some cases the source byte to determine whether to copy the message into memory as it passes by. Address zero has special meaning to the Ethernet. Packets with destination zero are broadcast packets, and all active receivers will copy them into memory. If a program wishes to receive all packets on the net regardless of address, it should put zero instead of the machine serial number into ESLOC.

7.2 The Ethernet Interface

The Ethernet interface consists of a data buffer, an output shift register and phase encoder, a clock recovery circuit and input shift register, a CRC register, and one microcode task. The hardware is shown in block diagram form in figure 7. Packets on the ether are phase encoded and transmitter

synchronous: it is the responsibility of the receiver to decide where a packet begins (and thus establish the phase of the data clock), separate the clock from the data, and deserialize the incoming bit stream. The purpose of the write register is to synchronize data transfers between the input shift register whose clock is derived from the incoming data, and the main buffer which is synchronous to the processor system clock. The large main buffer is necessary because the Ethernet task is relatively low priority, and the worse case latency from request to task wakeup is on the order of 20 microseconds. The phase encoder uses the system clock where one bit time is two clock periods, so the output shift register is synchronous with the main buffer, however latency is still a problem and the large buffer is needed for the transmitter also.

Included in the clock recovery section is a one-shot which is retriggered by each level transition of a passing packet. This detects the envelope of a packet and is called its 'carrier'. Ethernet phase encoders mark the beginning of a packet by appending a single 1, called the sync bit, to the beginning of all transmissions. The leading edge of the sync bit of a packet will trigger the carrier one-shot of a listening receiver and unambiguously establish the receiver clock phase. The sync bit is clocked into the input shift register and recirculated every 16 bit times thereafter to mark the presence of a complete word in the register. If carrier drops without the sync bit at the end of the register, the transmission was incomplete, and is flagged in the status bits. When the shift register is full, the word is transferred to the main buffer write register where it sits until the buffer control has synchronized its presence and and there is room to write it into the buffer. If the shift register fills up again before the word has been transferred from the write register to the main buffer, then the main buffer has overflowed and the input data late flip flop is set. Ethernet transmitters accumulate a 16 bit cyclic redundancy checksum on the data as it is serialized, and append it after the last data word. As a receiver deserializes an incoming packet it recomputes the checksum over the data plus the appended CRC word. If the resulting receiver checksum is non-zero, the received packet is in error, and the condition is flagged in the hardware status byte. Since the CRC is of no interest to the emulator program, a wakeup request to empty data from the buffer is only made when it contains two or more words. This reduces the effective size of the buffer by one word, but insures that the CRC will be left behind at the end of a packet.

The phase encoder starts up when the microcode has decremented the countdown to zero, there is no carrier present, and either the main buffer is full, or if the message is less than 16 words long, all of it has been transferred to the buffer. The phase encoder will not start up while there is carrier present. This means that collisions can only happen because of delay in sensing carrier between widely spaced transmitters. Collisions are detected at the transceiver by comparing the data the interface is supplying to the data being received off the ether. If the two are not identical, a signal is returned to the interface which immediately stops the phase encoder and sets the collision flip flop causing a wakeup request to the microcode. Countdowns are accomplished by setting a flip flop from the microcode which will cause a wakeup request on the next occurrance of SWAKMRT'. This makes the grain size of countdowns about 37 microseconds.

By changing the timing components on the one-shots in the clock recovery section and dividing the system clock to the phase encoder by two, the interface can be run at half speed (1.47 MBits/sec.), reducing the number of processor cycles required.

The interface and the transceiver are connected together by three twisted pairs for signals plus two supply voltages and ground supplied from the interface. The signals are (1) transmitted data to the transceiver, (2) received data from the transceiver, and (3) the signal from the transceiver indicating interference.

7.3 Ethernet Microcode

The Ethernet microcode uses a single task and 2 registers in R:

ECNTR: holds the number of buffer words to be processed
EPNTR: points at the word prior to that next to be processed

The task and R registers are shared by input and output so that at any time they are (1) unused, (2) transmitting a packet, or (3) receiving a packet. When an Ethernet SIO is issued while the Ethernet microcode is reset, the code dispatches on whether it's to do input or output.

Each Ethernet SIO has a result which is posted. The state of the microcode and hardare at the time of the post are deposited in EPLOC. The contents of ECNTR are deposited in EELOC. And the contents of EBLOC are ORed into NWW. The hardware responds to an EPFCT (F1=14) by placing its status on the bus; EPFCT clears the hardware.

An input command causes the microcode to load the input buffer size and pointer into ECNTR and EPNTR, respectively, and to start up the input hardware with EISFCT (F2=16). Then the task blocks waiting for the first word of a passing packet. The input hardware waits for quiet on the ether, turns on looking for a packet's first word, and then wakes up the input task. Upon waking, the input task asks whether the hardware is delivering the first word of a packet using EBFCT (F2=14); EBFCT puts a 1 on NEXT(7) if something other than a data word is causing the wakeup. If a data wakeup, the task checks the packet's first word addresses against the filtering specification in ESLOC. If ESLOC is zero, the packet is accepted and the input main loop is used to transfer the packet into the waiting input buffer. If the packet's first byte is zero, it is a broadcast packet and is accepted. Otherwise, the first byte of the packet must match that stored in ESLOC if the packet is to be accepted. If the packet is not accepted, the hardware is reset and started up again looking for input. If a non-data wakeup, the task posts.

The input main loop repeatedly counts down the buffer size in ECNTR and advances the buffer pointer in EPNTR depositing packet words until either the hardware terminates the flow or the buffer overflows; in either case, the input terminates and posts. Packet words arriving at the input buffer wake up the input main loop once every 5.44 microseconds on the average. With each wakeup, the input main loop checks for data with EBFCT. Data is taken from the hardware with EIDFCT (a task-dependent bus source, KDATA). Each word requires 6 cycles.

An output command causes the microcode to see if a 1 has been shifted into bit 0 of ELLOC and if so posts a load overflow. If not, its contents are used to mask off the clock in R37 to generate a random number; ELLOC is then shifted left once bringing a 1 in from the right making the mask (i.e., the load) larger. The random number is then counted down in microcode using the memory refresh task clock, SWAKMRT', to wake it up every 37 microseconds. The EEFCT (F1=15) executed in the task instruction of the countdown loop causes the hardware to request a wakeup at the next tic of SWAKMRT'. During countdown, if EICLOC is non-zero, the hardware is

enabled for input; if an input word arrives during countdown, ECBFCT (F2=15) branches the microcode to input to process it as if an input command had been issued. When the countdown reaches zero, the output buffer size and address are loaded into ECNTR and EPNTR, respectively. The output hardware is started with EOSFCT (F2=11) and the main output loop is entered to transmit the packet. Actual transmission of the packet does not begin until the hardware's buffer has been filled by the output main loop or an EEFCT (below) has signalled the end of the packet. With a full buffer, the hardware then waits for silence on the cable Ether before beginning to transmit; we call this Deference.

The output main loop repeatedly counts down the packet length in ECNTR and advances the address in EPNTR taking words from the output buffer and giving them to the output hardware until either the buffer is emptied or a hardware condition aborts it. Once again, EBFCT is used to branch on a non-data wakeup: EODFCT (F2=10) is used to transfer data from the bus to the output hardware. The output main loop is awakened for a packet word once every 5.44 microseconds on the average. When the output buffer is emptied, the microcode issues an EOT with EEFCT (F2=13) and waits for the hardware to terminate; it then causes an output post.

On output, EBFCT generates a 4-way branch by ORing bits onto NEXT(6) and NEXT(7). If neither are ORed with a 1, the wakeup is a data request. If NEXT(6) is turned on, output should be aborted with a post. If only NEXT(7) is ORed on, a collision is indicated; the hardware has detected interference in transmission of the current packet. If a collision is indicated during output, the microcode immediately aborts transmission (Deference again) with EPFCT and branches back as if a new output command had been issued: but now the load is larger than the last time around. Our current algorithm requires that ELLOC be zeroed prior to issuing an output command so that it takes 16 increasingly infrequent collisions to cause a load overflow.

7.4 Software Initiated Boot Feature

Since the Ethernet interface already decodes the emulator instruction SIO, it was easy to attach meaning to another bit in AC0. If Bit 0 of AC0 is one when an SIO is executed, the result is identical to pressing the boot button on the Alto keyboard.

8.0 Control RAM

The control RAM is an optional logic card containing a fast (90 nsec.) 1024-word by 32-bit read/write memory, an even faster (40 nsec.) 32-word by 16-bit read/write memory, and logic to interface those memories to the Alto's microinstruction bus, processor bus, and ALU output. Unlike other memories in the Alto, the larger memory of the control RAM can hold microinstructions and/or data, and may be used exactly as the memory of a von Neumann computer.

8.1 RAM-Related Tasks

The control RAM performs data manipulation (as distinct from microcode fetching) functions in response to certain values of the Function 1 and Bus Source fields of the microinstruction. Not all tasks will likely be interested in these functions. More important, not all tasks will have the appropriate values of the Function 1 and Bus Source fields uncommitted. A RAM-related task is defined as one during whose execution the control RAM card will respond to the Function 1 and Bus Source fields of microinstructions. The standard Alto is wired so that the emulator task is the only RAM-related task. At most two other tasks can be made RAM-related by a simple backpanel wiring change.

8.2 Processor Bus and ALU Interface

Unfortunately, since the Alto's ALU output and processor bus are each 16-bits wide and its microinstruction bus is 32-bits wide, loading the control RAM from the ALU output and reading the control RAM onto the processor bus is slightly clumsy. It is done by using the RAM-related F1's WRTRAM and RDRAM (see Appendix A).

For both reading and writing, the control RAM address is specified by the control RAM address register, which is loaded from the ALU output whenever T is loaded from its source. This load may take place as late as the microinstruction in which WRTRAM or RDRAM is asserted. The bits of the ALU output have the following significance as a control RAM address:

BIT	USE	
0-3	Ignored	
4	RAM/ROM	0 means read/write the control RAM 1 means read the control ROM
5	HALFSEL -	Ignored on writing 0 means read out the low-order 16-bits of the addressed word 1 means read out the high-order 16-bits of the addressed word
6-15	Word address (0-1023)	

Since it was expected that reading the control RAM would be a relatively infrequent operation, a single assertion of RDRAM reads out only one half of a 32-bit control RAM word onto the processor bus. To read out both halves, the control RAM address register must be loaded twice and RDRAM invoked twice. Data resulting from RDRAM is AND'ed onto the processor bus during the microinstruction following that in which the RDRAM was asserted.

In contrast, it was expected that writing into the control RAM would occur frequently. Therefore a single application of WRTRAM writes both halves of

a control RAM word at once. The M register contents (see section 8.6) after the microinstruction containing the WRTRAM will be written into the high-order half of the addressed control RAM word. The ALU output during the microinstruction following the WRTRAM will be written into the low-order half. This protocol mates well with doubleword main memory reads.

Both RDRAM and WRTRAM cause the microprocessor's system clock to stop for one cycle. This may yield unspecified results if the system clock is also stopped for some other reason (e. g. waiting for memory data). As a general rule, the system clock should run without hesitation during the microinstruction following a RDRAM or WRTRAM, except for the effect of the RDRAM or WRTRAM itself.

8.3 Microinstruction Bus Interface

The PC0 bit of the program counter of each Alto task specifies whether that task is currently executing microinstructions from the control ROM or the control RAM. The next microinstruction address field of a microinstruction is not wide enough to specify a transfer from ROM to RAM or vice-versa. A special transfer mechanism exists only for RAM-related tasks, in the form of SWMODE, a RAM-related F1. SWMODE inverts the PC0 bit of the emulator task, taking effect after the microinstruction following that in which the SWMODE appears. In other words, in RAM-related tasks SWMODE behaves much like an address modifier. Other tasks cannot switch between ROM and RAM.

The correspondence of ALU output bits with microinstruction fields appears in the following table:

High/Low Order Halfword	Bit of ALU Output	Meaning	Value in Example
H	0-4	R Register Select	0
H	5-8	ALU Function Select	0
H	9-11	Bus Data Source	5
H	12-15 *	Function 1	2
L	0-3 *	Function 2	0
L	4	Load T	0
L	5 *	Load L	1
L	6-15	Next micro address	325

Fields denoted by * are represented with their high-order bit inverted; this is an artifact of hardware microinstruction decoding. As an example, consider the representation of the microinstruction

```
L←MD, TASK, :LOCA;
```

where LOCA is 325. The values for the various microinstruction fields are listed in the table above. After complementing the appropriate high-order bits and concatenating, we see that the microinstruction above would be represented as 132 in its high-order halfword and 12325 in its low-order halfword.

8.4 Reset Mode Register

The RAM-related F1 RMR← causes the reset mode register to be loaded from the processor bus. This register is used to supply the initial value of the PC0 bit of each task's program counter during the next reset ("boot") operation. The 16 bits of the processor bus correspond to the 16 Alto tasks in the following way: the low order bit of the processor bus specifies the

initial mode of the lowest priority task (emulator), and the high-order bit of the bus specifies the initial mode of the highest priority task. A task will commence in the control ROM if its associated bit in the reset mode register contains the value 1; otherwise it will start in the control RAM. Upon initial power-up of the Alto, and after each reset operation, the reset mode register is automatically set to all 1's, corresponding to starting all tasks in the control ROM.

8.5 Standard Emulator Access

In addition to the instructions listed in section 3.1, the standard emulator includes three extra instructions allowing basic access to the control RAM. More sophisticated access may be implemented by using the basic access primitives to write microcode into the control RAM and then transferring control to that microcode.

RDRAM (61011) Read from Control RAM:

Reads the control RAM halfword addressed by AC1 into AC0. The microcode is:

```
RDRM:  T←AC1, RDRAM;
       L←ALLONES;    (AND'ed with control RAM data)
       AC0←L, :START;
```

WRTRAM (61012) Write into Control RAM:

Writes AC0 into the high-order half and AC3 into the low-order half of the control RAM word addressed by AC1. The microcode is:

```
WTRM:     T←AC1;
          L←AC0, WRTRAM; (This loads the M register)
          L←AC3;
          :START;
```

JMPRAM (61010) Jump to Control RAM:

Sends control of the emulator task to the RAM location in AC1 (mod 1024). This operation is fraught with peril. If done in error it is the only emulator instruction which can cause the machine to plunge off the deep end. If the RAM is not installed, control will go to the ROM location in AC1. Clever coders can use this feature to determine from within whether or not a control RAM is installed. However they are better advised to make this determination using WRTRAM and RDRAM. The microcode for JMPRAM is:

```
JMPR:     T←AC1, BUS, SWMODE;
          :NOVEM;   (NOVEM = 0)
```

8.6 M and S Registers

The control RAM card also includes an M register and 31 S registers. The M register is the analog of the basic Alto's L register. It provides data for the S registers, which are analogous to the basic Alto's R registers. These additional registers were provided to ease the tight constraint on R register availability which might have limited the utility of the control RAM.

The similarities between the M and L registers, and between the R and S

registers are striking. Both M and L are loaded from the output of the ALU, and only when the Load L bit of the microinstruction is active. R registers are loaded from L, and S registers are loaded from M. Both R and S registers output data onto the processor bus. Both R and S registers are addressed by the RSELECT field of the microinstruction. (Thus the same caveats which apply to the use of R37 apply to S37 (see section 2.3 f).) Loading and reading of both R and S registers are controlled by the Bus Source field of the microinstruction.

Nevertheless there are considerable differences. To begin with, the M and S registers are active only when a RAM-related task is executing. This means, for example, that in the highest-priority RAM-related task it is not necessary to save the value of M across a TASK, since no higher-priority task can change the value of M. Unlike the data path from the L register to the R registers, the data path from the M register to the S registers contains no shifter. When an S register is being loaded from M, the processor bus is not set to zero. The emulator-specific functions ACSOURCE and ACDEST have no effect on S register addressing. And finally, when reading data from the S registers onto the processor bus, the RSELECT value 0 causes the current value of the M register to appear on the bus. (This explains why there are only 31 useful S registers.)

APPENDIX A - MICROINSTRUCTION SUMMARY

FIELDS:		
	0-4	R SELECT
	5-8	ALUF
	9-11	BUS SOURCE
	12-15	F1
	16-19	F2
	20	LOAD L
	21	LOAD T
	22-31	NEXT

ALUF:								
	0:	BUS	4:	BUS XOR T	10:	BUS-T	14:	BUS.T*
	1:	T	5:	BUS+1*	11:	BUS-T-1	15:	UNDEFINED
	2:	BUS OR T*	6:	BUS-1*	12:	BUS+T+1*	16:	UNDEFINED
	3:	BUS AND T	7:	BUS+T	13:	BUS+SKIP	17:	UNDEFINED

*LOADS T FROM ALU OUTPUT

BUS SOURCE:				
	0:	←RLOCATION	4:	(task-specific)
	1:	RLOCATION←	5:	←MD
	2:	Undefined	6:	←MOUSE
	3:	(task-specific)	7:	←DISP

F1(STANDARD):				
	0:	--	4:	←L LSH 1
	1:	MAR←	5:	←L RSH 1
	2:	TASK	6:	←L LCYS
	3:	BLOCK	7:	←CONSTANT

F2(STANDARD):				
	0:	--	4:	BUS
	1:	BUS=0	5:	ALUCY
	2:	SH < 0	6:	MD←
	3:	SH = 0	7:	←CONSTANT

APPENDIX A (Continued)

BUS SOURCE(TASK SPECIFIC):

	0 CPU	7 ETHER	4,16 KSEC,KWD	RAM Related
3:	←SLOCATION	-	←KSTAT	←SLOCATION
4:	SLOCATION←	EIDFCT	←KDATA	SLOCATION←

F1(TASK SPECIFIC):

	0 CPU	4,16 KSEC,KWD	7 ETHER	10 MRT	11 DWT	12 CURT	13 DHT	14 DVT	15 PART	RAM Related
10:	SWMODE	-	-	-	-	-	-	-	-	SWMODE
11:	WRTRAM	STROBE	-	-	-	-	-	-	-	WRTRAM
12:	RDRAM	KSTAT←	-	-	-	-	-	-	-	RDRAM
13:	RMR←	INCRECNO	ELFCT	-	-	-	-	-	-	RMR←
14:	-	CLRSTAT	EPFCT	-	-	-	-	-	-	-
15:	-	KCOMM←	EWFCT	-	-	-	-	-	-	-
16:	-	KADR←	-	-	-	-	-	-	-	-
17:	START	KDATA←	-	-	-	-	-	-	-	-

F2(TASK SPECIFIC):

	0 CPU	4,16 KSEC,KWD	7 ETHER	10 MRT	11 DWT	12 CURT	13 DHT	14 DVT	15 PART
10:	BUSODD	INIT	EODFCT	-	DDR←	XPREG←	EVENFIELD	EVENFIELD	-
11:	MAGIC	RWC	EOSFCT	-	-	CSR←	SETMODE	-	-
12:	DNS←	RECNO	ERBFCT	-	-	-	-	-	-
13:	ACDEST	XFRDAT	EEFCT	-	-	-	-	-	-
14:	IR←	SWRNRDY	EBFCT	-	-	-	-	-	-
15:	IDISP	NFER	ECBFCT	-	-	-	-	-	-
16:	ACSOURCE	STROBON	EISFCT	-	-	-	-	-	-
17:	-	-	-	-	-	-	-	-	-

APPENDIX B - RESERVED MEMORY LOCATIONS

Location	Name	Contents
420	DASTART	Display list header
421	-	Display vertical field interrupt bitword
422	ITQUAN	Interval timer stored quantity
423	ITIBITS	Interval timer bitword
424	MOUSELOC	Mouse X coordinate
425	-	Mouse Y coordinate
426	CURLOC	Cursor X coordinate
427	-	Cursor Y coordinate
430	RTC	Real Time Clock
431-450	CURMAP	Cursor bitmap
452	WW	Interrupt wakeups waiting
453	ACTIVE	Active interrupt bitword
460-477	MASKTAB	Mask table for convert
500	PCLOC	Saved interrupt PC
501-517	INTVEC	Interrupt Transfer Vector
521	KBLK	Disk command block address
522	-	Disk status at start of current sector
523	-	Disk address of most recently started disk command
524	-	Sector interrupt bit mask
525	ITTIME	Interval timer time
527	TRAPPC	Trap saved PC
530-567	TRAPVEC	Trap vector
600	EPLOC	Ethernet post location
601	EBLOC	Ethernet interrupt bit mask
602	EELOC	Ethernet EOT count
603	ELLOC	Ethernet load location
604	EICLOC	Ethernet input buffer count
605	EIPLOC	Ethernet input buffer pointer
606	EOCLOC	Ethernet output buffer count
607	EOPLOC	Ethernet output buffer pointer
610	ESLOC	Ethernet serial number
611	-	Reserved for Ethernet expansion
612	ESPARE	Reserved for Ethernet expansion
613	-	Reserved for Ethernet expansion
614	DCBR	posted by parity task when a main memory parity error is detected.
615	KNMAR	" " " " " " "
616	DWA	" " " " " " "
617	CEA	" " " " " " "
620	PC	" " " " " " "
621	SAD	" " " " " " "

```
/*
/&
```