BLUEFISH Functional Specification

14 July 1978

Art Lim

Table of Contents

Section 1 PDP 11/68 Overview

1.1 Introduction

1.2 11/68 Internal I/O Page Address Register Index

Section 2 PDP 11/68 Processor

2.1 Base Processor Instructions

Table 2.1 Addressing Modes

Table 2.2 Single Operand Instructions

Table 2.3 Double Operand Instructions

Table 2.4 Processor Control Instructions

Table 2.5 Miscellaneous Instructions

Table 2.6 Condition Code Operators

Table 2.7 Programming Difference List

2.2 Processor Control Registers

2.3 Aborts, Traps, Interrupts

2.4 Memory Management

2.5 Cache/Memory Operations

Section 3 Floating Point Processor/Instructions

Section 4 Memory System
To be specified

Section 5 Console

Sections 6 Software Issues

Appendix A Midrange Systems Console Functional Specification

PROJECT BLUEFISH (PDP 11/68)

1.0 Introduction

BLUEFISH represents the new high end of the PDP-11 architecture offering performance in excess of the 11/74 (target at 1.5 times 11/74 base processor) at a significantly reduced cost (½ transfer cost of 11/74). Multiprocessor hooks will be incorporated within the hardware to permit the multiprocessor multi-port memory configuration of the 11/74.

Full 11/74 functionality will be provided with the exception of the following:

System I/D Register	17 777 764
System Size Register	17 777 760 -- 17 777 762
Trap functions within Memory Management	

The PDP 11/68 system configuration is illustrated in figure #1. Communication between functional units of the system is provided through a high bandwidth synchronous backplane interconnect called the PPBI.

Mass bus peripherals will communicate with primary memory through RH68 controllers compatible with RH70 controllers of the 11/70.

The Unibus mapping functions, Unibus arbitration as well as translation of signals from Unibus to PPBI is provided by the Unibus Controller (UBC).

BLUEFISH will provide the Commercial Instruction (DEC STD 168 Revision B) and both integral and accelerated versions of floating point containing the full FP11 instruction set. The cache buffer provides for 2048 bytes of data storage configured in set size 2 and block size 2.

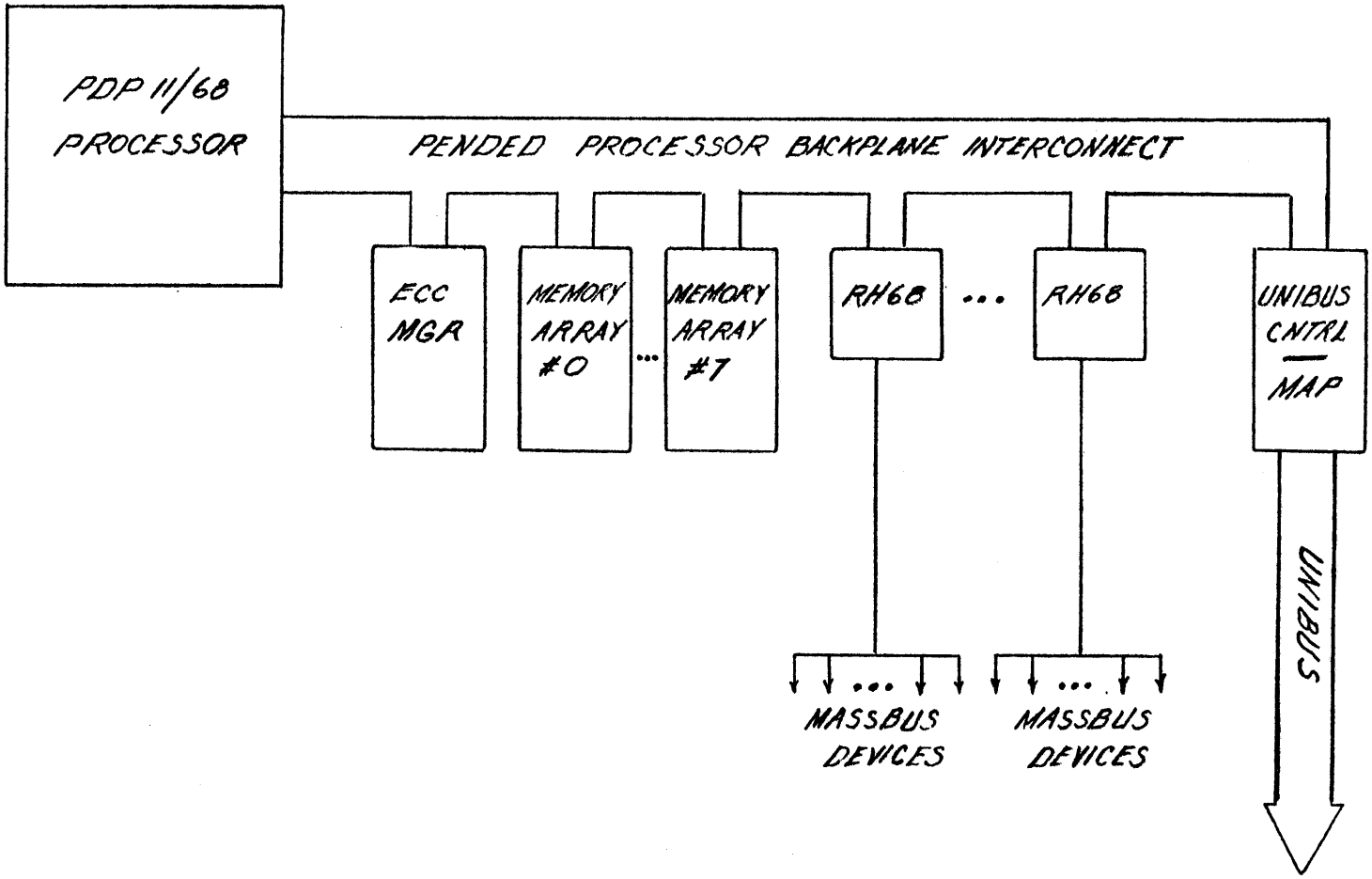
This document has been assembled as a preliminary software reference guide for the Bluefish processor. It's purpose is to summarize the software compatibilities, point out any instructions and in general to eliminate any confusion that may arise in the upgrade of DEC operating systems and diagnostics.

Please review the contents of this document primarily focussing on the format of control and status registers that reflect processor specific functions, trap vectors, and priorities as well as the difference list which identifies the differences in the implementation of functions by different processes in the PDP-11 family. (Table 2-7).

1.2 11/68 Internal I/O Page Address Register Index

Address	Register	Section Described
17 777 776	Processor Status Word	Program Control
17 777 774	Stack Limit	Program Control
17 777 772	Program Interrupt Request	Program Control
17 777 77Ø	Micro break	
17 777 766	CPU Error	Program Control
17 777 764	System I/D	Not Implemented
17 777 762	Upper System Size	Not Implemented
17 777 76Ø	Lower System Size	Not Implemented
17 777 754	Cache/Memory Maintenance Register Ø	Cache/Memory
17 777 752	Cache/Memory Maintenance Register 1	Cache/Memory
17 777 75Ø	Cache/Memory Maintenance Register 2	Cache/Memory
17 777 746	Cache Control	Cache/Memory
17 777 744	Cache/Memory System Error	Cache/Memory
17 777 742	High Error Address	Cache/Memory
17 777 74Ø	Low Error Address	Cache/Memory
17 777 696 17 777 6ØØ	User I/D PAR/PDR	Memory Management
17 777 576	Memory Management Register 2	Memory Management
17 777 574	Memory Management Register 1	Memory Management
17 777 572	Memory Management Register 0	Memory Management
17 777 57Ø	Console and Display	Console
17 772 576	Memory Management Register 3	Memory Management

Address	Register	Section Described
17 772 376 17 772 3ØØ	Kernal I/D PAR/PDR	Memory Management
17 772 276 17 772 2ØØ	Supervisor I/D PAR/PDR	Memory Management



PDP11/68 SYSTEM CONFIGURATION

Section 2 PDP 11/68 Processor

2.1 Base Processor Instructions

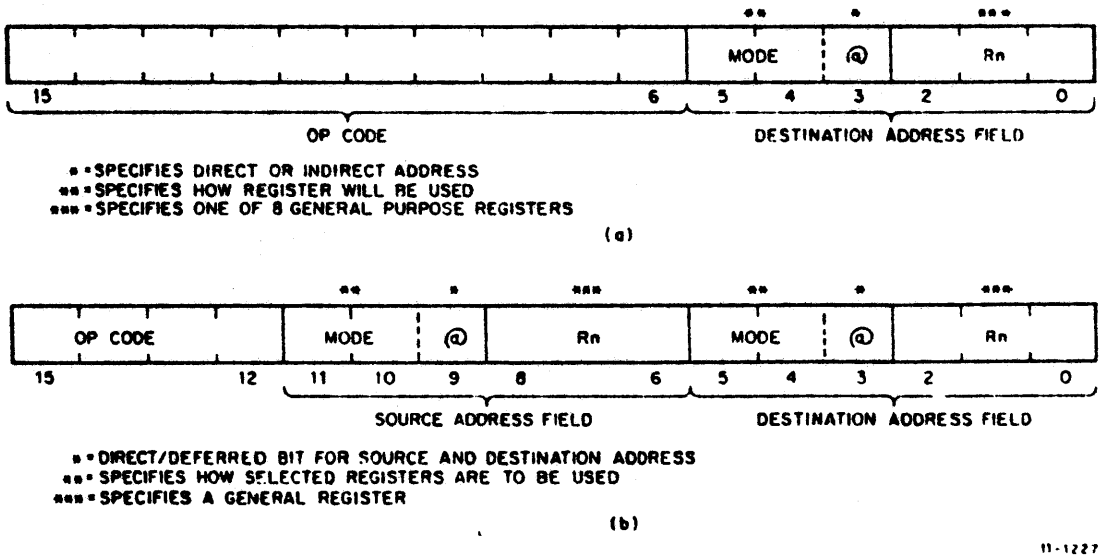


Figure 2-1 Addressing Mode Instruction Formats

Addressing Modes

Mode	Binary Code	Name	Assembler Syntax*	Function
			Direct Modes	
0	000	Register	Rn	Register contains operand. Operations performed on registers in byte mode refer to the low order byte (bits <7:0>) of the register.
2	010	Autoincrement	(Rn)+	Register contains address of operand. Register contents incremented after reference. Registers 6 (SP) and 7 (PC) are always incremented by 2 after reference. RØ - R5 are incremented by 2 for word and by 1 for byte instructions.
4	100	Autodecrement	-(Rn)	Register contents decremented before reference. Register contains address of operand. Registers 6 (SP) and 7 (PC) are always decremented by 2 before reference. RØ - R5 are decremented by 2 for word and by 1 for byte instructions.
6	110	Index	X(Rn)	Value X (stored in a word following the instruction) is added to (Rn) to produce address of operand. Neither X nor (Rn) is modified.

Addressing Modes (con't)

Deferred Modes

1	001	Register Deferred	@Rn or (Rn)	Register contains the address of the operand.
3	011	Autoincrement Deferred	@(Rn)+	Register is first used as a pointer to a word containing the address of the operand, then incremented (always by two, even for byte instructions).
5	101	Autodecrement Deferred	@-(Rn)	Register is decremented (always by two, even for byte instructions) and then used as a pointer to a word containing the address of the operand.
7	111	Index Deferred	@X(Rn)	Value X (stored in the memory word following the instruction) and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) is modified.

Table 2-1 Addressing Modes (cont)

Mode	Binary Code	Name	Assembler Syntax*	Function
			PC Addressing	
2	010	Immediate	#n	Operand follows instruction.
3	011	Absolute	@#A	Absolute address follows instruction.
6	110	Relative	A	Address of A, relative to the instruction, follows the instruction.
7	111	Relative Deferred	@A	Address of location containing address of A, relative to the instruction, follows the instruction.

* Rn = Register
X, n, A = next program counter (PC) word (constant)

PDP 11/68 INSTRUCTIONS

PDP 11/68 instructions can be divided into five groups:

1. Single-Operand Instructions (shifts, multiple precision instructions, rotations)

2. Double-Operand Instructions (arithmetic and logical instructions)

3. Program Control Instructions (branches, subroutines, traps)

4. Operate Group Instructions (processor control operations)

5. Condition Code Operators (processor status word bit instructions)

Tables 2-2 through 2-6 list each instruction, including byte instructions for the respective instruction groups. Figure 2-2 shows the six different instruction formats of the instruction set, and the individual instructions in each format.

Table 2-2 Single Operand Instructions

Mnemonic	OP Code	Operation	Condition Codes	Description
CLR CLRB Clear	0050DD* 1050DD	(dst) ← 0	N: cleared Z: set V: cleared C: cleared	Contents of specified destination are replaced with zeroes.
COM COMB Complement	0051DD 1051DD	(dst) ← ~(dst)	N: set if most significant bit of result is 0 Z: set if result is 0 V: cleared C: set	Replaces the contents of the destination address by its 1's logical complement (each bit equal to 0 set and each bit equal to 1 cleared).
INC INCB Increment	0052DD 1052DD	(dst) ← (dst) + 1	N: set if result is less than 0 Z: set if result is 0 V: set if (dst) was 077777 C: not affected	Add 1 to the contents of the destination.
DEC DECB Decrement	0053DD 1053DD	(dst) ← (dst) - 1	N: set if result is less than 0 Z: set if result is 0 V: set if (dst) was 100000 C: not affected	Subtract 1 from the contents of the destination.
NEG NEGB Negate	0054DD 1054DD	(dst) ← -(dst)	N: set if result is less than 0 Z: set if result is 0 V: set if result is 100000 C: cleared if result is 0	Replaces the contents of the destination address by its 2's complement. Note that 100000 is replaced by itself.
ADC ADCB Add Carry	0055DD 1055DD	(dst) ← (dst) + C	N: set if result is less than 0 Z: set if result is 0 V: set if (dst) is 077777 and C is 1 C: set if (dst) is 177777 and C is 1	Adds the contents of the C-bit into the destination. This permits the carry from the addition of the low-order words/bytes to be carried into the high-order results.

Table 2-2 Single Operand Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
SBC SBCB Subtract Carry	0056DD 1056DD	(dst) ← (dst) -C	N: set if result is less than 0 Z: set if result is 0 V: set if (dst) was 100000 C: cleared if (dst) is 0 and C is 1	Subtracts the contents of the C-bit from the destination. This permits the carry from the subtraction of the low order words/bytes to be subtracted from the high-order part of the result.
TST TSTB Test	0057DD 1057DD	(dst) ← (dst)	N: set if result is less than 0 Z: set if result is 0 V: cleared C: cleared	Sets the condition codes N and Z according to the contents of the destination address.
ROR RORB Rotate Right	0060DD	(dst) ← (dst) rotate right one place.	N: set if high-order bit of the result is set Z: set if all bits of result are 0 V: loaded with the exclusive-OR of the N-bit and the C-bit as set by ROR	Rotates all bits of the destination right one place. The low-order bit is loaded into the C-bit and the previous contents of the C-bit are loaded into the high-order bit of the destination.
ROL ROLB Rotate Left	0061DD 1061DD	(dst) ← (dst) rotate left one place.	N: set if the high order bit of the result word is set (result < 0); cleared otherwise Z: set if all bits of the result word = 0; cleared otherwise V: loaded with the exclusive-OR of the N-bit and C-bit (as set by the completion of the rotate operation) C: loaded with the high order bit of the destination	Rotate all bits of the destination left one place. The high-order bit is loaded into the C-bit of the status word and the previous contents of the C-bit are loaded into the low-order bit of the destination.

Table 2-2 Single Operand Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
ASR ASRB Arithmetic Shift Right	0062DD 1062DD	(dst) ← (dst) shifted one place to the right.	N: set if the high order bit of the result is set (result < 0); cleared otherwise Z: set if the result = 0; cleared otherwise V: loaded from the exclusive-OR of the N-bit and C-bit (as set by the completion of the shift operation) C: loaded from low order bit of the destination	Shifts all bits of the destination right one place. The high-order bit is replicated. The C-bit is loaded from the low-order bit of the destination. ASR performs signed division of the destination by two. also locks bus
ASL ASLB Arithmetic Shift Left	0063DD 1063DD	(dst) ← (dst) shifted one place to the left.	N: set if high-order bit of the (result < 0); cleared otherwise Z: set if the result = 0; cleared otherwise V: loaded with the exclusive-OR of the N-bit and C-bit and C-bit (as set by the completion of the shift operation) C: loaded with the high-order bit of the destination	Shifts all bits of the destination left one place. The low-order bit is loaded with a 0. The C-bit of the status word is loaded from the high-order bit of the destination. ASL performs a signed multiplication of the destination by 2 with overflow indication

Mnemonic	Op Code	Operation	Condition Codes	Description
ASH Arithmetic Shift	072RSS	R R Shifted Arithmetically NN places to right or left Where NN = (src)	N: set if result<0; cleared otherwise. Z: set if result = 0; cleared otherwise. V: set if sign of register changed during shift; cleared otherwise. C: loaded from last bit shift out of register.	The contents of the register are shifted right or left the number of times specified by the source operand. The shift count is taken as the low-order 6 bits of the source operand. This number ranges from -32 to +31. Negative is a right shift and positive is a left shift.
ASHC Arithmetic Shift Combined	073RSS	R, Rvl← R, Rvl The double word is shifted NN places to the right or left, where NN = (src)	N: set if result <0; cleared otherwise. Z: set if result = 0; cleared otherwise. V: set if sign bit changes during shift; cleared otherwise. C: loaded with high-order bit when right shift (loaded with the last bit shifted out of the 32-bit operand).	The contents of the register R and Rvl are treated as one 32-bit register and shift count (low order 6 bits of source operand). This number ranges from -32 to +31. Negative is right shift and positive is a left shift. Condition codes are affected by the 32-bit result. Bits <31:16> of the result are stored in R if R is even. Bits <15:0> are stored in Rvl. R can be odd or even. If R is odd, the left shift works like a 16-bit left shift. A right shift works like a 16-bit right rotate for shift counts up to 16. If a right shift by more than 16-bits is specified the operation results in an arithmetic right shift by an amount equal to the shift count less 16.
SXT Sign Extend	0067DD	(dst)← 0 if N bit is clear (dst)← -1 N bit is set	N: unaffected Z: set if N bit clear V: cleared C: unaffected	If the condition code bit N is set then a -1 is placed in the destination operand: If N bit is clear, then a 0 is placed in the destination operand. This instruction is particularly useful in multiple precision arithmetic because it permits the sign to be extended through multiple words.

Table 2-2 Single Operand Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
SWAB Swap Byte	0003DD	Byte 1/Byte 0 Byte 0/Byte 1	N: set if high-order bit of low-order byte (bit 7) of result is set; cleared otherwise. Z: set if low-order byte of result = 0; cleared otherwise. V: cleared C: cleared	Exchanges high-order byte and low-order byte of the destination word (destination must be a word address).

Table 2-3 Double Operand Instructions

Mnemonic	OP Code	Operation	Condition Codes	Description
MOV MOVB Move	01SSDD* 11SSDD	(dst) ← (src) ‡	N: set if (src) < 0, cleared otherwise Z: set if (src) = 0, cleared otherwise V: cleared C: not affected	Word: Moves the source operand to the destination location. The previous contents of the destination are lost. The source operand is not affected. Byte: Same as MOV. The MOVB to a resistor (unique among byte instructions) extends the most significant bit of the low order byte (sign extension). Otherwise MOVB operates on bytes exactly as MOV operates on words.
CMP CMPB Compare	02SSDD 12SSDD	(src) - (dst) [in detail, (src) + ~ (dst) + 1]	N: set if result < 0, cleared otherwise Z: set if result = 0; cleared otherwise V: set if there was arithmetic overflow (i.e., operands were of opposite signs and the sign of the destination was the same as the sign of the result); cleared otherwise. C: cleared if there was a carry from the most significant bit of the result, set otherwise	Compares the source and destination operands and sets the condition codes which may then be used for arithmetic and logical conditional branches. Both operands are unaffected. The only action is to set the condition codes. The compare is customarily followed by a conditional branch instruction. Note that unlike the subtract instruction the order of operation is (src) - (dst) not (dst) - (src).

Table 2-3 Double Operand Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
BIT BITB Bit Test	03SSDD 13SSDD	(src) ∧ (dst)	N: set if high order bit of result set; cleared otherwise Z: set if result = 0; cleared otherwise V: cleared C: not affected	Performs logical AND comparison of the source and destination operands and modifies condition codes accordingly. Neither the source nor destination operands are affected. The BIT instruction may be used to test whether any of the corresponding bits that are set in the destination are clear in the source.
BIC BICB Bit Clear	04SSDD 14SSDD	(dst) ← ~ (src) ∧ (dst)	N: set if high order bit of result set; cleared otherwise Z: set if result = 0; cleared otherwise V: cleared C: not affected	Clears each bit in the destination that corresponds to a set bit in the source. The original contents of the destination are lost. The contents of the source are unaffected.
BIS BISB Bit Set	05SSDD 15SSDD	(dst) ← (src) ∧ (dst)	N: set if high order bit of result set; cleared otherwise Z: set if result = 0; cleared otherwise V: cleared C: not affected	Performs inclusive-OR operation between the source and destination operands and leaves the result at the destination address; i.e., corresponding bits set in the destination. The contents of the destination are lost.
ADD Add	06SSDD	(dst) ← (src) + (dst)	N: set if result 0; cleared otherwise Z: set if result = 0; cleared otherwise	Adds the source operand to the destination operand and stores the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. Two's complement addition is performed.

Table 2-3 Double Operand Instruction (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
ADD (Cont)			V: set if there was arithmetic overflow as a result of the operation (that is, both operands were of the same sign and the result was of the opposite sign); cleared otherwise. C: set if there was a carry from the most significant bit of the result; cleared otherwise.	
SUB Subtract	16SSDD	(dst) ← (dst) - (src) in detail, (dst) + ~ (src) + 1 (dst)	N: set if result < 0, cleared otherwise Z: set if result = 0, cleared otherwise V: set if there was arithmetic overflow as a result of the operation (i.e., if operands were of opposite signs and the sign of the source was the same as the sign of the result); cleared otherwise C: cleared if there was a carry from the most significant bit of the result; set otherwise	Subtracts the source operand from the destination operand and leaves the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. In double precision arithmetic, the C-bit, when set, indicates a borrow.

* SS = source (address mode and register)
† (src) = source contents

Table 2-3 Double Operand Instructions (Cont.)

Mnemonic	OP Code	Operation	Condition Codes	Description
MUL Multiply	070RSS	R,Rv1 ←Rx(src)	N: set if product is<0: cleared otherwise. Z: set if product is 0; cleared otherwise. V: cleared C: set if the result is less than -2^{15} or greater than or equal to $2^{15}-1$.	The contents of the destination register and source taken as two's complement integers are multiplied and stored in the destination register and the succeeding register (if R is even). If R is odd, only the low-order product is stored. Assembler syntax is: MUL S,R. (Note that the actual destination is R,Rv1 which reduces to just R when R is odd.)
DIV Divide	071RSS	R,Rv1←R,Rv1 (src)	N: unpredictable if V is set. Set if quotient 0; cleared otherwise. Z: unpredictable if V is set. Set if quotient =0; cleared otherwise. V: set if source = 0 or overflow (quotient less than -2^{15} or greater than $2^{15}-1$). C: set if source = 0; cleared otherwise.	If division by zero is attempted the instruction is terminated and the destination operand is left unchanged. Otherwise, the 32-bit two's complement integer in R and Rv1 is divided by the source operand. The quotient is left in R; the remainder in Rv1. Non-zero remainder always has the same sign as the dividend. If the quotient cannot be represented as a 16 bit two's complement integer, overflow occurs. In this case the instruction aborts and the contents of the destination registers are unpredictable. If R is odd or if R6 is used the result is unpredictable.
XOR	074RDD	(dst)←Rv(dst)	N: set if the result <0; cleared otherwise. Z: set if result=0; cleared otherwise. V: cleared C: unaffected	The exclusive OR of the register and destination operand is stored in the destination address. Contents of register are unaffected. Assembler format is XOR R,D.

Table 2-4 Program Control Instructions

Mnemonic	OP Code	Operation	Condition Codes	Description
BR Branch	000400 xxx†	PC ← PC + (2 X offset)	Unaffected	Provides a way of transferring program control within a range of -128 to +127 words with a one word instruction. It is an unconditional branch.
BNE Branch if not equal	001000 xxx	PC ← PC + (2 X offset) if Z = 0	Unaffected	Tests the state of the Z-bit and causes a branch if the Z-bit is is clear. BNE is the complementary operation to BEQ. It is used to test inequality following a CMP, to test that some bits set in the destination were also in the source, following a BIT, and generally, to test that the result of the previous operation was not 0.
BEQ Branch if equal	001400 xxx	PC ← PC + (2 X offset) if Z = 1	Unaffected	Tests the state of the Z-bit and causes a branch if Z is set. As an example, it is used to test equality following a CMP operation, to test that no bits set in the destination were also set in the source following a BIT operation, and generally, to test that the result of the previous operation was 0.
BGE Branch if greater than or equal	002000 xxx	PC ← PC + (2 X offset) if N v V = 0	Unaffected	Causes a branch if N and V are either both clear or both set. BGE is the complementary operation to BLT. Thus, BGE always causes a branch when it follows an operation that caused addition to two positive numbers. BGE also causes a branch on a 0 result.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
BLT Branch if less than	002400 xxx	PC ← PC + (2 X offset) if N V = 1	Unaffected	Causes a branch if the exclusive-OR of the N- and V-bits are 1. Thus, BLT always branches following an operation that added two negative numbers, even if overflow occurred. In particular, BLT always causes a branch if it follows a CMP instruction operating on a negative source and a positive destination (even if overflow occurred). Further, BLT never causes a branch when it follows a CMP instruction operating on a positive source and negative destination. BLT does not cause a branch if the result of the previous operation was 0 (without overflow).
BGT Branch if greater than	003000 xxx	PC ← PC + (2 X offset) if Z v (N⊻ V) = 0	Unaffected	Operation of BGT is similar to BGE, except BGT does not cause a branch on a 0 result.
BLE Branch if less than or equal to	003400 xxx	PC ← PC + (2 X offset) if Z v (N⊻V) = 1	Unaffected	Operation is similar to BLT, but in addition will cause a branch if the result of the previous operation was 0.
BPL Branch if plus	100000 xxx	PC ← PC + (2 X offset) if N = 0	Unaffected	Tests the state of the N-bit and causes a branch if N is clear. BPL is the complementary operation of BMI.
BMI Branch if minus	100400 xxx	PC ← PC + (2 X offset) if N = 1	Unaffected	Tests the state of the N-bit and causes a branch if N is set. It is used to test the sign (most significant bit) of the result of the previous operation.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
BHI Branch if higher	101000 xxx	PC ← PC + (2 X offset) if C = 0	Unaffected	Causes a branch if the previous operation causes neither a carry nor a 0 result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.
BLOS Branch if lower or same	101400 xxx	PC ← PC + (2 X offset) if C v Z = 1	Unaffected	Causes a branch if the previous operation caused either a carry or a 0 result. BLOS is the complementary operation to BHI. The branch occurs in comparison operations as long as the source is equal to or has a lower unsigned value than the destination. Comparison of unsigned values with the CMP instruction to be tested for "higher or same" and "higher" by a simple test of the C-bit.
BVC Branch if V-bit clear	102000 xxx	PC ← PC + (2 X offset) if V = 0	Unaffected	Tests the state of the V-bit and causes a branch if the V-bit is clear. BVC is complementary operation to BVS.
BVS Branch if V-bit set	102400 xxx	PC ← PC + (2 X offset) if V = 1	Unaffected	Tests the state of V-bit (overflow) and causes a branch if the V-bit is set. BVS is used to detect arithmetic overflow in the previous operation.
BCC BHIS Branch if carry clear Branch if higher than the same	103000 xxx	PC ← PC + (2 X offset) if C = 0	Unaffected	Tests the state of the C-bit and causes a branch if C is clear. BCC is the complementary operation to BCS.
BCS BLO Branch if carry set Branch if lower	103400 xxx	PC ← PC + (2 X offset) if C = 1	Unaffected	Tests the state of the C-bit and causes a branch if C is set. It is used to test for a carry in the result of a previous operation.

Mnemonic	OP Code	Operation	Condition Codes	Description
JMP Jump	0001DD	PC←(dst)	Unaffected	JMP provides more flexible program branching than provided with the branch instruction. Control may be transferred to any location in memory (no range limitation) and can be accomplished with the full flexibility of the addressing modes with the exception of register mode 0. Execution of a jump with mode 0 will cause an illegal instruction condition. (Program control cannot be transferred to a register.) Register deferred mode is legal and will cause program control to be transferred to the address held in the specified register. Note that instructions are word data and must therefore be fetched from an even numbered address. A boundary error trap condition will result when the processor attempts to fetch an instruction from an odd address.
SPL Set Priority Level	00023N	PS <7:5> ← priority N	Unaffected	The least significant three bits of the instruction are loaded into the Program Status Word(PS) bits 7-5 thus causing a changed priority. The old priority is lost. Assembler syntax is: SPL N Note: This instruction is a no op in User and Supervisor modes.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
JRS Jump to subroutine	004RDD	(tmp) ← (dst) (tmp is an internal processor register) ↓ (SP) ← reg (push reg contents onto processor stack) reg ← PC PC holds location following JSR; this address PC ← (tmp), now put in (reg)	Unaffected	In execution of the JSR, the old contents of the specified register (the linkage pointer) are automatically pushed onto the processor stack and new linkage information placed in the register. Thus, subroutines nested within subroutines to any depth may all be called with the same linkage register. There is no need either to plan the maximum depth at which any particular subroutine will be called or to include instructions in each routine to save and restore the linkage pointer. Further, since all linkages are saved in a re-entrant manner on the processor stack, execution of a subroutine may be interrupted, and the same subroutine re-entered and executed by an interrupt service routine. Execution of the initial subroutine can then be resumed when other requests are satisfied. This process (called nesting) can proceed to any level. JSR PC, dst is a special case of the PDP-11 subroutine call suitable for subroutine calls that transmit parameters.
RTS Return from subroutine	00020R	PC ← (reg) (reg) ← SP ↑	Unaffected	Loads contents of register into PC and pops the top element of the processor stack into the specified register. Return from a non-re-entrant subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exits with an RTS PC, and a subroutine called with a JSR R5, dst may pick up parameters with addressing modes (R5) +, X (R5), or @X (R5) and finally exit, with an RTS R5.

Mnemonic	OP Code	Operation	Condition Codes	Description
RTI	000002	PC←(SP)↑ PSW←(SP)↑	N: loaded from processor stack Z: loaded from processor stack V: loaded from processor stack C: loaded from processor stack	Used to exit from an interrupt or trap service routine. The PC and PSW are restored (popped) from the processor stack. If the RTI sets the T-bit in the PSW, a trace trap will occur prior to executing the next instruction.
RTT	000006	PC←(SP)↑ PS←(SP)↑	N: loaded from processor stack Z: loaded from processor stack V: loaded from processor stack C: loaded from processor stack	This is the same as the RTI instruction, except that it inhibits a trace trap, while RTI permits a trace trap. If a trace trap is pending, the first instruction after the RTT will be executed prior to the next "T" trap. In the case of the RTI instruction, the "T" trap will occur immediately after the RTI.

In RTT and RTI, only transactions from more privileged processor modes to the same or less privileged modes are allowed. When executed in Supervisor mode, the new PSW bits cannot be Kernel. When executed in User mode, the new PSW mode bits can only be User.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description

MARK	0064NN	SP ← SP + 2xnn
		PC ← R5
		R5 ← (SP) +
		nn = number of parameters
			Unaffected	Used as part of the standard PDP-11 subroutine return convention. MARK facilitates the stack cleanup procedures involved in subroutine exit. Assembler format is: MARK N
				
				Example: MOV R5,-(SP) ;place old R5 on stack
				MOV P1,-(SP) ;place N parameters on
				MOV P2,-(SP) ;the stack to be used
				;there by the subroutine
				
				MOV PN,-(SP) ;places the instruction
				MOV #MARKN,-(SP) ;MARK N on the stack
				;set up address at Mark
				MOV SP,R5 ;N instruction
				
				JSR PC,SUB ;jump to subroutine
				
				At this point the stack is as follows:
				
				OLD R5
				P1
				PN
				MARK N
				OLD PC
				
				And the program is at the address SUB which is the beginning of the subroutine.
				SUB: ;execution of the subroutine itself
				RTS R5: ;the return begins
				
				This causes the contents of R5 to be placed in the PC which then results in the execution of the instruction MARK N. The contents of old PC are placed in R5
				
				MARK N causes: (1) the stack pointer to be adjusted to point to the old R5 value; (2) the value now in R5 (the old PC) to be placed in the PC, and (3) contents of the old R5 to be popped into R5, thus completing the return from subroutine.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
SOB Subtract one and branch if not equal to 0	077R00 plus offset	R ← R - 1 if this result ≠ 0 then PC ← PC -(2 x offset)	Unaffected	The register is decremented. If it is not equal to 0, twice the offset is subtracted from the PC (now pointing to the following word). The offset is interpreted as a six-bit positive number. This instruction provides a fast, efficient method of loop control. Assembler syntax is: SOB R,A where A is the address to which transfer is to be made if the decremented R is not equal to 0. Note that the SOB instruction cannot be used to transfer control in the forward direction.
BPT Break-point Trap	000003	↓ (SP) ← PS ↓ (SP) ← PC PC ← (14) PS ← (16)	N: loaded from trap vector Z: loaded from trap vector V: loaded from trap vector C: loaded from trap vector	Performs a trap sequence with a trap vector address of 14. Used to call debugging aids. The user is cautioned against employing code 000003 in programs run under these debugging aids.
IOT IOT Trap	000004	↓ (SP) ← PS ↓ (SP) ← PC PC ← (20) PS ← (22)	N: loaded from trap vector Z: loaded from trap vector C: loaded from trap vector	Performs a trap sequence with a trap vector address of 20. Used to call the I/O executive routine IOX in the paper-tape software system and for error reporting in the disk operating system.

Table 2-4 Program Control Instructions (Cont)

Mnemonic	OP Code	Operation	Condition Codes	Description
EMT Emulator Trap	104000	↓(SP) ← PS ↓(SP) ← PC PC ← (30) PS ← (32)	N: loaded from trap vector Z: loaded from trap vector V: loaded from trap vector C: loaded from trap vector	All operation codes from 104000 to 104377 are EMT instructions and may be used to transmit information to the emulating routine (e.g., function to be performed). The trap vector for EMT is at address 30; the new central processor status (PS) is taken from the word at address 32. CAUTION **EMT is used frequently by DEC system software and is therefore not recommended for general use.**
TRAP	104400 to 104777	↓(SP) ← PS ↓(SP) ← PC PC ← (34) PS ← (36)	N: loaded from trap vector Z: loaded from trap vector V: loaded from trap vector C: loaded from trap vector	Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34. NOTE **Since DEC software makes frequent use of EMT, the TRAP instruction is recommended for general use.**

NOTE: Condition Codes are unaffected by these instructions

*DD = destination (address mode and register)
†(dst) = destination contents

Table 2-5 Miscellaneous Instructions (Cont.)

Mnemonic	OP Code	Operation	Condition Codes	Description
WAIT	000001		Unaffected	Provides a way for the processor to relinquish use of the bus while it waits for an external interrupt. Having been given a WAIT command, the processor will not compete for bus by fetching instructions or operands from memory. This permits higher transfer rates between device and memory, as no processor-induced latencies will be encountered by bus requests from the device. In WAIT, as in all instructions, the PC points to the next instruction following the WAIT operation. Thus, when an interrupt causes the PC and PS to be pushed onto the stack, the address of the next instruction following the WAIT is saved. The exit from the interrupt routine (i.e., execution of an RTI instruction) will cause resumption of the interrupted process at the instruction following the WAIT.
RESET	000005	PC(SP) PSW(SP)	Unaffected	Sends INIT on the Unibus for 100 ms. All devices on the Unibus are reset to their state at power-up.
MFPT	~~000007~~ ~~076010~~	R0<7:0>← Processor Code Code 1 R0<15:8>← Processor Subcode	Unaffected	Upon execution, the MFPT instruction returns to the low byte of R0 a processor model code (octal 3 for PDP 11/68.) The high byte of R0 will be loaded with a processor specified subcode (octal Ø for PDP 11/68.

Table 2-5 Miscellaneous Instructions (Cont.)

Mnemonic	OP Code	Operation	Condition Codes	Description
MFPI MFPD	006555 106555	(temp)←(src) ↓(sp)←(temp)	N: set if the source <0; otherwise cleared Z: set if the source =0; otherwise cleared V: cleared C: unaffected	This instruction is provided in order to allow inter-address space communication when the PDP 11/68 is using the Memory Management unit. The address of the source operand is determined in the current address space. That is, the address is determined using the SP and memory pages determined by PS<15:14>. The address itself is then used in the previous I(D) space (as determined by PS<13:12> to get the source operand. This operand is then pushed on to the current R6 stack.
MTPI MTPD	006655 106655	(temp)←(SP)↑ (dst)←(temp)	N: set if the source <0; otherwise cleared Z: set if the source =0; otherwise cleared V: cleared C: unaffected	The address of the destination operand is determined in the current address space. MTPI(D) then pops a word off the current stack and stores that word in the destination address in the previous mode's I(D) space (bits 13, 12 of PS).
HALT	000000		Unaffected	Causes the processor operation to cease. The console is given control of the processor. The console data lights display the address of the HALT instruction plus two. Transfers on the Unibus are terminated immediately. The PC points to the next instruction to be executed. Pressing the CONT key on the console causes processor operation to resume. No INIT signal is given.

Table 2-6 Condition Code Operators

Mnemonic	Op Code	Instruction
CLC	000241	Clear condition code C.
CLV	000242	Clear condition code V.
CLZ	000244	Clear condition code Z.
CLN	000250	Clear condition code N.
CCC	000257	Clear all condition code bits.
SEC	000261	Set condition code C.
SEV	000262	Set condition code V.
SEZ	000264	Set condition code Z.
SEN	000270	Set condition code N.
SCC	000277	Set all condition code bits.

NOTE

Selectable combinations of condition code bits may be cleared or set together. The status of bit 4 controls the way in which bits 0, 1, 2, and 3 are to be modified. If bit 4 = 1, the specified bits are set; if bit 4 = 0, the specified bits are cleared.

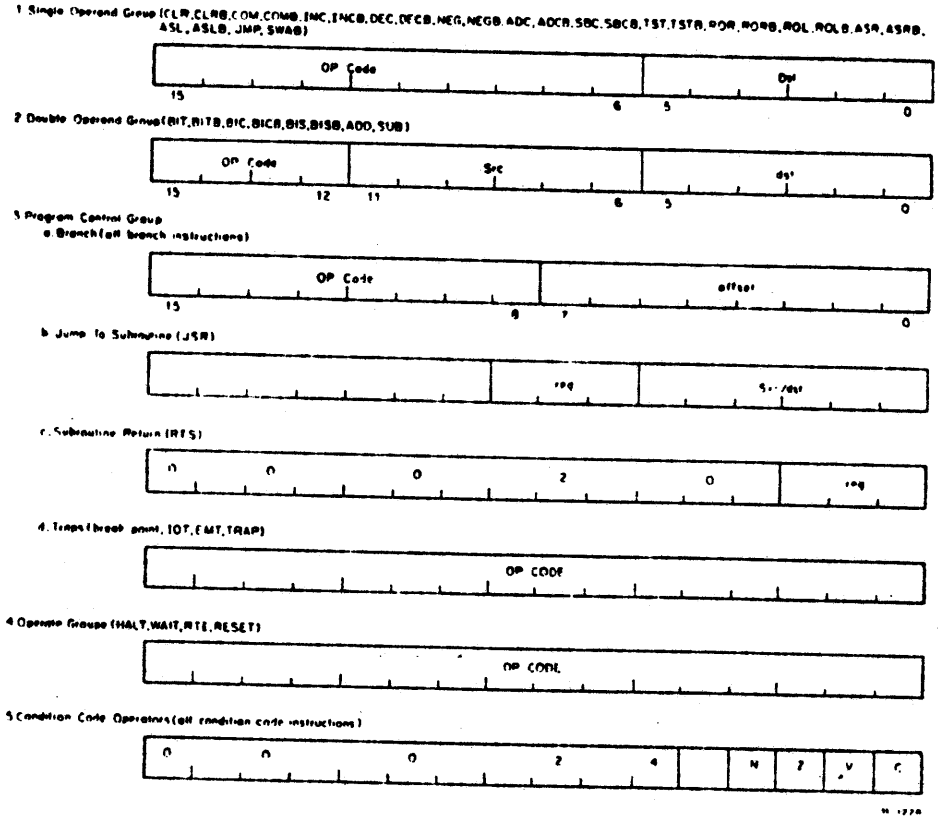


Figure 2-2 PDP-11 Instruction Formats

Table 2-7 Programming Differences

	44	04	34	F11	LSI11	05/10	15/20	35/40	45	70	68
1. OPR%R, (R)+ or OPR%R, -(R) using the same register as both source and destination: contents of R are incremented (decremented) by 2 before being used as the source operand.				X			X	X			
OPR%R, (R)+ or OPR%R, -(R) using the same register as both register and destination: initial contents of R are used as the source operand.	X	X	X		X	X			X	X	X
2. OPR%R, @(R)+ or OPR%R, @-(R) using the same register as both source and destination: contents of R are incremented (decremented) by 2 before being used as the source operand.				X			X	X			
OPR%R, @(R)+ or OPR%R, @-(R) using the same register as both source and destination: initial contents of R are used as the source operand.	X	X	X		X	X			X	X	X
3. OPR PC, X(R); OPR PC, @ X(R); OPR PC, @ A; OPR PC, A: Location A will contain the PC of OPR +4.				X			X	X			
OPR PC, X(R); OPR PC, @ X(R), OPR PC, A; OPR PC, @ A: Location A will contain the PC of OPR +2.	X	X	X		X	X			X	X	X
4. JMP (R)+ or JSR reg, (R)+: Contents of R are incremented by 2, then used as the new PC address						X	X				
JMP (RP)+ or JSR reg. (R)+: Initial contents of R are used as the new PC.	X	X	X	X	X			X	X	X	X
5. JMP %R or JSR reg, %R traps to 4 (illegal instruction).		X	X	X	X	X	X	X			

Table 2-7 Programming Differences

	44	04	34	F11	LSI11	05/10	15/20	35/40	45	70	68
JMP %R or JSR reg, %R traps to 10 (illegal instruction).	X								X	X	X
6. SWAB does <u>not</u> change V.							X				
SWAB clears V.	X	X	X	X	X	X		X	X	X	X
7. Register addresses (177700-177717) are valid Program addresses when used by CPU.						X					
Register addresses (177700-177717) time out when used as a program address by the CPU. Can be addressed under console operation. ← Maybe Note addresses cannot be addressed under Console for LSI-11 or F11.	X	X	X	X	X		X	X	X	X	X
8. <u>Basic Instructions</u> noted in PDP-11 processor handbook.	X	X	X	X	X	X	X	X	X	X	X
SOB, MARK, RTT, SXT instructions	X		X	X	X			X	X	X	X
ASH, ASHC, DIV, MUL, XOR	X		X	X	X			X	X	X	X
MFPT Instruction	X										X
The external option KE11-A provides MUL, DIV, SHIFT operation in the same data format.						X	X				
The KE11-E (Expansion Instruction Set) provides the instructions MUL, DIV, ASH, and ASHC. These new instructions are 11/45 compatible.								X			
The KE11-F adds unique stack ordered floating point instructions: FADD, FSUB, FMUL, FDIV.								X			
The KEV-11 adds EIS/FIS instructions					X						
SPL Instruction									X	X	X

Table 2-7 Programming Differences (Cont.)

	44	04	34	F11	LSI11	05/10	15/20	35/40	45	70	68
9. Power fail during RESET instruction is not recognized until after the instruction is finished (70 milliseconds). RESET instruction consists of 70 millisecond pause with INIT occurring during first 20 milliseconds.							X	X			
Power fail immediately ends the RESET instruction and traps if an INIT is in progress. A minimum INIT of 1 microsecond occurs if instruction aborted. PDP 11/04/34/44 are similar with no minimum INIT time.	X	X	X						X	X	X
Power fail acts the same as 11/45 (22 milliseconds with about 300 nanoseconds minimum). Power fail during RESET fetch is fatal with no power down sequence.						X					
RESET instruction consists of 10 Usec of INIT followed by a 90 Usec pause. Power fail not recognized until the instruction is complete.				X	X						
10. No RTT instruction						X	X				
If RTT sets the T bit, the T bit trap occurs after the instruction following RTT.	X	X	X	X	X			X	X	X	X
11. If RTI sets "T" bit, "T" bit trap is acknowledged after instruction following RTI.						X	X				
If RTI sets "T" bit, "T" bit trap is acknowledged immediately following RTI.	X	X	X	X	X			X	X	X	X
12. If an interrupt occurs during an instruction that has the "T" bit set, the "T" bit trap is acknowledged before the interrupt.	X	X	X	X	X	X	X	X			
If an interrupt occurs during an instruction and the "T" bit is set, the interrupt is acknowledged before "T" bit trap.									X	X	X

Table 2-7 Programming Differences (Cont.)

	44	04	34	F11	LSI11	05/10	15/20	35/40	45	70	68
13. "T" bit trap will sequence out of WAIT instruction.	X	X	X	X		X	X	X			
"T" bit trap will not sequence out of WAIT instruction. Waits until an interrupt.					X				X	X	X
14. Explicit reference (direct access) to PS can load "T" bit. Console can also load "T" bit.		X				X	X				
Only implicit references (RTI, RTT, traps and interrupts) can load "T" bit. Console cannot load "T" bit.	X		X	X	X			X	X	X	X
15. Odd address/non-existent references using the SP cause a HALT. This is a case of double bus error with the second error occurring in the trap servicing the first error. Odd address trap not in LSI-11 or F-11.	X	X	X	X	X	X	X				
Odd address/non-existent references using the stack pointer cause a fatal trap. On bus error in trap service, new stack created at 0/2.								X	X	X	X
16. The first instruction in an interrupt routine will not be executed if another interrupt occurs at a higher priority level than assumed by the first interrupt.	X	X	X	X	X	X		X	X	X	X
The first instruction in an interrupt service is guaranteed to be executed.							X				
17. 8 General purpose registers.	X	X	X	X	X	X	X	X			
16 General purpose registers.									X	X	X

Table 2-7 Programming Differences (Cont.)

	44	04	34	F-11	LSI11	05/10	15/20	35/40	45	70	68
18. PSW address, 177776, not implemented must use new instructions, MTPS (move to PS) and MFPS (move from PS).					X						
PSW address implemented, MTPS and MFPS not implemented.	X	X				X	X	X	X	X	X
PSW address and MTPS and MFPS implemented.			X	X							
19. Only one interrupt level (BR4) exists.					X						
Four interrupt levels exist.	X	X	X	X		X	X	X	X	X	X
20. Stack overflow not implemented.					X						
Some sort of stack overflow implemented.	X	X	X	X		X	X	X	X	X	X
21. Odd address trap not implemented.				X	X						
Odd address trap implemented.	X	X	X			X	X	X	X	X	X
22. FMUL and FDIV instructions implicitly use R6 (one push and pop); hence R6 must be set up correctly.					X						
FMUL and FDIV instructions do not implicitly use R6.								X			
23. Due to their execution time, EIS instructions can abort because of a device interrupt.					X						
EIS instructions do not abort because of a device interrupt.	X		X	X				X	X	X	X
24. Due to their execution time, FIS instructions can abort because of a device interrupt.					X			X			
25. EIS instructions do a DATIP and DATO bus sequence when fetching source operand.					X						

Table 2-7 Programming Differences (Cont.)

	44	04	34	F-11	LSI11	05/10	15/20	35/40	45	70	68
EIS instructions do a DATI bus sequence when fetching source operand.	X		X	X				X	X	X	X
26. MOV instruction does just a DATO bus sequence for the last memory cycle.	X		X	X	X			X	X	X	X
MOV instruction does a DATIP and DATO bus sequence for the last memory cycle.		X				X	X				
27. If PC contains non-existent memory address and a bus error occurs, PC will have been incremented.	X	X	X	X	X	X	X		X	X	X
If PC contains non-existent memory address and a bus error occurs, PC will be unchanged.								X			
28. If register contains non-existent memory address in mode 2 and a bus error occurs, register will be incremented.				X	X	X	X	X	X	X	X
Same as above but register is unchanged.	X	X	X								
29. If register contains an odd value in mode 2 and a bus error occurs, register will be incremented.				X	X			X	X	X	X
If register contains an odd value in mode 2 and a bus error occurs, register will be unchanged.	X	X	X			X	X				
30. Condition codes restored to original values after FIS interrupt abort (EIS doesn't abort on 35/40)								X			
Condition codes that are restored after EIS/FIS interrupt abort are indeterminate.					X						
31. OP codes 075040 through 075377 unconditionally trap to 10 as reserved Op codes.	X	X	X	X		X	X	X	X	X	X

Table 2-7 Programming Differences (Cont.)

	44	04	34	F-11	LSI11	5/10	15/20	35/40	45	70	68
If KEV-11 option is present, Op codes 75040 through 07533 perform a memory read using the register specified by the low order 3 bits as a pointer. If the register contents are a non-existent address, a trap to 4 occurs. If the register contents are an existent address, a trap to 10 occurs.					X						
32. Op codes 210 thru 217 trap to 10 as reserved Op codes.	X	X	X	X		X	X	X	X	X	X
Op codes 210 thru 217 are used as a maintenance instruction.					X						
33. Op codes 75040 thru 75777 trap to 10 as reserved Op codes.	X	X	X	X		X	X	X	X	X	X
Only if KEV-11 option is present, Op codes 75040 thru 75377 can be used as escapes to user microcode. Op codes 75400 thru 75777 can also be used.					X						
As escapes to user microcode and KEV-11 option need not be present. If no user microcode exists, a trap to 10 occurs.											
34. Op codes 170000 thru 177777 trap to 10 as reserv reserved instructions.		X				X	X	X			
Op codes 170000 thru 177777 are implemented as floating point instructions.	X		X	X					X	X	X
Op codes 170000 thru 177777 can be used as escapes to user microcode. If no user microcode exists, a trap to 10 occurs.					X						

Table 2-7 Programming Differences (Cont.)

	44	04	34	F-11	LSI11	05/10	15/20	35/40	45	70	68
35. CLR and SXT do just a DATO sequence for the last bus cycle.				X							
CLR and SXT do DATIP-DATO sequence for the last bus cycle.	X	X	X		X	X	X	X	X	X	X
36. MEM.MGT maintenance mode SRØ bit 8 is implemented.	X		X					X	X	X	X
MEM.MGT maintenance mode SRØ bit 8 is not implemented.				X							
37. PS<15:12>, user mode, user stack pointer, and MTPX and MFPX instructions exist even when MEM.MGT is not configured.	X			X					X	X	X
PS<15:12>, user mode, user stack pointer, and MTPX and MFPX instructions exist only when MEM.MGT is configured.								X			
38. Current mode PS bits <15:14> set to illegal mode will cause a MEM.MGT trap upon any memory reference (Ø1 is illegal in 34, 60, 35/40)	X		X					X	X	X	X
Current mode PS bits <15:14> set to 01 or 10 will be treated as kernel mode (00) and not cause a MEM.MGT trap.				X							
39. MTPS in user mode will cause MEM.MGT trap if PS address 177776 not mapped. If mapped PS <7:5> and <3:0> affected.			X								
MTPS in user mode will only affect PS <3:0> regardless of whether PS address 177776 is mapped.				X							
40. MFPS in user mode will cause MEM.MGT trap if PS address 177776 not mapped. If mapped, PS <7:0> are accessed.			X								

Table 2-7 Programming Differences (Cont.)

	44	04	34	F-11	LSI11	05/10	15/20	35/40	45	70	68
MFPS in user mode will access PS <7:0> regardless of whether PS address 177776 is mapped.				X							
41. A HALT instruction in user mode traps to 4.	X								X	X	X
A HALT instruction in user mode traps to 10.			X	X				X			

Hardware Differences -- Traps
(Transparent to Software)

LSI-11

Priority of processor traps.

Bus error trap
Memory refresh
TRAP Instructions
TRACE Trap
Power Fail Trap
HALT LINE
Event Line Interrupt
Device (BUS)
Interrupt Request

PDP11/05,10

Priority of internal processor traps, external interrupts, HALT and WAIT:

Bus Error Trap
TRAP Instructions
TRACE Trap
OVFL Trap
PWR Fail Trap
UNIBUS BUS
REQUEST
CONSOLE STOP
WAIT LOOP

PDP11/15,20

Priority of internal processor traps, external interrupts, HALT AND WAIT:

Bus Error Trap
Trap Instructions
TRACE Trap
OVFL Trap
PWR Fail Trap
CONSOLE BUS REQUEST
UNIBUS BUS REQUEST
WAIT LOOP

PDP11/35,40

Priority of internal processor traps, external interrupts, HALT and WAIT:

Memory Parity Errors
Memory Management Fault
BUS ERROR Traps
OVFL Trap (red zone)
TRAP instructions
TRACE Trap
OVFL Trap (yellow zone)
PWR Fail Trap
CONSOLE BUS request
UNIBUS BUS request
WAIT LOOP

PDP11/45,70

Priority of internal processor traps, external interrupts, HALT and WAIT:

Memory Parity Error
Bus Error Traps
TRAP Instructions
CONSOLE BUS Request
Memory Management
OVFL Trap
FLOATING POINT Trap
PROGRAM INTERRUPT request
UNIBUS BUS Request
WAIT LOOP
TRACE Trap

11/04

Same as 11/05

11/34

Same as 11/35 but no red zone stack overflow.

F-11

Same as 11/34

11/68

(See Trap Priorities Section)

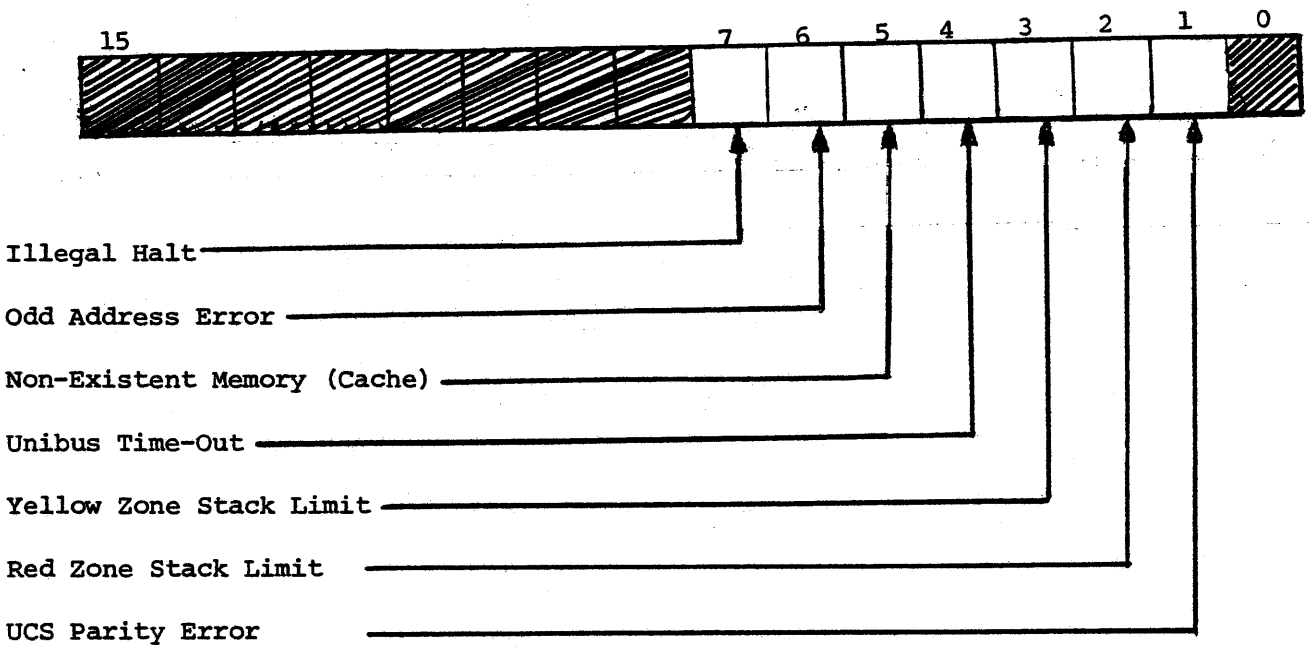
11/44

Same as 11/34 with PROGRAM INTERRUPT Request having higher priority than UNIBUS BUS REQUESTS.

2.2 Processor Control Registers

Processor Registers

CPU Error Register 17 777 766

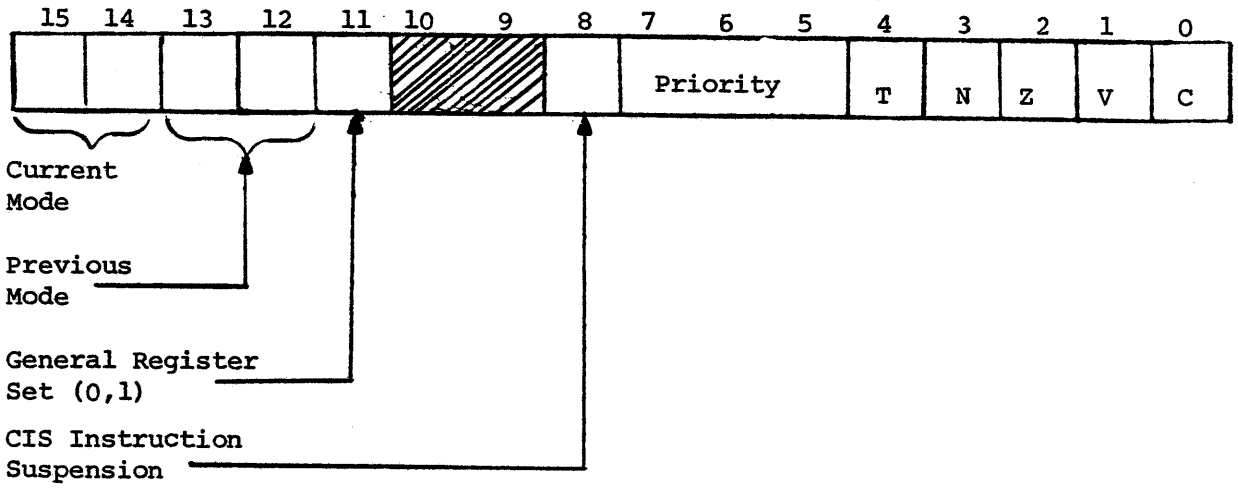


This register identifies the source of the abort or trap that used the vector at location 4.

Bit	Name	Function
7	Illegal Halt	Set when trying to execute a HALT instruction when CPU is in User or Supervisor mode (not Kernel).
6	Odd Address Error	Set when a program attempts to do a word reference to an odd address.
5	Non-Existent Memory	Set when the CPU receives a timeout upon reference to a main memory address. This does not include UNIBUS addresses.
4	UNIBUS Timeout	Set when there is no response on the UNIBUS within approx. 20 usec.
3	Yellow Zone Stack Limit	Set when a yellow zone trap occurs.
2	Red Zone Stack Limit	Set when a red zone trap occurs.
1	UCS Parity Error	Set when a UCS parity error occurs.

NOTE: Any DATO or DABOB to this register clears it.

Processor Status Word 17 777 776

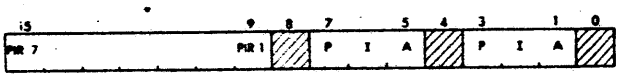


MODE: 00 = Kernel
01 = Supervisor
11 = User

The Processor Status Word contains information on the current status of the CPU. This information includes the register set currently in use; current processor priority; current and previous operational modes; the condition codes describing the results of the last instruction; and an indicator for detecting the execution of an instruction to be trapped during program debugging. The CIS suspension bit indicates that a CIS instruction has been interrupted before completion.

Program Interrupt Register 17 777 772

A request is booked by setting one of the bits 15 through 9 (for PIR 7 -- PIR 1) in the Program Interrupt Register at location 17 777 772. The hardware sets bits 7 - 5 and 3 - 1 to the encoded value of the highest PIR bit set. This Program Interrupt Active (PIA) should be used to set the Processor Level and also index through a table of interrupt vectors for the seven software priority levels. The figure shows the layout of the PIR Register.



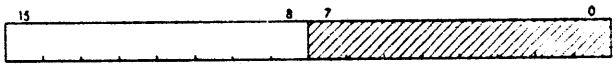
When the PIR is granted, the Processor will Trap to location 240 and pick up PC in 240 and the PSW in 242. It is the interrupt service routine's responsibility to queue requests within a priority level and to clear the PIR bit before the interrupt is dismissed.

STACK LIMIT Register 17 777 774

The Stack Limit allows program control of the lower limit for permissible stack addresses. This limit may be varied in increments of (400)$_8$ bytes or (200)$_8$ words, up to a maximum address of 177 400 (almost the top of a 32K memory).

The normal boundary for stack addresses is 400. The Stack Limit option allows this lower limit to be raised, providing more address space for interrupt vectors or other data that should not be destroyed by the program.

There is a Stack Limit Register, with the following format:



The Stack Limit Register can be addressed as a word at location 17 777774, or as a byte at location 17 777775. The register is accessible to the processor and console, but not to any bus device.

The 8 bits, 15 through 8, contain the stack limit information. These bits are cleared by System Reset, Console Start, or the RESET instruction. The lower 8 bits are not used. Bit 8 corresponds to a value of (400)$_8$ or (256)$_{10}$.

Stack Limit Violations

When instructions cause a stack address to exceed (go lower than) a limit set by the programmable Stack Limit Register, a Stack Violation occurs. There is a Yellow Zone (grace area) of 16 words below the Stack Limit which provides a warning to the program so that corrective steps can be taken. Operations that cause a Yellow Zone Violation are completed, then a bus error trap is effected. The error trap, which itself uses the stack, executes without causing an additional violation, unless the stack has entered the Red Zone.

A Red Zone Violation is a Fatal Stack Error. (Odd stack or non-existent stack are the other Fatal Stack Errors.) When detected, the operation causing the error is aborted, the stack is repositioned to address 4, and a bus error occurs. The old PC and PS are pushed into location 0 and 2, and the new PC and PS are taken from locations 4 and 6.

Stack Limit Addresses

The contents of the Stack Limit Register (SL) are compared to the stack address to determine if a violation has occurred. The least significant bit of the register (bit 8) has a value of (400)$_8$. The determination of the violation zones is as follows:

Yellow Zone = (SL) + (340 through 377)$_8$.	execute, then trap
Red Zone $\leqslant$ (SL) + (337)$_8$.	abort, then trap to location 4

If the Stack Limit Register contents were zero:

Yellow Zone = 340 through 377
Red Zone = 000 through 337

internal pushes not allowed

2.3 Aborts, Traps, Interrupts

Processor Traps

There are a series of errors and programming conditions which will cause the Central Processor to trap to a set of fixed locations. These include Power Failure, Odd Addressing Errors, Stack Errors, Timeout Errors, Non-Existent Memory References, Memory Parity Errors, Memory Management Violations, Floating Point Processor Exception Traps, use of Reserved Instructions, use of the T bit in the Processor Status Word, and use of the IOT, EMT, and TRAP instructions.

Trap Priorities

Aborts

Micro break

UCS Parity Error

Odd address

Red Zone

Memory Management Abort

Cache Parity Abort

Memory Parity Error

Bus Errors (Timeout)

Interrupts and Traps

Trap Instructions

Console interrupt

Cache Parity Trap

Yellow Zone Stack Warning

Power Fail

Floating Point Exception

PIRQ7
BR7

PIRQ6
BR6

PIRQ5
BR5

PIRQ4
BR4

PIRQ3

PIRQ2

PIRQ1

Trace Trap

Processor Trap Vectors

The following summary is the set of conditions and vector locations which causes the processor to trap.

Vector (8)	Conditions	
004	CPU Errors	Illegal HALT, illegal odd-address reference, Non-existant Memory reference (Main-Memory Timeout), Unibus Timeout, Yellow Zone, Red Zone, Stack Violations, Control Store Parity error. Conditions are logged in the CPU Error Register. (See Section on Programming Differences).
010	Illegal and Reserved Instructions	JMP and JSR. Mode Ø plus reserved opcodes.
014	BPT breakpoint and Trace Trap	
020	IOT Input/output Trap	
024	Power Fail/Power up	
030	EMT Emulator Trap	
034	TRAP instruction	
114	Memory System Errors	Cache parity aborts, Cache parity traps, Main Memory aborts. Conditions are logged in Cache/Memory System Error Register. (See section on Programming Differences).
240	PIR	
244	Floating Point exceptions	Conditions logged in FEC & FEA registers accessed via STST instruction.
250	Memory Management aborts	Conditions & information logged in MMRØ, MMR1, MMR2, MMR3,

UCS Parity Error

If a parity error is detected during access to the User Control store, an abort occurs with a resultant trap to vector 4.

Odd Addressing Errors

This error occurs whenever a program attempts to execute a word instruction on an odd address (in the middle of a word boundary). The instruction is aborted and the CPU traps through location 4.

Stack Limit Violations

When instructions cause a stack address to exceed a limit set by the programmable Stack Limit Register, a stack violation occurs resulting in a trap to vector 4. A yellow zone stack violation provides a warning to the program so that corrective steps can be taken. If stack operations result in pushes beyond the 16 word grace area below the stack limit, a red zone Fatal Stack Violation occurs. (See Stack Limit operations in Processor Control section.)

Memory Management Abort

When the memory management unit is enabled (MMRØ<Ø>=1), program access to non-resident pages, write operations to read-only pages, and program references to addresses beyond the limit set for the current pages results in an instruction abort with a resultant trap to vector 240. (See Memory Management Section.)

Memory System Errors

Memory errors resulting from cache parity errors with traps enabled (CCR<Ø> = Ø), cache parity aborts (CCR<Ø7> = 1) enabled, or main memory double bit errors results in a trap to vector 114.

Non-Existent Memory Errors

This error occurs when a program memory reference results in a time-out response. The cycle is aborted and the processor traps through vector 4 with bit <5> set in the CPU Error Register.

UNIBUS Time-out Error

This error occurs when a Master Synchronization pulse is placed on the UNIBUS and there is no slave pulse within 20 usec. This error usually occurs in attempts to address non-existent peripherals.

The offending instruction is aborted and the processor traps through location 4.

Reserved Instructions

There is a set of illegal and reserved instructions which cause the processor to trap through Location 10. All illegal and reserved instructions trap to vector 10 with the exception of an illegal HALT which traps to vector 4 with Bit <7> set in the CPU Error Register.

Power Failure

Whenever AC power drops below 95 volts for 110v power (190 volts for 220V) or outside the limit of 47 to 63 Hz, as measured by DC power, the power fail sequence is initiated. The Central Processor automatically traps to location 24 and the power fail program has 2 msec. to save all volatile information (data in registers), and to condition peripherals for power fail.

When power is restored the processor traps to location 24 and executes the power up routine to restore the machine to its state prior to power failure.

2.4 Memory Management

General

The PDP-11/68 Memory Management Unit provides the hardware facilities necessary for complete memory management and protection. It is designed to be a memory management facility for accessing all of physical memory and for multi-user, multi-programming systems where memory protection and relocation facilities are necessary.

In order to most effectively utilize the power and efficiency of the PDP-11/68 in medium and large scale systems it is necessary to run several programs simultaneously. In such multi-programming environments, several user programs would be resident in memory at any given time. The task of the supervisory program would be: control the execution of the various user programs, manage the allocation of memory and peripheral device resources, and safeguard the integrity of the system as a whole by careful control of each user program.

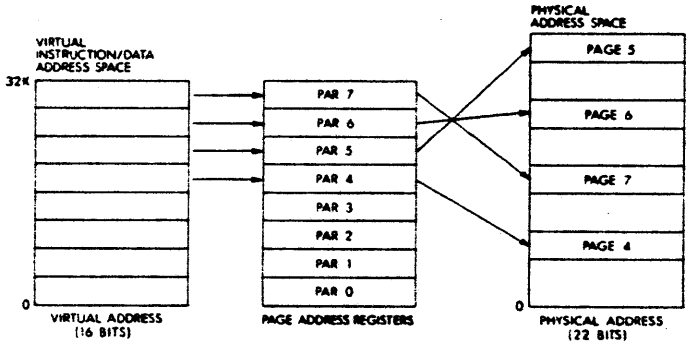
In a multi-programming system, the Memory Management Unit provides the means for assigning memory pages to a user program and preventing that user from making any unauthorized access to these pages outside his assigned area. Thus, a user can effectively be prevented from accidental or willful destruction of any other user program or the system executive program.

The basic characteristics of the PDP-11/68 Memory Management Unit are:

- 16 User mode memory pages
- 16 Supervisor mode memory pages
- 16 Kernel mode memory pages
- 8 pages in each mode for instructions
- 8 pages in each mode for data
- page lengths from 32 to 4096 words
- each page provided with full protection and relocation
- transparent operation
- 4 modes of memory access control
- memory access to 2 million words (4 million bytes)

Virtual Addressing

When the PDP-11/68 Memory Management Unit is operating, the normal 16 bit direct byte address is no longer interpreted as a direct Physical Address (PA) but as a Virtual Address (VA) containing information to be used in constructing a new 22-bit physical address. The information contained in the Virtual Address (VA) is combined with relocation information contained in the Page Address Register (PAR) to yield a 22-bit Physical Address (PA). Using the Memory Management Unit, memory can be dynamically allocated in pages each composed of from 1 to 128 integral blocks of 32 words.



PAR -- Page Address Register

Figure 2-3 Virtual Address Mapping into Physical Address

The starting physical address for each page is an integral multiple of 32 words, and each page has a maximum size of 4096 words. Pages may be located anywhere within the Physical Address space. The determination of which set of 16 pages registers is used to form a Physical Address is made by the current mode of operation of the CPU; i.e., Kernel, Supervisor or User mode.

Interrupt Conditions under Memory Management Control

The Memory Management Unit relocates all addresses. Thus, when it is enabled, all trap, abort, and interrupt vectors are considered to be in Kernel mode Virtual Address Space. When a vectored transfer occurs, control is transferred according to a new Program Counter (PC) and Processor Status Word (PS) contained in a two-word vector relocated through the Kernel Page Address Register Set. Relocation of trap addresses means that the hardware is capable of recovering from a failure in the first physical bank of memory.

When a trap, abort, or interrupt occurs the "push" of the old PC, old PS is to the User/Supervisor/Kernel R6 stack specified by CPU mode bits 15,14 of the new PS in the vector (bits 15,14: 00 = Kernel, 01 = Supervisor, 11 = User). The CPU mode bits also determine the new PAR set. In this manner it is possible for a Kernel mode program to have complete control over service assignments for all interrupt conditions, since the interrupt vector is located in Kernel space. The Kernel program may assign the service of some of these conditions to a Supervisor or User mode program by simply setting the CPU mode bits of the new PS in the vector to return control to the appropriate mode.

Construction of a Physical Address

All addresses with memory relocation enabled either reference information in instruction (I) Space or Data (D) Space. I Space is used for all instruction fetches, index words, absolute addresses and immediate operands, D Space is used for all other references. I Space and D Space each have 8 PAR's in each mode of CPU operation, Kernel, Supervisor, and User. Using Memory Management Register #3, the operating system may select to disable D space and map all references (Instructions and Data) through I space, or to use both I and D space.

The basic information needed for the construction of a Physical Address (PA) comes from the Virtual Address (VA), which is illustrated in Figure 2-4 and the appropriate PAR set.

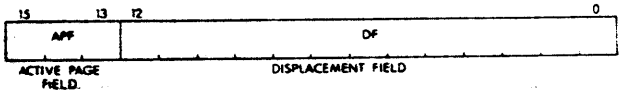


Figure 2-4 Interpretation of a Virtual Address

The Virtual Address (VA) consists of:

1. The Active Page Field (APF). This 3-bit field determines which of eight Page Address Registers (PAR0-PAR7) will be used to form the Physical Address (PA).

2. The Displacement Field (DF). This 13-bit field contains an address relative to the beginning of a page. This permits page lengths up to 4K words (2^{13} = 8K bytes). The DF is further subdivided into two fields as shown in Figure 2-5.

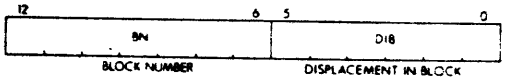


Figure 2-5 Displacement Field of Virtual Address

The Displacement Field (DF) consists of:

1. The Block Number (BN). This 7-bit field is interpreted as the block number within the current page.

2. The Displacement in Block (DIB). This 6-bit field contains the displacement within the block referred to by the Block Number (BN).

The remainder of the information needed to construct the Physical Address comes from the 16-bit Page Address Field (PAF) (the Page Address Register (PAR)) that specifies the starting address of the memory page which that PAR describes. The PAF is actually a block number in the physical memory; e.g., PAF = 3 indicates a starting address of 96 (3 x 32) words in physical memory.

The formation of the Physical Address (PA) is illustrated in Figure 2-6.

The logical sequence involved in constructing a Physical Address (PA) is as follows:

1. Select a set of Page Address Registers depending on the space being referenced.

2. The Active Page Field (APF) of the Virtual Address is used to select a Page Address Register (PAR0-PAR7).

3. The Page Address Field (PAF) of the selected Page Address Register (PAR) contains the starting address of the currently active page as a block number in physical memory.

4. The Block Number (BN) from the Virtual Address (VA) is added to the Page Address Field (PAF) to yield the number of the block in physical memory (PBN-Physical Block Number) which will contain the Physical Address (PA) being constructed.

5. The Displacement in Block (DIB) from the Displacement Field (DF) of the Virtual Address (VA) is joined to the Physical Block Number (PBN) to yield a true 22-bit PDP-11 Physical Address (PA).

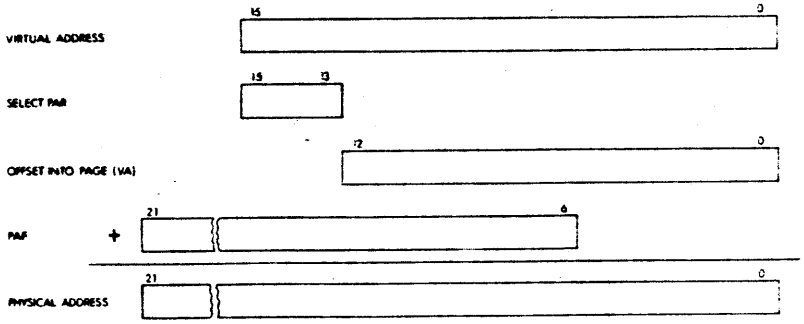


Figure 2-6 Construction of a Physical Address

Management Registers

The PDP-11/68 Memory Management Unit implements three sets of 32 sixteen bit registers. One set of registers is used in Kernel mode, another in Supervisor, and the other in User mode. The choice of which set is to be used is determined by the current CPU mode contained in the Processor Status word. Each set is subdivided into two groups of 16 registers. One group is used for references to Instruction (I) Space, and one to Data (D) Space. The I Space group is used for all instruction fetches, index words, absolute addresses and immediate operands. The D Space group is used for all other references, providing it has not been disabled by Memory Managements Register #3. Each group is further subdivided into two parts of 8 registers. One part is the Page Address Register (PAR) whose function has been described in previous paragraphs. The other part is the Page Descriptor Register (PDR). PARs and PDRs are always selected in pairs by the top three bits of the virtual address. A PAR/PDR pair contain all the information needed to describe and locate a currently active memory page.

The various Memory Management Registers are located in the uppermost 4K of PDP-11 physical address space along with the UNIBUS I/O device registers.

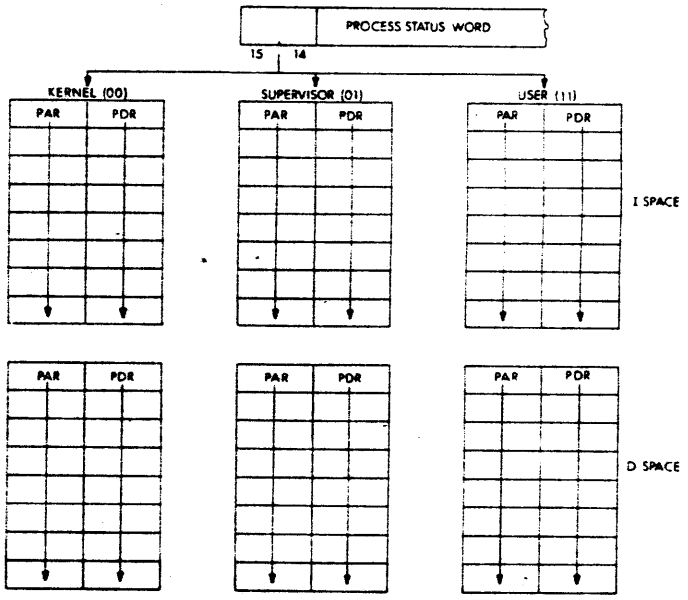


Figure 2-7 Active Page Registers

Page Address Registers (PAR)

The Page Address Register (PAR) contains the Page Address Field (PAF), 16-bit field, which specifies the starting address of the page as a block number in physical memory.

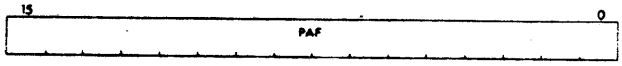


Figure 2-8 Page Address Register

The Page Address Register (PAR) which contains the Page Address Field (PAF) may be alternatively thought of as a relocation register containing a relocation constant, or as a base register containing a base address. Either interpretation indicates the basic importance of the Page Address Register (PAR) as a relocation tool.

Page Descriptor Register

The Page Descriptor Register (PDR) contains information relative to page expansion, page length, and access control.

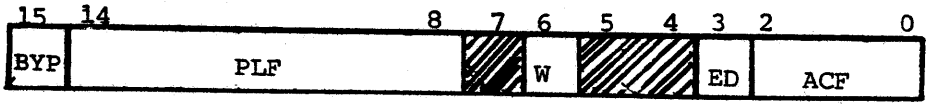


Figure 2-9 Page Description Register

Access Control Field (ACF)

This three-bit field, occupying bits 2-0 of the Page Descriptor Register (PDR) contains the access rights to this particular page. ACF <Ø> is treated as a don't care condition. The access codes or "keys" specify the manner in which a page may be accessed and whether or not a given access should result in an abort of the current operation. A memory reference which causes an abort is not completed. Aborts are used to catch "missing page faults", prevent illegal access, etc.

In the context of access control the term "write" is used to indicate the action of any instruction which modifies the contents of any addressable word. "Write" is synonymous with what is usually called a "store" or "modify" in many computer systems.

The modes of access control are as follows:

ACF	Key	Mode	
00X	0	non-resident	abort all accesses
01X	2	read only	abort on write attempt
10X	4	(unused)	abort all accesses
11X	6	read/write	no system abort action

It should be noted that the use of I Space provides the user with a further form of protection, execute only.

Access Information Bits

W Bit (bit 6) -- This bit indicates whether or not this page has been modified (i.e., written into) since either the PAR or PDR was loaded. (W = 1 is Affirmative). The W Bit is useful in applications which involve disk swapping and memory overlays. It is used to determine which pages have been modified and hence must be saved in their new form and which pages have not been modified and can be simply overlaid.

Expansion Direction (ED)

Bit 03 of the Page Description Register (PDR) specifies in which direction the page expands. If ED = 0 the page expands upwards from Block Number 0 to include blocks with higher addresses; if ED = 1, the page expands downwards from Block Number 127 to include blocks with lower addresses. Upward expansion is usually used for program space while downward expansion is used for stack space.

Page Length Field (PLF)

This seven-bit field, occupying bits 14-8 of the Page Descriptor Register (PDR), specifies the block number, which defines the boundary of that page. The block number of the Virtual Address is compared against the Page Length Field to detect Length Errors. An error occurs when expanding upwards if the block number is greater than the Page Length Field, and when expanding downwards if the block number is less than the Page Length Field.

Bypass Cache (BYP) (PDR <15>)

When the BYP bit is set in a PDR, and relocation is enabled, any CPU reference to the Virtual page mapped by that PAR/PDR pair will go directly to main memory. If read or write hits occur, the contents of that location in cache will be invalidated. Read or write misses will not disturb the contents of the cache.

When relocation is disabled, this bit will have no effect on the cache.

This read/write bit is set and cleared like all other bits in the PDR.

Reserved Bits

Bits 7, 5, & 4 are spare and are always read as 0, and should never be written. They are unused and reserved for possible future expansion.

Fault Recovery Registers

Aborts and traps generated by the Memory Management hardware are vectored through Kernel virtual location 250, Memory Management Registers #0, #1, #2, and #3 are used in order to determine why the abort occurred, and allow for easy program restarting. Note that an abort to a location which is itself an invalid address will cause another abort. Thus the Kernel program must insure that Kernel Virtual Address 250 is mapped into a valid address, otherwise a loop will occur which will require console intervention.

Memory Management Register #0 (MMR0) (status and error indicators)

MMR0 contains error flags, the page number whose reference caused the abort, and various other status flags. The register is organized as shown in Figure 2-10.

Setting bit 0 of this register enables address relocation and error detection. This means that the bits in MMR0 become meaningful.

Bits 15-13 are the error flags. They may be considered to be in a "priority queue" in that "flags to the right" are less significant and should be ignored. That is, a "non-resident" fault-service routine would ignore length, access control, and memory management flags. A "page length" service routine would ignore access control and memory management faults, etc.

Bits 15-13 when set (error conditions) cause Memory Management to freeze the contents of bits 1-7 and Memory Management Registers #1 and #2. This has been done to facilitate error recovery.

These bits may also be written under program control. No abort will occur, but the contents of the Memory Management Registers will be locked up as in an abort.

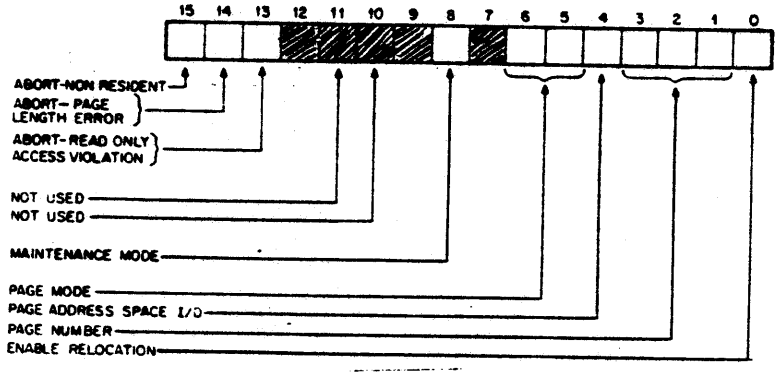


Figure 2-10 Format of Memory Management Register #0 (MMR0)

Abort -- Non-Resident, Bit 15

Bit 15 is the "Abort -- Non-Resident" bit. It is set by attempting to access a page with an Access Control Field (ACF) key equal to 0, 3, or 7. It is also set by attempting to use Memory Relocation with a processor mode of 2.

Abort -- Page Length, Bit 14

Bit 14 is the "Abort Page Length" bit. It is set by attempting to access a location in a page with a block number (Virtual Address bits, 12-6) that is outside the area authorized by the Page Length Field (PLF) of the Page Descriptor Register (PDR) for that page. Bits 14 and 15 may be set simultaneously by the same access attempt. Bit 14 is also set by attempting to use Memory Relocation with a processor mode of 2.

Abort -- Read Only, Bit 13

Bit 13 is the "Abort -- Read Only" bit. It is set by attempting to write in a "read-Only" page. "Read-Only" page has an access key of 2 (ACF = ØlX)

Bits 7, 9, 10, 11, and 12

Bits 7, 9, 10, 11, and 12 are spare and are always read as 0, and should never be written. They are unused and reserved for possible future expansion.

Maintenance/Destination Mode, Bit 8

Bit 8 specifies that only destination mode references will be relocated using Memory Management. This mode is only used for maintenance purposes.

Processor Mode, Bits 5 & 6

Bits 5 and 6 indicate the CPU MODE (Kernel/Supervisor/User) associated with the page causing the abort (Kernel = 00, Supervisor = 01, User = 11, Illegal Mode = 10). If an illegal mode is specified, bits 15 and 14 will be set.

Page Address Space, Bit 4

Bit 4 indicates the type of address space (I or D) the Unit was in when a fault occurred (0 = I Space, 1 = D Space). It is used in conjunction with bits 3-1, Page Number.

Page Number, Bits 3 to 1

Bits 3-1 contain the page number of a reference causing a Memory Management fault. Note that pages, like blocks, are numbered from 0 upwards.

Enable Relocation, Bit 0

Bit 0 is the "Enable Relocation" bit. When it is set to 1, all addresses are relocated by the unit. When bit 0 is set to 0 the Memory Management Unit is inoperative and addresses are not relocated or protected.

Memory Management Register #1 (MMR1)

MMR1 records any autoincrement/decrement of the general purpose registers, including explicit references through the PC. MMR1 is cleared at the beginning of each instruction fetch. Whenever a general purpose register is either autoincremented or autodecremented the register number and the amount (in 2s complement notation) by which the register was modified, is written into MMR1.

The information contained in MMR1 is necessary to accomplish an effective recovery from an error resulting in an abort. The low order byte is written first and it is not possible for a PDP-11 instruction to autoincrement/decrement more than two general purpose registers per instruction before an "abort-causing" reference. Register numbers are recorded "MOD 8"; thus it is up to the software to determine which set of registers (User/Supervisr/ Kernel -- General Set 0/General Set 1) was modified, by determining the CPU and Register modes as contained in the PS at the time of the abort. The 6-bit displacement on R6(SP) that can be caused by the MARK instruction cannot occur if the instruction is aborted.

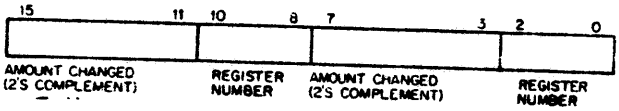


Figure 2-11 Format of Memory Management Register #1 (MMR1)

Memory Management Register #2

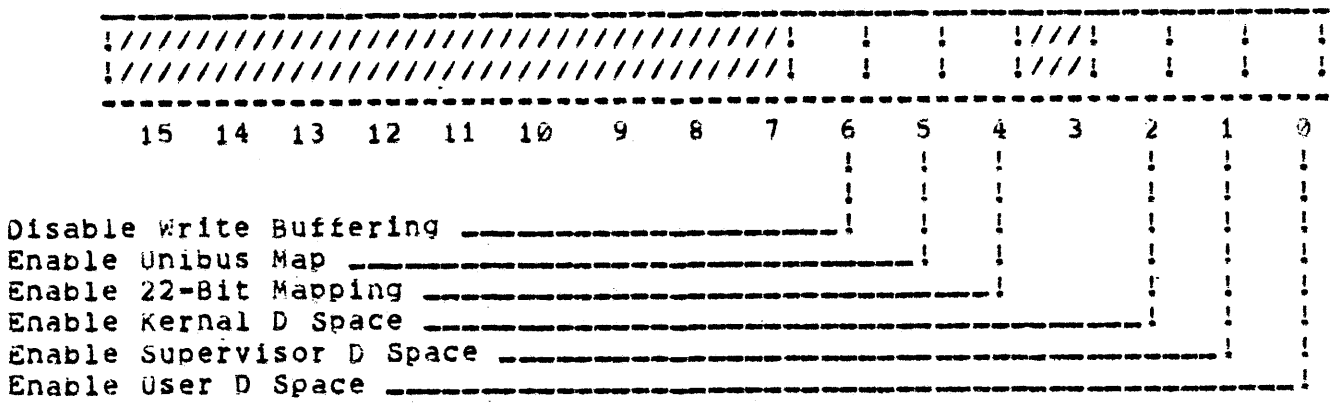
MMR2 is loaded with the 16-bit Virtual Address (VA) at the beginning of each instruction fetch. MMR2 is Read-Only; it cannot be written. MMR2 is the Virtual Address Program Counter.

Memory Management Register #3

The Memory Management Register #3 (MMR3) enables or disables the use of the D space PAR's and PDR's and 22-bit mapping and UNIBUS mapping. When D space is disabled, all references use the I space registers; when D space is enabled, both the I space and D space registers are used. Bit 0 refers to the User's Registers, Bit 1 to the Supervisor's, and Bit 2 to the Kernel's. When the appropriate bits are set D space is enabled; when clear, it is disabled. Bit 03 is read as zero and never written. It is reserved for future use. Bit 04 enables 22-bit mapping. If Memory Management is not enabled, bit 04 is ignored and 16-bit mapping is used.

If bit<4> is clear and Memory Management is enabled (MMR0=1), the processor uses 18-bit mapping. If Bit<4> is set and Memory Management is enabled, the processor uses 22-bit mapping. Bit<5> is set to enable relocation of the UNIBUS map; the bit is cleared to disable relocation. Bits<15:7> are unused. On initialization this register is cleared and only I-space is used.

Memory Management Register 3 17 772 576



FORMAT OF MEMORY MANAGEMENT REGISTER 3

Bit	State	Operation
6	0	Enables write buffering
	1	Disables write buffering
5	0	Unibus Map relocation disabled
	1	Unibus relocation enabled
4	0	Enable 18 bit mapping if Bit<0> of MMR0 = 1
	1	Enable 22 bit mapping if bit<0> of MMR0 = 1
2	1	Enable Kernal D Space
1	1	Enable Supervisior D Space
0	1	Enable User D Space

PDP 11/68 Write Buffering (MMR3<6>)

The PDP11/68 will buffer write operations and continue processing using cache data in its normal mode of operation. The Power up state will have this function enabled. When enabled, write buffering will be automatically disabled in operations performing all writes to the I/O Page (both internal and external). Flushing of the buffer will occur in all direct or indirect references to the Processor Status Word (Traps, RTI, RTT, interrupts; MOV X,PSW) such that all memory errors will be associated with the process that initiated the write.

Instruction Back-up/Restart Recovery

The process of "backing-up" and restarting a partially completed instruction involves:

1. Performing the appropriate memory management tasks to alleviate the cause of the abort (e.g., loading a missing page, etc.).

2. Restoring the general purpose registers indicated in MMR1 to their original contents at the start of the instruction by subtracting the "modify value" specified in MMR1.

3. Restoring the PC to the "abort-time" PC by loading R7 with the contents of MMR2, which contains the value of the Virtual PC at the time the "abort-generating" instruction was fetched.

Note that this back-up/restart procedure assumes that the general purpose register used in the program segment will not be used by the abort recovery routine. For back-up restart procedures for Commercial Instructions (CIS) see DEC STD 168.

Clearing Status Registers Following Trap/Abort

At the end of a fault service routine bits 15-12 of MMR0 must be cleared (set to 0) to resume error checking. On the next memory reference following the clearing of these bits, the various Registers will resume monitoring the status of the addressing operations. MMR2 will be loaded with the next instruction address, MMR1 will store register change information and MMR0 will log Memory Management Status information.

Multiple Faults

Once an abort has occurred, any subsequent errors that occur will not affect the state of the machine. The information saved in MMR0 through MMR2 will always refer to the first abort that it detected.

In the case that an abort occurs after a trap, but in the same instruction, only one stack operation will occur; and the PC and PS at the time of the abort will be saved.

REGISTER	ADDRESS
Memory Mgt Register #0(MMR0)	17 777572
Memory Mgt Register #1(MMR1)	17 777574
Memory Mgt Register #2(MMR2)	17 777576
Memory Mgt Register #3(MMR3)	17 772516
User I Space Descriptor Register (UISDR0)	17 777600
.	.
.	.
.	.
User I Space Descriptor Register (UISDR7)	17 777616
User D Space Descriptor Register (UDSDR0)	17 777620
.	.
.	.
.	.
User D Space Descriptor Register (UDSDR7)	17 777636
User I Space Address Register (UISAR0)	17 777640
.	.
.	.
.	.
User I Space Address Register (UISAR7)	17 777656
User D Space Address Register (UDSAR0)	17 777660
.	.
.	.
.	.
User D Space Address Register (UDSAR7)	17 777676
Supervisor I Space Descriptor Register (SISDR0)	17 772200
.	.
.	.
.	.
Supervisor I Space Descriptor Register (SISDR7)	17 772216
Supervisor D Space Descriptor Register (SDSDR0)	17 772226
.	.
.	.
.	.
Supervisor D Space Descriptor Register (SDSDR7)	17 772236
Supervisor I Space Address Register (SISAR0)	17 772240
.	.
.	.
.	.
Supervisor I Space Address Register (SISAR7)	17 772256
Supervisor D Space Address Register (SDSAR0)	17 772260
.	.
.	.
.	.
Supervisor D Space Address Register (SDSDR7)	17 772276
Kernel I Space Descriptor Register (KISDR0)	17 772300
.	.
.	.
.	.
Kernel I Space Descriptor Register (KISDR7)	17 772316

Kernel D Space Descriptor Register (KDSDR0)	17 772320
.	.
.	.
.	.
Kernel D Space Descriptor Register (KDSDR7)	17 772336
Kernel I Space Address Register (KISAR0)	17 772340
.	.
.	.
.	.
Kernel I Space Address Register (KISAR7)	17 772356
Kernel D Space Address Register (KDSAR0)	17 772360
.	.
.	.
.	.
Kernel D Space Address Register (KDSAR7)	17 772376

2.5 Cache/Memory Operations

CPU MEMORY REFERENCE

Cache memory within the 11/68 operates synchronously with CPU memory references. Virtual address information from the CPU is applied to main cache and, in parallel, to the Memory Management unit as described above. The CPU always looks for data in cache memory first when attempting a read before accessing main memory. If the data is in cache, a hit occurs and main memory is not accessed. Four bytes of data are returned to the CPU via the 32 bit internal bus in time for latching at the end of the microstate in which the bus code was issued. On a miss, the CPU clock is suppressed and two words are fetched from backing store and placed on the internal bus. The clock is then The data and tag of a cache location are updated to correspond to the data obtained in the access to main memory (allocation). During a write into memory, if a hit occurs, both main memory and cache are updated. If a miss occurs, main memory is written, but cache is not allocated because of the prevelence of single word writes which are not compatable with the block size of two.

HIT or MISS OPERATIONS

Processor Operation	Cache	Main Memory
Read		
Hit	No Change	No Change
Miss	Allocated	No Change
Write		
Hit	Updated	Updated
Miss	No Change	Updated
DMA Operations		
Read		
Hit (not checked)	No Change	No Change
Miss (not checked)	No Change	No Change
Write		
Hit	Invalidated	Updated
Miss	No Change	Updated

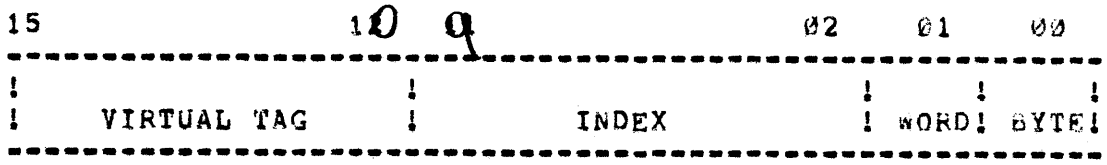
Allocated- The data and tag of the cache location is changed to correspond to the main memory location.

Updated- The data in cache or main memory is modified or revised.

Invalidated- Valid bit in the cache location is cleared to show that the data is stale and does not correspond to the data in main memory.

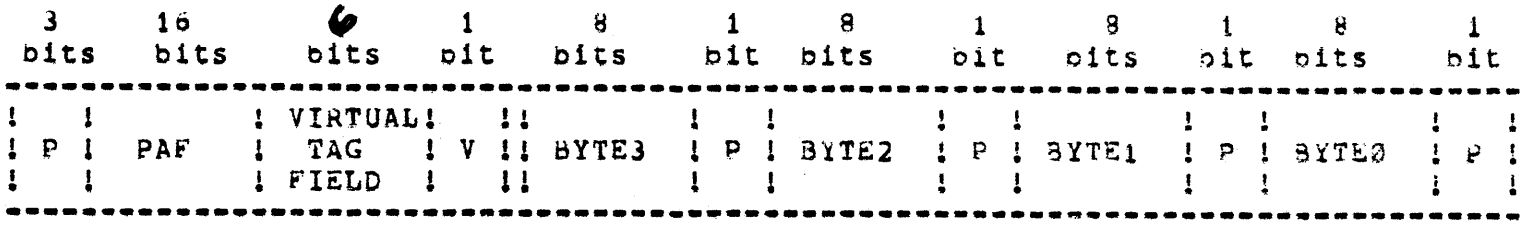
CACHE MEMORY FORMAT

The size of the cache memory is ~~4096~~ 1024 words (~~8192~~ 2048 bytes), organized as a two-way associative cache with two-word blocks. This means there are two sets in the cache; each set contains ~~1024~~ 256 blocks of data, and each block contains two PDP 11 words. Each block also has a virtual address tag field and a Page Address Field (PAF) of the Page Address Register (PAR) corresponding to the PAF accessed by address bits<15:13> of the virtual address during allocation. This information uniquely determines the physical address in main memory where the original copy of this data block resides. The data from main memory can be stored within the cache in one index position determined by its virtual address. The 10 bit index field (bits<12:2>) determine which element of the array will contain the data (but it can be either in Set 0 or Set 1).



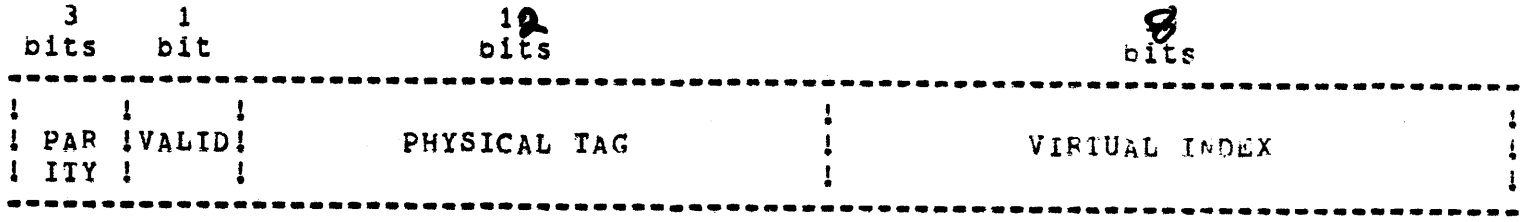
Virtually Addressed Data Cache Format

The elements of the cache must store not only the data, but also the address identification. Since the index position itself implies part of the address, only the high address field (called Tag and PAF data) must be stored. The combination of the virtual and PAF tag plus index gives the address of the two-word block in main memory. The lowest two bits in the address select the particular word in the block, and the byte (if needed). There are two places in the cache where any block of data can be stored, a particular index position in either Set 0 or Set 1. Random selection determines into which set the information is placed, overwriting the previous data. Another bit is needed within the cache to determine if the block has been loaded with data. When power is first applied, the cache data is invalid, and the valid bit for each data block is cleared. When a particular block location is updated, the associated valid bit is set to indicate good data. The following figure shows the organization for a single block of data within a set. Note that data has byte parity, and that the non-data part called Tag contains a 6 bit high order virtual address field (PAF), a 16 bit Page Address Field, a valid bit and 3 parity bits.


3 bits
16 bits
6 bits
1 bit
8 bits
1 bit
8 bits
1 bit
8 bits
1 bit
8 bits
1 bit
P
PAF
VIRTUAL TAG FIELD
V
BYTE3
P
BYTE2
P
BYTE1
P
BYTE0
P

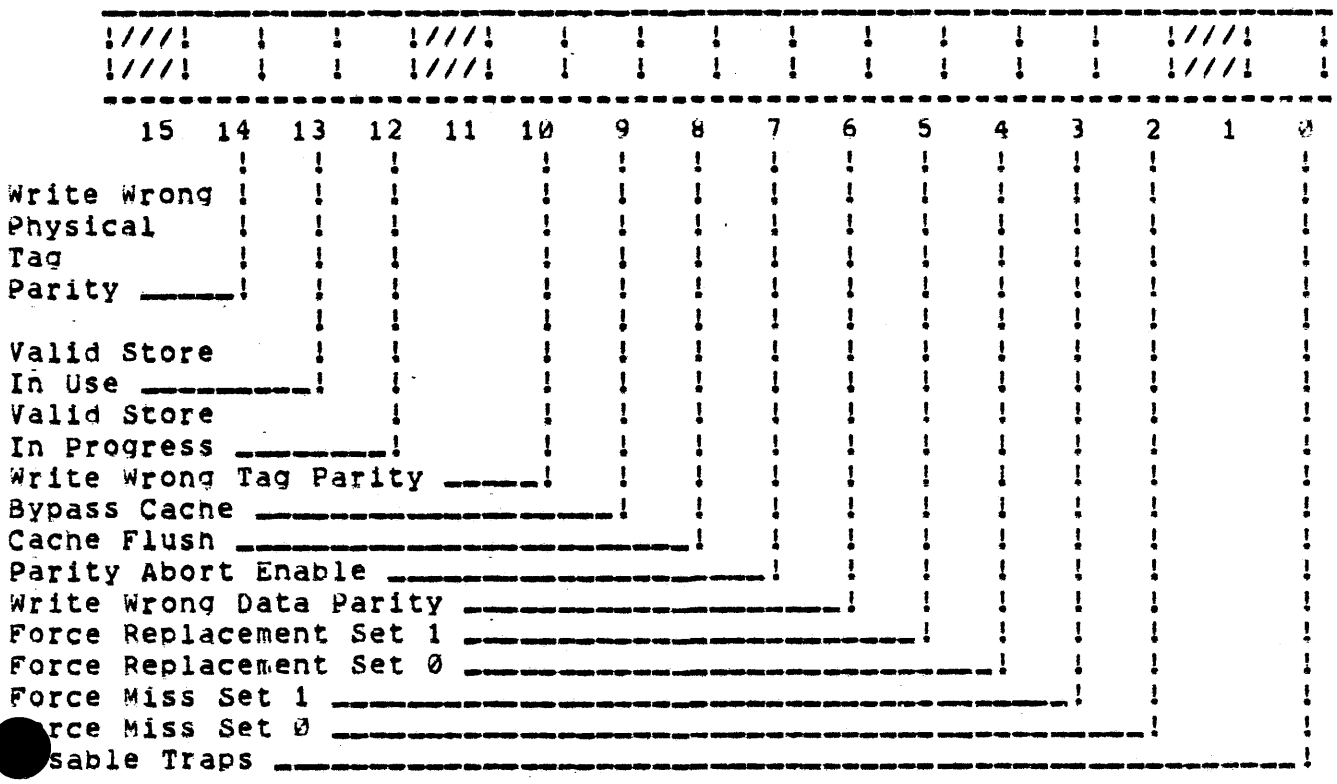
DMA OPERATIONS

Exterior DMA memory references that write into memory are monitored by the cache control logic. Physical address bits<19:2> are used as an index to access a Physical Tag Store. This physical cache tracks the main cache memory and contains the physical address tag of each location that has been allocated. In addition, the physical tag store contains the corresponding virtual index field which existed when the physical address tag was stored from the virtual address bits <19:2> which was used during allocation thereby always permitting a backward reference into the corresponding location in the virtually addressed data cache. If the tag bits of the physical address from a DMA write matches the address bits in the "physical" tag store, then the cache control will "steal" a cycle from main cache and invalidate the location. To provide the necessary cross reference between physical and virtual, the virtual index field is used to address the main data cache. If the tag bits of the physical address used in the DMA write operation do not match the address bits in the physical tag store, then no time is taken from processor main cache operations and no invalidation is necessary.



PHYSICAL TAG DATA FORMAT

Cache Control Register 17 777 746



CCR<14> Write Wrong Physical Tag Parity

This bit is read/write and when set causes tag parity bits to be written with wrong parity on CPU read misses. A parity error will thus occur on the next read miss allocation/invalidation cycle using this location in the physical tag store. A parity error will also occur in the invalidation cycle on a DMA write hit to this location in the physical tag store.

CCR<13> Valid Store In Use

This bit controls which set of valid store bits is currently being used to determine the validity of the contents of the tag store memory. It is read only and is complemented each time that the cache is flushed. When set valid bit B is is use. When clear valid bit A is in use.

CCR<12> Valid Clear In Progress

This bit is read only and is set to indicate that the cache is currently in the process of clearing a valid store bit. The clear cycle occurs on power up and when the flush cache bit is set. The hardware clear cycle will take approximately ?usec to be accomplished. While a valid store set is being cleared, the other set is in use allowing the cache to continue functioning.

CCR<10> Write Wrong Virtual Tag Parity

This bit is read/write and when set causes tag parity bits to be written with wrong parity on CPU read misses and write hits. A parity error will thus occur on the next access to that location.

CCR<09> Unconditional Cache Bypass

This bit is read/write. When set, all references to memory by the CPU will be forced to go to main memory. Read and write hits will result in invalidation of those locations in the cache and misses will not change the contents.

CCR<08> Flush Cache

This bit is write only. It will always be read as "0". Writing a "1" into this location will cause the entire cache contents to be declared invalid. Writing a "0" into this bit will have no effect.

CCR<07> Parity Error Abort

This bit is read/write and controls response of the cache to a parity error. When set a cache parity will cause the cache reference to be aborted and trap to vector 114. When cleared this bit inhibits the abort and enables an interrupt to parity error vector 114 depending on the condition of the traps enable bit CCR<00>.

CCR<06> Write Wrong Parity Data

This bit is read/write and when set causes high and low parity bytes to be written with wrong parity on all update cycles. This will cause a cache parity error to occur on the next access to that location.

CCR<05> Force Replacement Set 1

Setting this bit forces data replacement within Set 1 in the cache on a read miss by main memory.

CCR<04> Force Replacement Set 0

Setting this bit forces data replacemnet within Set 0 in the cache on a read miss by main memory.

CCR<03> Force Miss Set 1

Setting this bit inhibits hits from occurring from Set 1 forcing read references to occur from main memory.

CCR<02> Force Miss Set 0

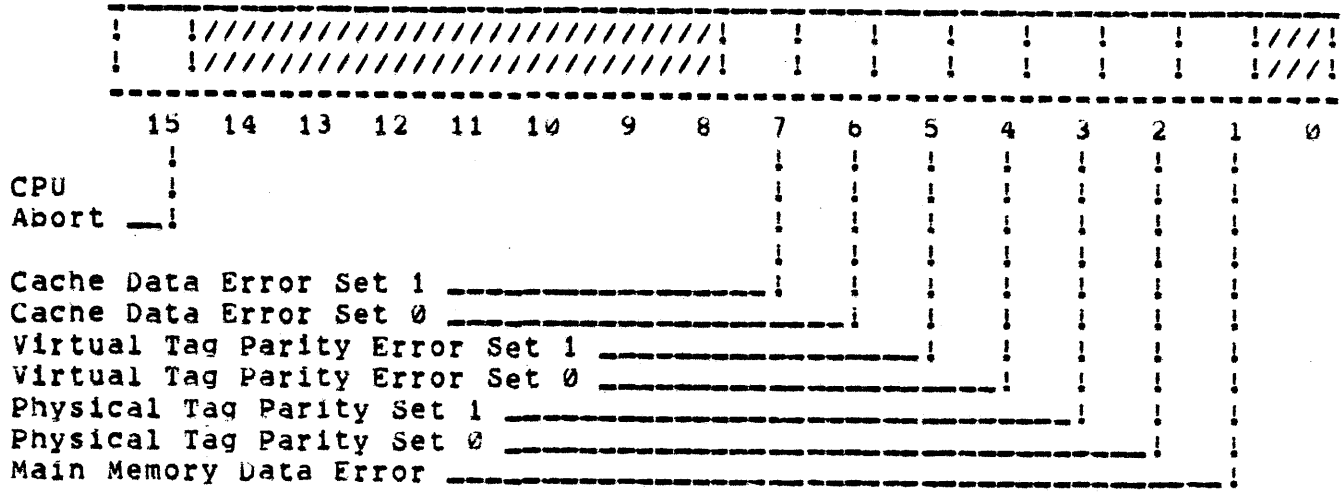
Setting this bit inhibits hits from occurring from Set 0 forcing read references to occur from main memory.

CCR<00> Disable Cache Parity Interrupt

This bit is read/write. When set this bit inhibits an interrupt from occurring on cache parity errors when cache parity abort is disabled (CCR<07>=0). All references resulting in a parity error with abort disabled will result in a force miss.

CCR<07>	CCR<00>	
0	0	Force miss and trap to 114
0	1	Force miss only
1	X	Abort and Trap to 114

Cache/Memory System Error Register 17 777 744



Bit<15> CPU Abort

Set if an error occurs which causes a processor memory reference to be aborted due to cache parity errors or main memory errors and timeouts.

Bit<07> Cache Data Error Set 1

This bit is set if a parity error is detected in the data field of Set 1 of the cache.

Bit<06> Cache Data Error Set 0

This bit is set if a parity error is detected in the data field of Set 0 of the cache.

Bit<05> Virtual Cache Tag Parity Error Set 1

This bit is set if a parity error is detected in the Tag field of the Virtual cache store of Set 1.

Bit<04> Virtual Cache Tag Parity Error Set 0

This bit is set if a parity error is detected in the Tag field of the Virtual Cache Store of Set 0.

Bit<03> Cache Physical Tag Parity Error Set 1

This bit is set if a parity error is detected in the Tag field of the Physical Cache Store of Set 1.

Bit<Ø2> Cache Physical Tag Parity Error Set Ø

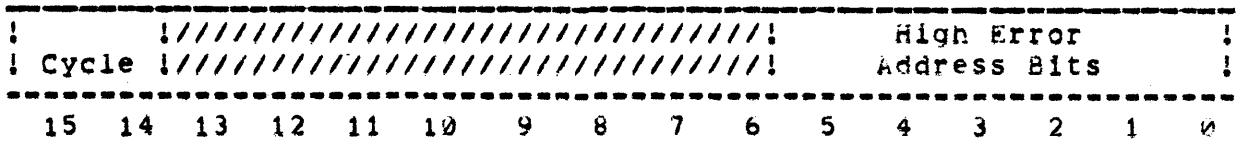
This bit is set if a parity error is detected in the TAG field of the Physical Cache Store of SET Ø.

Bit<Ø1> Main Memory Data Error

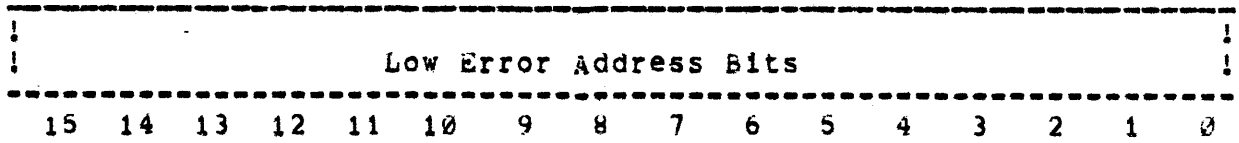
This bit is set if a non-correctable error is received from a main memory reference by the processor.

Note: Is there a need to distinguish the source of a physical tag store parity error ???? Issue to be resolved with diagnostics and software operating systems. Affects CCR<Ø1> and Cache/Memory System Error Register Bit<12>.

High Error Address Register 17 777 742



Low Error Address Register 17 777 740



The High and Low Error Address Registers log the 22-bit physical address being accessed when a memory reference error occurred. All bits are read only. The bits are undetermined after power up and are not affected by a console start or a RESET instruction. The Low Error Address Register contains the low order 16 bits of the physical address. The High Error Address Register contains the upper 6 bits of the physical address and the type of cycle being performed when the error occurred.All CPU cache/memory references that result in an error will have the physical address logged in the ERROR ADDRESS REGISTER. A CPU memory reference resulting in a CPU ABORT will cause the ERROR ADDRESS REGISTER to lock up until the condition is cleared in the Memory System Error Register Bit<15>. Bits<15:14> define the type of memory cycle performed when the error occurred.

Bit 15	Bit 14	Function
0	0	READ
0	1	READ PAUSE
1	0	WRITE
1	1	WRITE BYTE

(May expand to Bit<13> due to additional bus operations performed by the 11/68).

FLOATING POINT PROCESSOR

INTRODUCTION

The PDP-11/68 contains an integral Floating Point Instruction set supported by microcode in the base machine, and by the optional FP11 Floating Point Processor. This optional unit fits into the processor backplane and provides a high performance execution of the Floating Point Instruction set.
Both units provide significant improvement in execution over software subroutine implementation of floating point.

This chapter discusses the optional FP11 Floating Point Processor. Format and instruction information are the same for the integral floating point with the exceptions already noted The sequence of operation differs as the FP11 Floating Point Processor can operate in parallel with the base machine.

The features of the FP11 unit are:

- 17 digit accuracy
- Overlapped operation with the central processor
- High speed
- Single and double precision (32 or 64 bit) floating point modes
- Flexible addressing modes
- Six 64-bit floating point accumulators
- Error recovery aids

OPERATION

The Floating Point instruction set is an integral part of the Central Processor. It operates using similar address modes, and the same memory management facilities provided by the Memory Management Option, as the Central Processor. Floating Point Processor (FPP) instructions can reference the floating point accumulators, the Central Processor's general registers, or any location in memory.

When, in the course of a program, an FPP Instruction is fetched from memory, the FPP will execute instruction in parallel with the CPU continuing with its instruction sequence. The CPU is delayed a very short period of time during the FPP Instruction's Fetch operation, and then is free to proceed independently of the FPP. The interaction between the two processors is automatic, and a program can take full advantage of the parallel operation of the two processors by intermixing Floating Point Processor and Central Processor instructions.

Interaction between Floating Point and Central Processor instructions is automatically taken care of by the hardware. When an FPP Instruction is encountered in a program, the machine first initiates Floating Point handshaking and calculates the address of the operand. It then checks the status of the Floating Point Processor. If the FPP is "busy," the CPU will wait until it is "done" before continuing execution of the program. As an example, consider the following sequence of instructions:

```
        LDD(R3)+,AC3      ;Pick up constant operand and place it
                          in AC3
ADDLP:  LDD(R3) +, AC0    ;Load AC0 with next value in table
        MUL AC3,AC0       ;and multiply by constant in AC3
        ADDD AC0,AC1      ;and add the result into AC1
        SOB R5,ADDLP      ;check to see whether done
        STCDI AC1@R4      ;done, convert double to integer and
                          store
```

In the above example, the Floating Point Processor will execute the first three instructions. After the "ADDD" is fetched into the FPP, the CPU will execute the "SOB", calculate the effective address of the STCDI instruction, and then wait for the FPP to be "done" with the "ADDD" before continuing past the STCDI instruction.

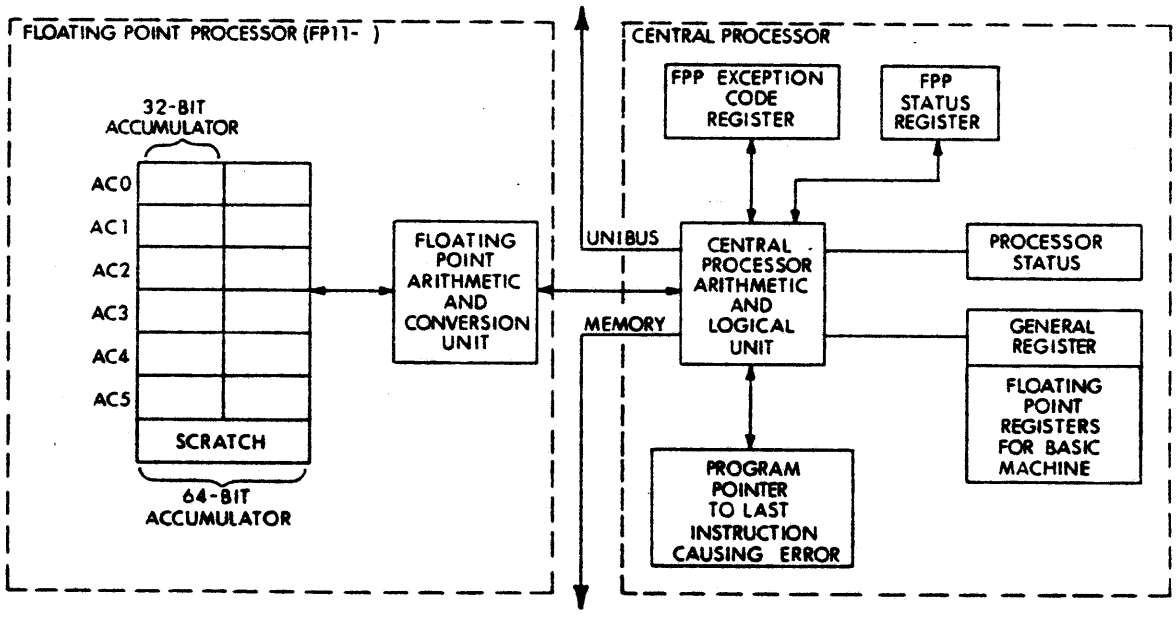
As can be seen from this example, autoincrement and autodecrement addressing automatically adds or subtracts the correct amount to the contents of the register, depending on the modes represented by the instruction.

ARCHITECTURE

The Floating Point Processor contains scratch registers and six general purpose accumulators (AC0-AC5).

Each accumulator is interpreted to be 32 or 64 bits long depending on the instruction and the status of the Floating Point Processor. For 32-bit instruction only the left-most 32 bits are used, while the remaining 32 bits remain unaffected.

The six Floating Point Accumulators are used in numeric calculations and interaccumulator data transfers; the first four (AC0-AC3) are also used for all data transfers between the FPP and the General Registers or Memory.



Floating Point Processor and Central Processor of the PDP-11/68

FLOATING POINT DATA FORMATS

Mathematically, a floating point number may be defined as having the form (2**K)*f, where K is an integer and f is a fraction. For a non-vanishing number, K and f are uniquely determined by imposing the condition $\frac{1}{2} \leq f < 1$. The fractional part, f, of the number is then said to be normalized. For the number zero, f must be assigned the value 0, and the value of K is indeterminate.

The FPP floating point data formats are derived from this mathematical representation for floating point numbers. Two types of floating point data are provided. In single precision, or Floating Mode, the word is 32 bits long. In double precision, or Double Mode, the word is 64 bits long. Sign magnitude notation is used.

Non-vanishing Floating Point Numbers

The fractional part f is assumed normalized, so that its most significant bit must be 1. This 1 is the "hidden" bit: it is not stored in the data word, but of course the hardware restores it before carrying out arithmetic operations. The Floating and Double modes reserve 23 and 55 bits, respectively, for f, which with the hidden bit, imply effective word lengths of 24 bits and 56 bits for arithmetic operations.

Eight bits are reserved for the storage of the exponent K in excess 128 (200 octal) notation (i.e. as K + 200 octal). Thus exponents from −128 to +127 could be represented by 0 to 377 (octal), or 0 to 255 (decimal). For reasons given below, a biased EXP of 0 (true exponent of −200 octal), is reserved for floating point zero. Thus exponents are restricted to the range −127 to +127 inclusive (−177 to 177 octal) or, in excess 200 (octal) notation, 1 to 377 (octal).

The remaining bit of the floating point word is the sign bit.

Floating Point Zero

Because of the hidden bit, the fractional part is not available to distinguish between zero and non-vanishing numbers whose fractional part is exactly 1/2. Therefore the FP11 reserves a biased exponent of 0 for this purpose. And any floating point number with biased exponent of 0 either traps or is treated as if it were an exact 0 in arithmetic operations. An exact zero is represented by a word, whose bits are all 0's. An arithmetic operation for which the resulting true exponent exceeds 177 (octal) is regarded as producing a floating overflow; if the true exponent is less than −177 (octal) the operation is regarded as producing a floating underflow. A biased exponent of 0 can thus arise from arithmetic operations as a special case of overflow (true exponent = 400 octal), or as a special case of underflow (true exponent = 0). (Recall that only eight bits are reserved for the biased exponent.) The fractional part of results obtained from such overflows and underflows is correct.

The Undefined Variable

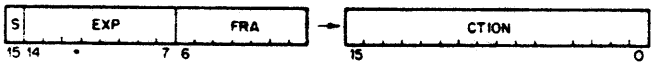
The undefined variable is defined to be any bit pattern with a sign bit of one and a biased exponent of zero. The term "undefined variable" is used, for historical reasons, to indicate that these bit patterns are not assigned a corresponding floating point arithmetic value. Note that the undefined variable is frequently referred to as "−0" elsewhere in this chapter.

A design objective of the FP11 was to assure that the undefined variable would not be stored as the result of any floating point operation in a program run with the overflow and underflow interrupts disabled. This is achieved by storing an exact zero on overflow or underflow, if the corresponding interrupt is disabled. This feature together with an ability to detect a reference to the undefined variable (implemented by the FIUV bit discussed in the next section) is intended to provide the user with a debugging aid: if the presence of −0 occurs, it did not result from a previous floating point arithmetic instruction.

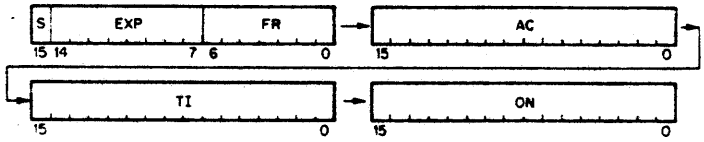
Floating Point Data

Floating point data is stored in words of memory as illustrated below.

F Format, single precision



D Format, double precision



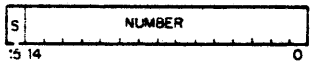
S = Sign of Fraction

EXP = Exponent in excess 200 notation, restricted to 1 to 377 octal for non-vanishing numbers.

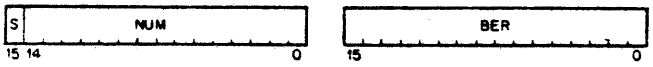
FRACTION = 23 bits in F Format, 55 bits in D Format, + one hidden bit (normalization). The binary radix point is to the left.

The FPP provides for conversion of Floating Point to Integer Format and vice-versa. The processor recognizes single precision integer (I) and double precision integer long (L) numbers, which are stored in standard two's complement form:

I Format:



L Format:



where

S = Sign of Number

NUMBER = 15 bits in I Format, 31 bits in L Format.

FLOATING POINT UNIT STATUS REGISTER (FPS Register)

This register provides (1) mode and interrupt control for the floating point unit, and (2) conditions resulting from the execution of the previous instruction.

Four bits of the FPS register control the modes of operation:

Single/Double: Floating point numbers can be either single or double precision.

Long/Short: Integer numbers can be 16 bits or 32 bits.

Chop/Round: The result of a floating point operation can be either chopped or rounded. The term "chop" is used instead of "truncate" in order to avoid confusion with truncation of series used in approximations for function subroutines.

Normal/Maintenance: a special maintenance mode is available.

The FPS register contains an error flag and four condition codes (5 bits):

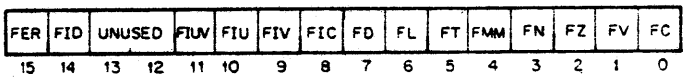
Carry, overflow, zero, and negative, which are equivalent to the Processor Status condition codes.

The floating point processor (FPP) recognizes seven "floating point exceptions":

detection of the presence of the undefined variable in memory
floating overflow
floating underflow
failure of floating to integer conversion
maintenance trap
attempt to divide by zero
illegal floating OP code

For the first five of these exceptions, bits in the FPS register are available to individually enable or disable interrupts. An interrupt on the occurrence of either of the last two exceptions can be disabled only by setting a bit which disables interrupts on all seven of the exceptions, as a group.

Of the fourteen bits described above, five are set by the FPP as part of the output of a floating point instruction: the error flag and condition codes. Any of the mode and interrupt control bits (except the FMM bit) may be set by the user; the LDFS instruction is available for this purpose. These fourteen bits are stored in the FPS register as follows:



BIT	NAME	DESCRIPTION
15	Floating Error (FER)	The FER bit is set by the FPP if 1. division by zero occurs 2. illegal OP code occurs 3. any one of the remaining occurs and the corresponding interrupt is enabled. Note that the above action is independent of whether the FID bit (next item) is set or clear. Note also that the FPP never resets the FER bit. Once the FER bit is set by the FPP, it can be cleared only by an LDFPS instruction (or by the RESET instruction described in Section 4.7). This means that the FER bit is up to date only if the most recent floating point instruction produced a floating point exception.
14	Interrupt Disable (FID)	If the FID bit is set, all floating point interrupts are disabled. Note that if an individual interrupt is simultaneously enabled, only the interrupt is inhibited; all other actions associated with the individual interrupt enabled take place.

NOTES

1. The FID bit is primarily a maintenance feature. It should normally be clear. In particular, it must be clear if one wishes to assure that storage of −0 by the FP11 is always accompanied by an interrupt.
2. Through the rest of this chapter, it is assumed that the FID bit is clear in all discussions involving overflow, underflow, occurrence of −0, and integer conversion errors.

13	Not Used	
12	Not used	
11	Interrupt on Undefined Variable (FIUV)	An interrupt occurs if FIUV is set and a −0 is obtained from memory as an operand of ADD, SUB, MUL, DIV, CMP, MOD, NEG, ABS, TST or any LOAD instruction. The interrupt occurs before execution on the FP11 except on NEG and ABS for which

BIT	NAME	DESCRIPTION
		it occurs after execution. When FIUV is reset, −0 can be loaded and used in any FPP operation. Note that the interrupt is not activated by the presence of −0 in an AC operand of an arithmetic instruction: in particular, trap on −0 never occurs in Mode 0. The FP11 will not store a result of −0 without the simultaneous occurrence of an interrupt
10	Interrupt on Underflow (FIU)	When the FIU bit is set, Floating Underflow will cause an interrupt. The fractional part of the result of the operation causing the interrupt will be correct. The biased exponent will be too large by 400 (octal), except for the special case of 0, which is correct. An exception is discussed in the detailed description of the LDEXP instruction. If the FIU bit is reset and if underflow occurs, no interrupt occurs and the result is set to exact 0.
9	Interrupt on Overflow (FIV)	When the FIV bit is set, Floating Overflow will cause an interrupt. The fractional part of the result of the operation causing the overflow will be correct. The biased exponent will be too small by 400 (octal). If the FIV bit is reset, and overflow occurs, there is no interrupt. The FP11 returns exact 0. Special cases of overflow are discussed in the detailed descriptions of the MOD and LDEXP instructions.
8	Interrupt on Integer Conversion Error (FIC)	When the FIC bit is set, and a conversion to integer instruction fails, an interrupt will occur. If

BIT	NAME	DESCRIPTION
		the interrupt occurs, the destination is set to 0, and all other registers are left untouched. If the FIC bit is reset, the result of the operation will be the same as detailed above, but no interrupt will occurr. The conversion instruction fails if it generates an integer with more bits than can fit in the short or long integer word specified by the FL bit (see 6 below).
7	Floating Double Precision Mode (FD)	Determines the precision that is used for floating point calculations. When set, double precision is assumed; when reset, single precision is used.
6	Floating Long Integer Mode (FL)	Active in conversion between integer and floating point format. When set, the integer format assumed is double precision two's complement (i.e. 32 bits). When reset, the integer format is assumed to be single precision two's complement (i.e. 16 bits).
5	Floating Chop Mode (FT)	When bit FT is set, the result of any arithmetic operation is chopped (or truncated). When reset, the result is rounded. See Section 10.8 for a discussion of the chopping and rounding operations.
4	Floating Maintenance Mode (FMM)	This code is a maintenance feature. Refer to the Maintenance Manual for the details of its operation. The FMM bit can be set only in Kernel Mode.
3	Floating Negative (FN)	FN is set if the result of the last operation was negative, otherwise it is reset.
2	Floating Zero (FZ)	FZ is set if the result of the last operation was zero; otherwise it is reset.
1	Floating Overflow (FV)	FV is set if the last operation resulted in an exponent overflow; otherwise it is reset.
0	Floating Carry (FC)	FC is set if the last operation resulted in a carry of the most significant bit. This can only occur in floating or double to integer conversions.

FLOATING EXCEPTION CODE AND ADDRESS REGISTER

One interrupt vector is assigned to take care of all floating point exceptions (location 244). The seven possible errors are coded in the four bit FEC (Floating Exception Code) register as follows:

2	Floating OP code error
4	Floating divide by zero
6	Floating (or double) to integer conversion error
8	Floating overflow
10	Floating underflow
12	Floating undefined variable
14	Maintenance trap

The address of the instruction producing the exception is stored in the FEA (Floating Exception Address) register.

The FEC and FEA registers are updated only when one of the following occurs:

1. divide by zero
2. illegal OP code
3. any of the other five exceptions with the corresponding interrupt is enabled.

NOTE

1. If one of the last five exceptions occurs with the corresponding interrupt disabled, the FEC and FEA are not updated.
2. Inhibition of interrupts by the FID bit does not inhibit updating of the FEC and FEA, if an exception occurs.
3. The FEC and FEA do not get updated if no exception occurs. This means that the STST (store status) instruction will return current information only if the most recent floating point instruction produced an exception.
4. Unlike the FPS register, no instructions are provided for storage into the FEC and FEA registers.

FLOATING POINT PROCESSOR INSTRUCTION ADDRESSING

Floating Point Processor instructions use the same type of addressing as the Central Processor instructions. A source or destination operand is specified by designating one of eight addressing modes and one of eight central processor general registers to be used in the specified mode. The modes of addressing are the same as those of the central processor except for mode 0. In mode 0 the operand is located in the designated Floating Point Processor Accumulator, rather than in a Central processor general register. The modes of addressing:

0 = Direct Accumulator
1 = Deferred
2 = Auto-increment
3 = Auto-increment deferred
4 = Auto-decrement
5 = Auto-decrement deferred
6 = Indexed
7 = Indexed deferred

Autoincrement and autodecrement operate on increments and decrements of 4 for F Format and 10, for D Format.

In mode 0, the user can make use of all six FPP accumulators (AC0—AC5) as his source or destination. In all other modes, which involve transfer of data from memory or the general register, the user is restricted to the first four FPP accumulators (AC0—AC3).

In immediate addressing (Mode 2, R7) only 16 bits are loaded or stored.

ACCURACY

General comments on the accuracy of the FPP are presented here. The descriptions of the individual instructions include the accuracy at which they operate. An instruction or operation is regarded as "exact" if the result is identical to an infinite precision calculation involving the same operands. The a priori accuracy of the operands is thus ignored. All arithmetic instructions treat an operand whose biased exponent is 0 as an exact 0 (unless FIUV is enabled and the operand is −0, in which case an interrupt occurs). For all arithmetic operations, except DIV, a zero operand implies that the instruction is exact. The same statement holds for DIV if the zero operand is the dividend. But if it is the divisor, division is undefined and an interrupt occurs.

For non-vanishing floating point operands, the fractional part is binary normalized. It contains 24 bits or 56 bits for Floating Mode and Double Mode, respectively. The internal hardware registers contain 60 bits for processing the fractional parts of the operands, of which the high order bit is reserved for arithmetic overflow. Therefore there are, internally, 35 guard bits for Floating Mode and 3 guard bits for Double Mode arithmetic operations. For ADD, SUB, MUL, and DIV, two guard bits are necessary and sufficient to guarantee return of a chopped or rounded result identical to the corresponding infinite precision operation chopped or rounded to the specified word length. Thus, with two guard bits, a chopped result has an error bound of one least significant bit (LSB); a rounded result has an error bound of 1/2 LSB. (For a radix other than 2, replace "bit" with "digit" in the two preceding sentences to get the corresponding statements on accuracy.) These error bounds are realized by the FP11 for most instructions. For the addition of operands of opposite sign or for the subtraction of operands of the same sign in rounded double precision, the error bound is 3/4 LSB, which is slightly larger than the 1/2 LSB error bound for all other rounded operations.

In the rest of this chapter an arithmetic result is called exact if no non-vanishing bits would be lost by chopping. The first bit lost in chopping is referred to as the "rounding"bit. The value of a rounded result is related to the chopped result as follows:

1. if the rounding bit is one, the rounded result is the chopped result incremented by an LSB (least significant bit).
2. if the rounding bit is zero, the rounded and chopped results are identical.

It follows that

1. If the result is exact
   rounded value = chopped value = exact value
2. If the result is not exact, its magnitude
   (a) is always decreased by chopping
   (b) is decreased by rounding if the rounding bit is zero
   (c) is increased by rounding if the rounding bit is one.

Occurrence of floating point overflow and underflow is an error condition: the result of the calculation cannot be correctly stored because the exponent is too big to fit into the 8 bits reserved for it. However, the internal hardware has produced the correct answer. For the case of underflow replacement of the correct answer by zero is a reasonable resolution of the problem for many applications. This is done on the FP11 if the underflow interrupt is disabled. The error incurred by this action is an absolute rather than a relative error; it is bounded (in absolute value) by 2**(−128). There is no such simple resolution for the case of overflow. The action taken, if the overflow interrupt is disabled, is described under FIV (bit 9)

The FIV and FIU bits (of the floating point status word) provide the user with an opportunity to implement his own fix up of an overflow or underflow condition. If such a condition occurs and the corresponding interrupt is enabled, the hardware stores the fractional part and the low eight bits of the biased exponent. The interrupt will take place and the user can identify the cause by examination of the FV (floating overflow) bit or the FEC (floating exception) register. The reader can readily verify that (for the standard arithmetic operations ADD, SUB, MUL, and DIV) the biased exponent returned by the hardware bears the following relation to the correct exponent generated by the hardware:

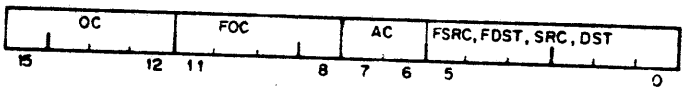
1. on overflow: it is too small by 400 octal
2. on underflow: if the biased exponent is 0 it is correct. If it is not 0, it is too large by 400 octal.

Thus, with the interrupt enabled, enough information is available to determine the correct answer. The user may, for example, rescale his variables (via STEXP and LDEXP) to continue his calculation. Note that the accuracy of the fractional part is unaffected by the occurrence of underflow or overflow.

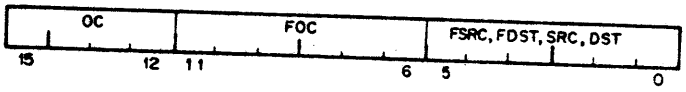
FLOATING POINT INSTRUCTIONS

Each instruction that references a floating point number can operate on either floating or double precision numbers depending on the state of the FD mode bit. Similarly, there is a mode bit FL that determines whether a 32-bit integer (FL = 1) or a 16-bit integer (FL = 0)is used in conversion between integer and floating point representation. FSRC and FDST use floating point addressing modes; SRC and DST use CPU addressing Modes.

Floating Point Instruction Format
Double Operand Adressing



Single Operand Addressing

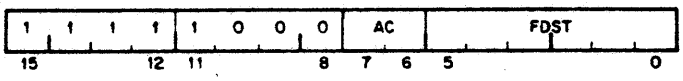


OC = Op Code = 17
FOC = Floating Op Code
AC = Accumulator
FSRC, FDST use FPP Address Modes
SRC, DST use CPU Address Modes

General Definitions:
XL = largest fraction that can be represented:
1−2**(−24), FD = 0; single precision
1−2**(−56), FD = 1); double precision
XLL = smallest number that is not identically zero = 2**(−128) −2**(−127))*(1/2)
XUL = largest number that can be represented = 2**(127)*XL
JL = largest integer that can be represented:
2**(15)−1 if FL = 0 2**(31)−1 if FL = 1
ABS (address) = absolute value of (address)
EXP (address) = biased exponent of (address)
.LT. = "less than"
.LE. = "less than or equal"
.GT. = "greater than"
.GE. = "greater than or equal"
LSB = least significant bit

STF
STD

Store Floating/Double 174ACFDST



Operation: FDST ← (AC)

Condition Codes:
FC ← FC
FV ← FV
FZ ← FZ
FN ← FN

Description: Store Single or Double Precision Number from Accumulator.

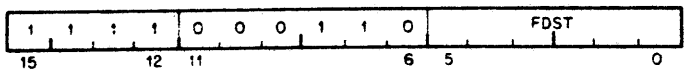
Interrupts: These instructions do not interrupt if FIUV enabled, because the −0, if present, is in AC, not in memory. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit storage of a −0 in memory from AC. There are two conditions in which minus 0 can be stored in AC of the FP11-C or FP11 One occurs when underflow or overflow is present and the corresponding interrupt is enabled. A second occurs when an LDI or LDF instruction is executed and the FIUV bit is disabled.

ABSF
ABSD

Make Absolute Floating/Double 1706FDST



Operation: If (FDST) < 0, FDST ← −(FDST).
If EXP(FDST) = 0, FDST ← exact 0.
For all other cases, FDST ← (FDST).

Condition Codes: FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 0

Description: Set the contents of FDST to its absolute value.

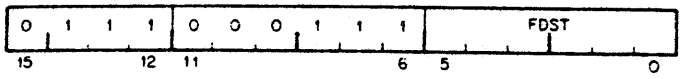
Interrupts: If FIUV is set. Trap on −0 occurs after execution.

Accuracy: These instructions are exact.

Special Comment: If a minus 0 is present in memory and the FIUV bit is enabled, then the FP11 and integral floating point unit store exact 0 in memory (zero exponent, zero fraction, and positive sign). The condition code reflects an exact 0 (FZ ← 1).

NEGF
NEGD

Negate Floating/Double 1707FDST



Operation: FDST ← −(FDST) if EXP(FDST) ≠ 0, else FDST ← exact 0.

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If EXP(FDST) = 0, else FZ ← 0.
FN ← 1 If (FDST) < 0, else FN ← 0.

Description: Negate single or double Precision number, store result in same location. (FDST)

Interrupts: If FIUV is enabled
Trap on −0 occurs after execution.

Neither overflow nor underflow can occur.

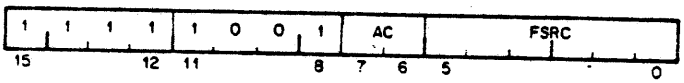
Accuracy: These instructions are exact.

Special Comment: If a minus 0 is present in memory and the FIUV bit is enabled, then the FP11 and the integral floating point unit store exact 0 in memory (zero exponent, zero fraction, and positive sign). The condition code reflects an exact 0 (FZ ← 1).

DIVF
DIVD

Divide Floating/Double

174(AC + 4)FSRC



Operation:

If EXP(FSRC) = 0, AC ← (AC): instruction is aborted.

If EXP(AC) = 0, AC ← exact 0.

For all other cases, let QUOT = (AC)/(FSRC):

If underflow occurs and FIU is not enabled AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all remaining cases AC ← QUOT.

Condition Codes:

FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If either operand has a biased exponent of 0, it is treated as an exact 0. For FSRC this would imply division by zero; in this case the instruction is aborted, the FEC register is set to 4 and an interrupt occurs. Otherwise the quotient is developed to single or double precision with enough guard bits for correct rounding. The quotient is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for:
Overflow with interrupt disabled.

Underflow with interrupt disabled.

For these exceptional cases an exact 0 is stored in accumulator.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If EXP(FSRC) = 0 interrupt traps on attempt to divide by 0.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy:

Errors due to overflow, underflow and division by 0 are described above. If none of these occurs, the error in the quotient will be bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

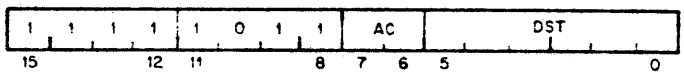
Special Comment:

The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled.

STCFI
STCFL
STCDI
STCDL

Store and Convert from Floating or Double to Integer or Long Integer

175(AC + 4)DST



Operation:

DST ← C_{ij} (AC) if $-JL-1 < C_{ij}$ (AC) $< JL+1$, else DST ← 0, where C_{ij} specifies conversion from floating mode x to integer mode j:

j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

JL is the largest integer:

2**15 − 1 for FL = 0
2**31 − 1 for FL = 1

Condition Codes:

C ← FC ← 0 if $-JL-1 < C_{ij}$ (AC) $< JL+1$, else FC ← 1.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

Description:

Conversion is performed from a floating point representation of the data in the accumulator to an integer representation.

If the conversion is to a 32-bit word (L mode) and an address mode of 0, or immediate adressing mode, is specified, only the most significant 16 bits are stored in the destination register.

If the operation is out of the integer range selected by FL, FC is set to 1 and the contents of the DST are set to 0.

Numbers to be converted are always chopped (rather than rounded) before conversion. This is true even when the Chop Mode bit, FT is cleared in the Floating Point Status Register.

Interrupts:

These instructions do not interrupt if FIUV is enabled, because the −0, if present, is in AC, not in memory.

If FIC enabled, trap on conversion failure will occur.

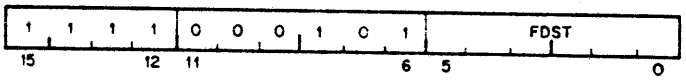
Accuracy:

These instructions store the integer part of the floating point operand, which may not be the integer most closely approximating the operand. They are exact if the integer part is within the range implied by FL.

TSTF
TSTD

Test Floating/Double

1705FDST



Operation: FDST ← (FDST)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 1 if (FDST) < 0, else FN ← 0.

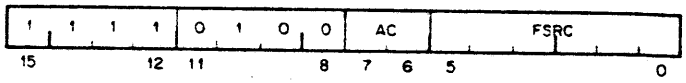
Description: Set the Floating Point Processor's Condition Codes according to the contents of FDST.

Interrupts: If FIUV is set, trap on −0 occurs after execution
Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

ADDF
ADDD

Add Floating/Double 172ACFSRC



Operation:

Let SUM = (AC) + (FSRC):

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all other cases, AC ← SUM.

Condition Codes:

FC ← 0.
FV ← 1 If overflow occurs, else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

Add the contents of FSRC to the contents of AC. The addition is carried out in single or double precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for overflow with interrupt disabled and underflow with interrupt disabled.

For these exceptional cases an exact 0 is stored in AC.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy:

Errors due to overflow and underflow are described above. If neither occurs, then: For oppositely signed operands with exponent differences of 0 or 1, the answer returned is exact if a loss of significance of one or more bits occurs. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

1 LSB in chopping mode with either single or double precision.

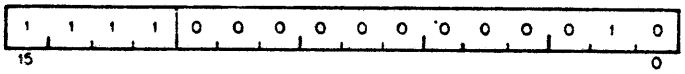
3/4 LSB in rounding mode with double precision. For an ADD instruction specifying double precision with rounding, the accuracy of the integral floating point unit is 1/2 LSB.

Special Comment:

The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled.

SETI

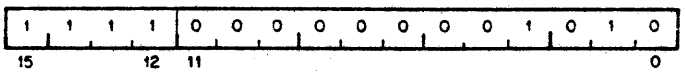
Set Integer Mode 170002



Operation: FL ← 0
Description: Set the FPP for Integer Data.

SETL

Set Long Integer Mode 170012

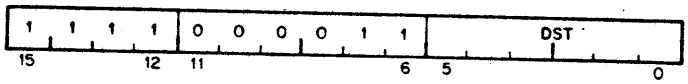


Operation: FL ← 1
Description: Set the FPP for Long Integer Data.

STST

Store FPPs Status

1703DST



Operation:

DST ← (FEC)
DST + 2 ←(FEA)

Description:

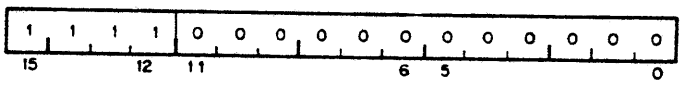
Store the FEC and then the FPP's Exception Address Pointer in DST and DST + 2.

NOTES:
1. If destination mode specifies a general register or immediate addressing, only the FEC is saved.
2. The information in these registers is current only if the most recently executed floating point instruction (refer to Section 10.6) caused a floating point exception.

CFCC

Copy Floating Condition Codes

170000



Operation:

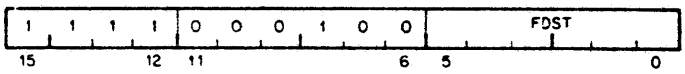
C ← FC
V ← FV
Z ← FZ
N ← FN

Description:

Copy FPP Condition Codes into the CPU's Condition Codes.

CLRF
CLRD

Clear Floating/Double 1704FDST



Operation: FDST ← exact 0.

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1
FN ← 0.

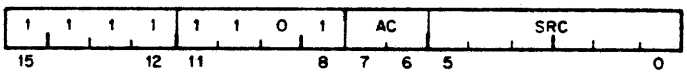
Description: Set FDST to 0. Set FZ condition code and clear other condition code bits.

Interrupts: No interrupts will occur. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

LDEXP

Load Exponent 176(AC + 4)SRC



Operation:

NOTE: 177 and 200, appearing below, are octal numbers.

If $-200 < SRC < 200$, EXP(AC) ←(SRC) + 200 and the rest of AC is unchanged on FP11C and FP11B.

If (SRC) > 177 and FIV is enabled,
EXP(AC) ← (SRC) < 6:0 >
EXP(AC) ← (SRC)< 7:0 > on FP11B.

If (SRC) > 177 and FIV is disabled,
AC ← exact 0
EXP(AC) ← (SRC + 200) < 7:0 > on FP11B.

If (SRC) < −177 and FIU is disabled,
AC ← exact 0.

If (SRC) < −177 and FIU is enabled,
EXP(AC) ← (SRC) < 6:0 >
EXP(AC) ← (SRC)+ 200) < 7:0 >

Condition Codes:

FC ← 0.
FV ← 1 if (SRC) > 177, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

Change AC so that its unbiased exponent = (SRC). That is, convert (SRC) from 2's complement to excess 200 notation, and insert in the EXP field of AC. This is a meaningful operation only if ABS(SRC).LE.177.

If SRC > 177, result is treated as overflow. If SRC < −177, result is treated as underflow. Note that the FP11C and FP11B do not treat these abnormal conditions in exactly the same way.

Interrupts:

No trap on −0 in AC occurs, even if FIUV enabled.

If SRC > 177 and FIV enabled, trap on overflow will occur.

If SRC < −177 and FIU enabled, trap on underflow will occur.

Accuracy:

Errors due to overflow and underflow are described above. If EXP(AC) = 0 and SRC ≠ −200, (AC) changes from a floating point number treated as 0 by all floating arithmetic operations to a non-zero number. This is because the insertion of the "hidden" bit in the hardware implementation of arithmetic instructions is triggered by a non-vanishing value of EXP.

For all other cases, LDEXP implements exactly the transformation of a floating point number (2**K)*f into (2**(SRC))*f where 1/2 .LE.ABS (f).LT.1.

LDF
LDD

Load Floating/Double 172(AC + 4)FSRC

Operation: AC ← (FSRC)

Condition Codes:
FC ← 0
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description: Load Single or Double Precision Number into Accumulator.

Interrupts: If FIUV is enabled, trap on −0 occurs before AC is loaded. However, the condition codes will reflect a fetch of minus 0 regardless of the FIUV bit. Neither overflow nor underflow can occur.

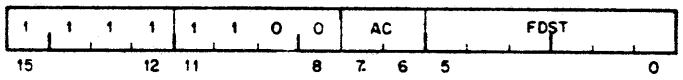
Accuracy: These instructions are exact.

Special Comment: These instructions permit use of −0 in a subsequent floating point instruction if FIUV is not enabled and (FSRC) = −0.

STCFD
STCDF

Store and convert from Floating to Double or from Double to Floating

176ACFDST



Operation:

If EXP(AC) = 0, FDST ← exact 0

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, FDST ← exact 0.

In all other cases, FDST ← C. (AC), where
C. specifies conversion from floating mode x to floating mode y;

x = F and y = D if FD = 0,
x = D and y = F if FD = 1.

Condition Codes:

FC ← 0.
FV ← 1 If conversion produces overflow else FV ←0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

If the current mode is single precision, the Accumulator is stored left justified in FDST and the lower half is cleared. If the current mode is double precision, the contents of the accumulator are converted to single precision, chopped or rounded depending on the state of FT, and stored in FDST.

Interrupts:

Trap on −0 will not occur even if FIUV is enabled because FSRC is an accumulator.

Underflow cannot occur.

Overflow cannot occur for STCFD.

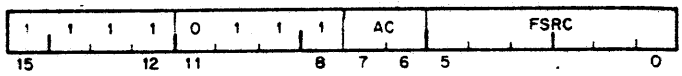
A trap occurs if FIV is enabled, and if rounding with STCDF causes overflow; FDST ← overflowed result of conversion. This result must be +0 or −0.

Accuracy:

STCFD is an exact instruction. Except for overflow, described above, STCDF incurs an error bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

CMPF
CMPD

Compare Floating/Double 173 (AC + 4) FSRC



Operation: (FSRC) — (AC)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If (FSRC) — (AC) = 0, else FZ ← 0.
FN ← 1 If (FSRC) — (AC) < 0, else FN ← 0.

Description: Compare the contents of FSRC with the accumulator. Set the appropriate floating point condition codes. FSRC and the accumulator are left unchanged.

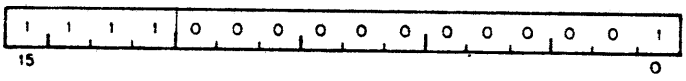
Interrupts: If FIUV is enabled, trap on —0 occurs before execution.

Accuracy: These instructions are exact.

Special Comment: An operand which has a biased exponent of zero is treated as if it were exact zero. In the case where both operands are zero, the floating point processor will store exact 0 in the AC.

SETF

Set Floating Mode 170001

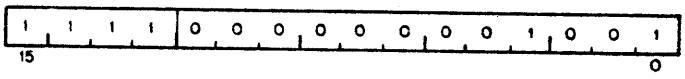


Operation: FD ← 0

Description: Set the FPP in Single Precision Mode.

SETD

Set Floating Double Mode 170011

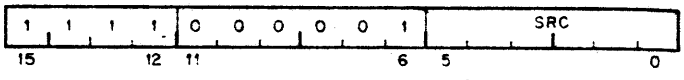


Operation: FD ← 1

Description: Set the FPP in Double Precision Mode.

LDFPS

Load FPPs Program Status 1701SRC



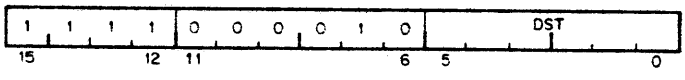
Operation: FPS ← (SRC)

Description: Load FPP's Status from SRC.

Special Comment: The user is cautioned not to use bits 13 and 12 for his own purposes, since these bits are not recoverable by the STFPS instruction.

STFPS

Store FPPs Program Status 1702DST



Operation: DST ← (FPS)

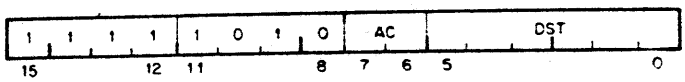
Description: Store FPP's Status in DST.

Special Comment: Bits 13 and 12 are loaded with 0. All other bits are the corresponding bits in the FPS.

STEXP

Store Exponent

175ACDST



Operation: DST ← EXP(AC)−200 octal

Condition Codes:
C ← FC ← 0.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

Description: Convert accumulator's exponent from excess 200 octal notation to 2's complement, and store result in DST.

Interrupts: This instruction will not trap on −0.

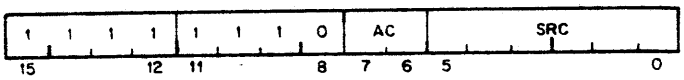
Overflow and underflow cannot occur.

Accuracy: This instruction is always exact.

LDCIF
LDCID
LDCLF
LDCLD

Load and Convert Integer or Long Integer to Floating or Double Precision

177ACSRC



Operation:

AC ← C_{ij} (SRC), where
C_{ij} specifies conversion from integer mode j to floating mode x;
j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

Condition Codes:

FC ← 0.
FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description:

Conversion is performed on the contents of SRC from a 2's complement integer with precision j to a floating point number of precision x. Note that j and x are determined by the state of the mode bits FL and FD: J = I or L, and X = F or D.

If a 32-bit Integer is specified (L mode) and (SRC) has an addressing mode of 0, or immediate addressing mode is specified, the 16 bits of the source register are left justified and the remaining 16 bits loaded with zeroes before conversion.

In the case of LDCLF the fractional part of the floating point representation is chopped or rounded to 24 bits for FT = 1 and 0 respectively.

Interrupts:

None; SRC is not floating point, so trap on −0 cannot occur.

Overflow and underflow cannot occur.

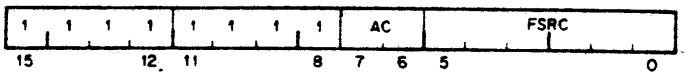
Accuracy:

LDCIF, LDCID, LDCLD are exact instructions. The error incurred by LDCLF is bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

LDCDF
LDCFD

Load and convert from Double to Floating or from Floating to Double

177(AC + 4)FSRC



Operation:

If EXP(FSRC) = 0, AC ← exact 0.

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, AC ← exact 0.

In all other cases AC ← $C_{x,y}$ (FSRC), where $C_{x,y}$ specifies conversion from floating mode x to floating mode y;

x = D, y = F if FD = 0 (single)
x = F, y = D if FD = 1. (double)

Condition Codes:

FC ← 0.
FV ← 1 if conversion produces overflow, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the current mode is Floating Mode (FD = 0) the source is assumed to be a double-precision number and is converted to single precision. If the Floating Chop bit (FT) is set, the number is chopped, otherwise the number is rounded.

If the current mode is Double Mode (FD = 1), the source is assumed to be a single-precision number, and is loaded left justified in the AC. The lower half of the AC is cleared.

Interrupts:

If FIUV is enabled, trap on −0 occurs before execution. However, the condition codes will reflect a fetch minus 0 regardless of the FIUV bit.

Overflow cannot occur for LDCFD.

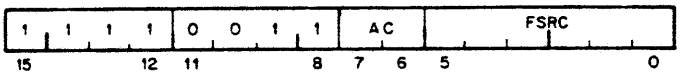
A trap occurs if FIV is enabled, and if rounding with LDCDF causes overflow; AC ← overflowed result of conversion. This result must be +0 or −0.

Underflow cannot occur.

Accuracy:

LDCFD is an exact instruction. Except for overflow, described above, LDCDF incurs an error bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

Multiply and Integerize Floating/Double 171(AC + 4)FSRC



Description and Operation

This instruction generates the product of its two floating point operands, separates the product into integer and fractional parts and then stores one or both parts as floating point numbers.

Let PROD = (AC)*(FSRC) so that in:

Floating point: ABS(PROD) = (2**K) *f

where 1/2.LE.f.LT.1 and
EXP(PROD) = (200 + K) octal

Fixed Point binary: PROD = N + g, with

N = INT(PROD) = the integer part of PROD

and

g = PROD − INT(PROD) = the fractional part of PROD with O.LE.g.LT.1

Both N and g have the same sign as PROD. They are returned as follows:

If AC is an even-numbered accumulator (0 or 2), N is stored in AC + 1 (1 or 3), and g is stored in AC.

If AC is an odd-numbered accumulator, N is not stored, and g is stored in AC.

The two statements above can be combined as follows: N is returned to ACv1 and g is returned to AC, where v means .OR.

Five special cases occur, as indicated in the following formal description with L = 24 for Floating Mode and L = 56 for Double Mode:

1. If PROD overflows and FIV enabled:

   ACv1 ← N, chopped to L bits, AC ← exact 0

Note that EXP(N) is too small by 400 (octal), and that ←0 can get stored in ACv1.

If FIV is not enabled: ACv1 ← exact 0, AC ← exact 0, and −0 will never be stored.

2. If 2**L.LE.ABS(PROD) and no overflow

   ACv1 ← N, chopped to L bits, AC ← exact 0

The sign and EXP of N are correct, but low order bit information, such as parity, is lost.

3. If 1.LE.ABS(PROD).LT.2**L

   ACv1 ← N, AC ← g

The integer part N is exact. The fractional part g is normalized, and chopped or rounded in accordance with FT. Rounding may cause a return of ±unity for the fractional part. For L = 24, the error in g is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode. For L = 56, the error in g increases from the above limits as ABS(N) increases above 3 because only 59 bits of PROD are generated:

if 2**p.LE.ABS(N).LT.2**(p + 1), with p > 2, the low order p − 2 bits of g may be in error.

4. If ABS (PROD). LT.1 and no underflow:

ACv1 ← exact 0 AC ← g

There is no error in the integer part. The error in the fractional part is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode. Rounding may cause a return of ±unity for the fractional part.

5. If PROD underflows and FIU enabled:

ACv1 ← exact 0 AC ← g

Errors are as in case 4, except that EXP(AC) will be too large by 400 octal (except if EXP = 0, it is correct). Interrupt will occur and −0 can be stored in AC.

IF FIU is not enabled, ACv1 ← exact 0 and AC ← exact 0. For this case the error in the fractional part is less than 2**(−128).

Condition Codes:

FC ← 0.
FV ← 1 if PROD overflows, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← if (AC) < 0, else FN ← 0.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC will occur before execution.

Overflow and Underflow are discussed above.

Accuracy:

Discussed above.

Applications:

1. Binary to decimal conversion of a proper fraction: the following algorithm, using MOD, will generate decimal digits D(1), D(2) . . . from left to right:

Initialize: I ← 0
X ← number to be converted;
ABS(X) < 1

```
While X ≠ 0 do
Begin PROD ← X*10;
  I ← I + 1;
  D(I) ← INT(PROD);
  X ← PROD − INT(PROD);
  END;
```

This algorithm is exact; it is case 3 in the description: the number of non-vanishing bits in the fractional part of PROD never exceeds L, and hence neither chopping nor rounding can introduce error.

2. To reduce the argument of a trigonometric function.

ARG*2/PI = N + g. The low two bits of N identify the quadrant, and g is the argument reduced to the first quadrant. The accuracy of N +g is limited to L bits because of the factor 2/PI. The accuracy of the reduced argument thus depends on the size of N.

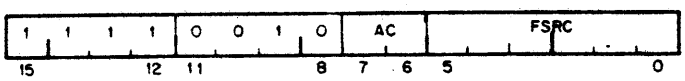
3. To evaluate the exponential function e**x, obtain

x*(log e base 2) = N + g.
Then e**x = (2**N)*(e**(g*1n 2))

The reduced argument is g*1n2 < 1 and the factor 2**N is an exact power of 2, which may be scaled in at the end via STEXP, ADD N to EXP and LDEXP. The accuracy of N + g is limited to L bits because of the factor (log e base 2). The accuracy of the reduced argument thus depends on the size of N.

MULF
MULD

Multiply Floating/Double 171ACFSRC



Operation:

Let PROD = (AC)*(FSRC)

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0.

For all other cases AC ←PROD

Condition Codes:

FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the biased exponent of either operand is zero, (AC) ← exact 0. For all other cases PROD is generated to 48 bits for Floating Mode and 59 bits for Double Mode. The product is rounded or chopped for FT = 0 and 1, respectively, and is stored in AC except for

Overflow with interrupt disabled.

Underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in accumulator.

Interrupts:

If FIUV is enabled, trap on −0 occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 otcal for underflow, except for the special case of 0, which is correct.

Accuracy:

Errors due to overflow and underflow are described above. If neither occurs, the error incurred is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

Special Comment:

The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if corresponding interrupt is enabled.

Section 5 PDP 11/68 Console

The following section describes specific details of the 11/68 console and conforms in all respects to the Mid-range Systems Console Functional Specification as described in Appendix A. The 11/68 console functions and operations are currently in the process of being defined such that the section described so far is incomplete.

FORWARD

This document will be used to define the proposed functions of the 11/68 ASCII console.There is going to be a corporate standard for the ASCII console functions and the syntax used to implement them.The 11/68 console will conform to these guidelines and specifications.

The console interfaces to an operator via a console terminal (e.g. LA36),and interfaces to the processor through a hardware interface.The console terminal and the CPU hardware will be linked by an 8085 microprocessor which will interpret the commands and execute them through the microprocessor software and CPU microcode.The console will be the functional equivilent of the past CPU's lights and switches console.The console terminal will also function as KB0: (TT0:),when not in console mode.

The console panel equivilent functions will include starting and stopping the CPU;reading and writting main memory,I/O registers,internal CPU registers and CPU execution control.

TERMINOLOGY & NOTATION

< >	Angle brackets are used to denote category names.For example,the category name <address> may be used to represent any legal address.
[ ]	Brackets surrounding part of an expression indicate that part of the expression is optional.
<sp>	Indicates one space.
<count>	Represents a numeric count.
<address>	Represents an address argument. EXP. <12345670> - numeric address <PC> - program counter (internal R7) <PS> - processor status word <SW> - switch register <SP> - stack pointer (GR6)
<data>	Represents a numeric argument.
<qualifier>	A command modifier or switch.
<prompt>	Indicates the console input prompt, '>>>'.
<cr>	Carriage return.
<lf>	Line feed.

COMMANDS

BOOT COMMAND

Syntax: B[<sp><device-name>]<cr>

Semantics: <device-name> is of the following format "DDn' where 'DD' is a two letter device mnemonic (such as DT for DEC-Tape), and 'n' is a one digit unit number.

If no <device-name> is given the console will preform the boot sequence for the default system device.

Response: The console enters the program I/O state (terminal functions as KB0:), after starting the boot sequence.

CONTINUE COMMAND

Syntax: C<cr>

Semantics: The CPU begins instruction execution at the address currently contained in the CPU program counter (PC). CPU initialization is NOT performed.The console enters program I/O mode after the continue is issued.

Response: <cr><lf>,console enters progam I/O mode.

DEPOSIT COMMAND

Syntax: D[qualifier-list]<sp><address><sp><data><cr>

Qualifiers:

/WCS	-deposit data to the writtable control store.
/B	-byte write.
/W	-word write.
/G	-internal register.
/P	-physical address
/V	-virtual address

Semantics: Deposits <data> to the <address> specified.The address type will depend upon the qualifier used with the command. Sequential deposits will write sequential locations.

Response: <cr><lf><prompt>

EXAMINE COMMAND

Syntax: E[<qualifier-list>][<sp><address>]<cr>

Qualifiers:

/B	-byte read.
/W	-word read.
/G	-internal register read.
/P	-physical address.
/V	-virtual address.

Semantics: Examines and displays the contents of the specified <address>.If no <address> is specified than the <default-address> is examined.

Response: <cr><lf><tab><address-space-identifier><sp><address><sp><data><cr><lf><prompt>

INITIALIZE COMMAND

Syntax: I<cr>

Semantics: A CPU system init is performed.

Response: <cr><lf><prompt>

HALT COMMAND

Syntax: H<cr>

Semantics: The CPU will stop instruction execution after the current instruction is completed.

Response: <contents of the CPU PC><cr><lf><prompt>

LOAD COMMAND

Syntax: L[qualifier-list]<sp><file-spec><cr>

Semantics: The load command is used to load (read),file data from the console's load device to main memory,or to the WCS. If no qualifier is used the data is transferred to memory.

Qualifiers: /S:<address>-the start switch is used to specify a starting address for the load .If no address is specified the starting address is Ø.
/WCS-used to specify the WCS is to be the target of the load.
/P or /V-force either physical or virtual main memory.

MICROSTEP COMMAND

Syntax: M[<sp><count>]<cr>

Semantics: The CPU is allowed to execute the number of microinstructions specified by the number <count>.If no count is specified the console enters SPACE-BAR-STEP mode. Each time the space bar is hit one microinstruction is executed.

Response: <cr><lf><tab>"halted at <contents of micro PC>"

NEXT COMMAND

Stntax: N[<sp><count>]<cr>

Semantics: The CPU is allowed to execute the number of MACROinstructions specified by the number <count>.If no <count> is specified the console enters SPACE-BAR-STEP mode.

Response: <cr><lf><tab>"halted at <contents of PC>"

START COMMAND

Syntax: s<sp><address><cr>

or

S/C<sp><address><cr>

Semantics: S <address> inits the CPU,loads the address into the PC and starts MACRO instruction execution.

S/C <address> deposits the <address> into the MICRO-PC and starts the system clock in free running mode.

Response: <cr><lf><prompt>

TEST COMMAND

Syntax: T<cr>

semantics: The console subsystem will execute a self check to insure its own integrity.

Qualifiers: /D-causes CPU micro-diagnostics to be run upon successful execution of the console self-test.

Response: To be determined.Some sort of message code.

X COMMAND

Syntax: Not fully defined at this time.

Semantics: This command is to be utilized by the APT (Automatic Processor Test system), and the RD (Remote Diagnosis), systems for loading binary data to either the CPU's main memory or the WCS option.Upon receiving the X command the local echo is turned off,a byte count and checksum is is received and binary data is transferred to the CPU. This allows diagnostics (both MACRO and MICRO), to be loaded for verification and trouble-shooting of the CPU.

Qualifiers: /WCS-allows the WCS option to receive binary data.

Response: Not defined.

CONTROL AND SPECIAL CHARACTERS

CONTROL C Causes the suspension of all repetitive console operations.

CONTROL O Supresses or enables console terminal output.

CONTROL P	Enters the terminal to console mode.
CONTROL U	Deletes all characters typed in a command line.
CR	Terminates a console command line.

BRIEF INTERFACE DESCRIPTION

The physical location of the console in the 11/68 is as yet undefined. It must be positioned in a way that it may get at the internal data bus of the CPU and also have a window to the UNIBUS.Incorporated into the console will be the SLU (Serial Line Unit), and LFC (Line Frequency Clock), functions now provided by the DL11-W option.The console load device (TU58), will also be able to be accessed from the UNIBUS to facilitate using the TU58 for software and possibly MICRO-CODE patching.

APPENDIX A

Midrange Systems Console Functional Specification

MIDRANGE SYSTEMS CONSOLE FUNCTIONAL SPECIFICATION

ABSTRACT

This specification defines the functions & command syntax of ASCII CONSOLE implementations for future processors developed by Digital's Mid-range Systems Development Engineering. The ASCII CONSOLE provides a programmed interface between an operator at a console terminal, and a given CPU's hardware and software. A minimum subset of console commands is defined, as are areas for individual machine-specific growth of functionality.

Revision History:

Rev.#	Description	Author	Revised Date
1	Original Compilation (Condensation of 11/780 Spec.)	K. OKIN	22-JUNE-1978
2.0	Edit after Internal Review on June 23,1978	K. OKIN	28-JUNE-1978

1.0 INTRODUCTION

This specification defines the command functions and syntax of the ASCII CONSOLE to be implemented on future DIGITAL Mid-range CPUs. The ASCII CONSOLE provides a programmed interface between an operator at the console terminal, and the CPU hardware, microcode, and ISP software. The console interfaces to an operator via a console terminal (e.g. LA36), and interfaces to the CPU via a hardware interface. The console interprets operator commands typed at the console keyboard, and performs the appropriate operations for each command by means of console software and/or CPU microcode. The console functionality is the equivalent of traditional CPU control and status functions performed by a 'lights and switches' console panel. When it is not in 'Console Mode,' the console terminal may also be used as a user terminal to communicate with an operating system.

The console command syntax will conform to the 'DEC Command Language Standard' (DCLS), or to a sub-set thereof.

2.0 CONSOLE FUNCTIONAL DESCRIPTION

This section describes the functions provided by the console. These functions are described in terms of the actions that may be initiated via console commands, however the actual command syntax used to implement this functionality is not discussed in this section.

2.1 CONSOLE PANEL EQUIVALENT FUNCTIONS

The console panel equivalent functions include starting and stopping the CPU Instruction Set Processor (ISP), reading and writing main memory, I/O registers, and processor & internal registers, and CPU execution control.

2.1.1 CPU ISP CONTROL

The functions listed below must be provided by the console to permit initiation and termination of ISP-Level program execution by the CPU.

1. CPU Initialization: The CPU can be initialized by setting certain CPU logic elements to a defined state.

2. ISP Execution Initiation: ISP-Level instruction execution can be initiated in the CPU. The CPU program counter (PC) contents will specify the memory address where instruction execution will begin.

3. ISP Execution Stop: CPU ISP-Level instruction execution may be stopped.

2.1.2 DISPLAY AND MODIFICATION OF MEMORY ELEMENTS

The console must provide for the display and modification of memory elements in the system, including elements in the main memory, I/O, General Register (R0 to Rn), Internal Processor Register (VAX only), and Machine Dependent Register addressing space. The address and data radix is architecture dependent; HEX for VAX implementations and OCTAL for PDP-11 implementations.

1. Main memory or I/O elements can be read and written as byte (8 bits (optional)), word (16 bits), longword (32 bits (VAX only)) quantities.

2. CPU General Registers are read and written as long word (32 bit VAX) or word (16 bit PDP-11) quantities. The general registers are R0 thru R13(VAX) or R0 thru R5(PDP-11), SP, and PC.

3. CPU Internal Processor Registers exist only in VAX implementations, and are read or written as long word quantities. The Internal processor Register space includes the architectural registers described in Chapter 9 of the VAX SRM, and other machine dependent registers available to an ISP level program through a MTPR of MFPR instruction.

4. CPU Machine Dependent Registers are read or written as word (PDP-11) or long word (VAX) quantities. The Machine Dependent Register space includes all the implementation specific registers which exist for a given machine. This is a defined area of incompatability; the Machine Dependent registers on one CPU will probably bear no resemblance to any other CPU.

2.1.3 CPU EXECUTION CONTROL

The execution control modes available as optional console functionality may include bootstrapping, single instruction step, and single micro-instruction step.

1. Single instruction step mode (when implemented) will allow CPU ISP-Level programs to be executed one instruction at a time. This mode causes the CPU to enter the halt state after the execution of an instruction is completed.

2. Single Micro Cycle step mode (when implemented) will cause the CPU microsequencer to stop each time a new micro-instruction is about to be executed. The processor will remain in this state

until it is told either to resume normal execution or to step another micro state.

3. Bootstrap: The CPU can be 'Bootstrapped'. A bootstrap sequence may consist of loading the CPU main memory with a specific file from the Console Sub-system load device, and initiating CPU ISP-Level instruction execution at a pre-defined address after the load. Certain implementations may not have a load device, and will merely initiate execution of an ISP ROM bootstrap program, while passing parameters to that program to specify the the exact bootstrap device.

2.2 DEFAULTS AND RADICES

The RADIX of a given console is architecturally defined and will be constant across a given family. The addressing space and data length referenced by console commands is not constant, and is selectable by command qualifiers. The following defaults will be used by the consoles upon power up:

Address Type	=	Physical
Radix	=	HEX (VAX)
Radix	=	OCTAL (PDP-11)
Data Length	=	Long Word (32 Bits VAX)
Data Length	=	Word (16 Bits PDP-11)

2.3 ABBREVIATIONS

Due to a possible lack of console program space, the consoles for this class of machine may not be able to accept the entire english spelling of a command name. The following standard will be adhered to:

> At the time of their implementation, ALL consoles MUST recognize the MINIMUM abbreviations for their implemented commands as defined in this spec. Depending upon the programming space available, the implementor may allow a given console to accept longer strings as a command (example: S, or ST, or START, for start.)
>
> However, should the console recognize only S , it should NOT accept a longer string WHICH IT CANNOT VERIFY. Should a longer string be typed as a command, the console should return an error message and not execute the command.

This restriction prevents a user from typing one command and having the console execute another, potentially destructive, command. It also preserves a common command set accross consoles, namely the shortest abbreviation possible for a given command. This implies that all consoles will process a S 100<CR> as a start command, and should some console command be added to this spec which also starts with the letter S, the minimum abbreviation for that new (hypothetical) command will consist of at least two letters.

2.4 TYPING ERRORS

Some consoles of this class will parse the input string as it is typed. It is thus may not be possible to "erase" errors by typing a rubout (or delete) character. However, should a given console not begin parsing a line until the receipt of a carriage return, it may allow rubouts to erase characters already typed. If this is allowed, when the rubout is typed the console must echo the character being deleted (after printing a backslash(\) upon receipt of the first rubout), and the console must also add a backslash between the last rubout typed and the next input character.

Example:	if the operator types:	12er<rubout><rubout>34,
	the console must print:	12er_\re_\34
	and will act as if:	1234
	was typed.	

Any given console will accept only certain characters as a command (i.e. the first letter typed on a line.) Once a valid command character is recognized, the syntax defines the next permissible character or characters. Whenever the console receives a character which is not permissible in the syntax or represents an unimplemented command, it will ignore the rest of the line (unless a control U or control C is typed) and print an error message upon receipt of the carriage return.

2.5 SUBSETTING

The console default data length is architecture dependent. All consoles MUST implement the following commands and qualifiers:

/P	Physical address space qualifier is default
/G	General Register address space
/I	Processor Internal Register address space (VAX only)
/L	Long word data length is default (VAX only)
/W	Word data length qualifier is default on PDP-11 and necessary on VAX implementations to reference UNIBUS physical address space
C	Continue
D	Deposit into memory, registers, or I/O space (data length default is architecture dependent)
E	Examine memory, registers, or I/O space (data length default is architecture dependent)
H	Halt
I	Init
X	Binary load command

The remaining commands and qualifiers described in this spec are optional, i.e. they are not required to exist, but if they do exist in a given implementation they have the syntax and effect described herein.

2.5.1 RECOMMENDED SUBSET

The following list of commands and qualifiers should be used as a guideline for implementors, If a given console will implement a subset of all possible commands, the ones closer to the top of the list should be implemented first, with the others added as toime and space allow:

6 manditory commands, /I (VAX only) and /G

S	Start
B	Bootstrap
T	Self test command
/B	Byte data length
/M	Machine dependent Internal Register space
/V	Virtual address space
N	Next command
/N	Next qualifier
M	Microstep command
/WCS	WCS qualifier
R	Repeat command

2.6 SUPERSETTING

It is concievable that a given CPU might want to implement a command not specified in this spec, The new command MUST nave an unique, distinct, abbreviation assigned to it, or it must start with a different letter from any of tne commands described here or in the Console Spec for the 11/780, Once the character denoting tne new command has been accepted by the architecture group, that command and its abbreviation will be entered into this spec, If some other implementor wished to add another command with different functionality, it MUST be assigned another unique abbreviation, This will insure that commands and their abbreviations have unique functionality wherever they are implemented,

3.0 CONSOLE COMMAND SYNTAX & SEMANTICS

< >	Angle brackets are used to denote category names. A category name is a label used to name a category. For example, the category name <address> may be used to represent any valid address, instead of actually listing all the strings of characters that can represent an address.
[]	Brackets surrounding part of an expression indicate that part of the expression is optional.
<SP>	represents one space.
<COUNT>	Represents a numeric count in the RADIX of the architecture.
<ADDRESS>	Represents an address argument. See section 5.2.1 for a list of valid <ADDRESS> types.
<DATA>	Represents a numeric argument.
<QUALIFIER>	A command modifier (switch). See section 5.
<INPUT-PROMPT>	Represents the console's input prompt string '>>>'.
<CR>	Carriage return.
<LF>	Line feed.

3.1 NOTATION EXAMPLE

E[<QUALIFIER-LIST>][<SP><ADDRESS>]<CR>

The explanation of the examine command is as follows:

1. An examine command may optionally contain a list of one or more qualifiers.

2. An examine command may optionally contain an address argument. If the address is specified it must be preceded by one space.

Listed below are several examples of valid examine commands.

```
E<CR>
E/B/V<CR>
E/V/B<CR>
E<SP>123456<CR>
E/W<SP>123456<CR>
```

4. COMMANDS

4.1 BOOT Command

SYNTAX: B[<SP><DEVICE-NAME>]<CR>

SEMANTICS: <DEVICE-NAME> is of the following format 'DDn', where 'DD' is a 2 letter device mnemonic (such as DT for DEC-Tape), and 'n' is a one-digit unit number.

If no <DEVICE-NAME> is given with the boot command, the console will perform the boot sequence for the default system device by either starting a ROM boostrap with predefined parameters, or by loading and executing a specific bootstrap program named 'DEFBOO.EXE' from the console load device. This program contains code which will bootstrap the system from a selected default device.

If a <DEVICE-NAME> is given with the command, the console will either calculate the correct parameters to be passed to the ROM bootstrap (or decide which ROM bootstrap to execute), or it may execute an program named ddnBOO.EXE, where 'ddn' is the <DEVICE-NAME> given.

Example:

'B RPØ' - will cause the console to either pass parameters to the ROM saying to boot from the Ø'th device on the first RH, or it may execute a program file named 'RPØBOO.EXE'.

RESPONSE: (Console enters program I/O mode, after starting the ROM program or executing the command file.)

NOTES

1) After CPU bootstrap completion, a response from the operating system will be displayed on the console terminal.

2) Bootstraps from devices other than the system default device are performed by passing different parameters or by loading and executing different bootstrap programs.

4.2 CONTINUE Command

SYNTAX: C<CR>

SEMANTICS: The CPU ISP begins instruction execution at the address currently contained in the CPU program counter (PC). CPU initialization is 'NOT' performed. The Console enters 'Program I/O' mode after issuing the Continue to the ISP. This command may be used to return the console to 'Program I/O' mode even if the CPU was already running.

RESPONSE: <CR><LF>(Console enters 'Program I/O' Mode)

4.3 DEPOSIT Command

SYNTAX: D[<QUALIFIER-LIST>]<SP><ADDRESS><SP><DATA><CR>

QUALIFIERS:

/B,/W,/L,/N,/V,/P,/I,/M,/G,/WCS. Refer to section 5 for a description of qualifiers & defaults.

SEMANTICS: Deposits (i.e. writes) <DATA> into the <ADDRESS> specified. The Address Space used will depend upon the Qualifiers specified with the command. If no Qualifiers are used, the current Address-Type default will determine the Address Space to be used. (PHYSICAL, VIRTUAL, INTERNAL REGISTER, GENERAL REGISTER.).

RESPONSE: <CR><LF><INPUT-PROMPT>

NOTE

In certain optional implementations, <ADDRESS> may also be one of the following symbolic addresses:

PS -Deposits to the processer status word (PDP-11).

PSL -Deposits to the processer status longword (VAX).

PC -Deposits to the program counter (General Register 7 (PDP-11) or F (VAX)).

SW -Deposits to the Switch Register (ISP reads from loc 177570 on PDP-11's ONLY)

SP -Deposits to the stack pointer

(General Register 6 (PDP-11) or E (VAX)).

Rn -Deposits to General Register n. 'n' is a number in the architecture default radix. Use of the '/G' qualifier is not necessary when 'Rn' is typed. Example:

D/G 5 1234 has the same effect as

D R5 1234

+ -Deposits to the location immediately following the 'Last' location referenced. For physical and virtual references the location referenced will be the 'Last Address' plus 'n' where n=1 for Byte, 2 for Word, 4 for Longword. For all other Address spaces 'n' is always equal to 1.

- -Deposits to the location immediately preceeding the 'Last' location referenced.

* -Deposits to the location last referenced.

@ -Deposits to the Address represented by the 'Last Data' examined or deposited.

E.G. 'E SP' - Examines the Stack Pointer.

'D @<DATA>' - Deposits <DATA> to the location specified by the contents of the stack pointer.

4.4 EXAMINE Command

SYNTAX: E[<QUALIFIER-LIST>][<SP><ADDRESS>]<CR>

QUALIFIERS: /B,/W,/L,/N,/V,/P,/I,/M,/G,/WCS. Refer to section 5 for a description of qualifiers & defaults.

SEMANTICS: Examines (i.e. Reads and displays) the contents of the specified <ADDRESS>. If no <ADDRESS> is specified, the current <DEFAULT-ADDRESS> is examined.

RESPONSE: <CR><LF><TAB><ADDRESS-SPACE-IDENTIFIER><ADDRESS> <DATA><CR><LF><INPUT-PROMPT>

NOTE

<ADDRESS> may also be one of the following symbolic address names:

PS -Displays the processer status word (PDP 11).

PSL -Displays the processer status longword (VAX).

PC -Displays the program counter (as in deposit)

SW -Displays the Switch Register (on PDP-11's only as in deposit)

SP -Displays the stack pointer (as in deposit)

Rn -Displays General Register 'n' (See Deposit command.)

+ -Displays the location immediately following the last location referenced.

- -Displays the location immediately preceeding the last location referenced.

* -Displays the last location referenced.

@ -Displays the location whose <ADDRESS> is the 'Last Data' deposited or examined.

Sample Examine Responses: (console output underlined)

```
>>> E/P  1234
----
      P  00001234  ABCDEF89
      --------------------

>>> E/V  1234
----
      P  0005634  01234567
      --------------------
```

NOTE: That the translated physical Address is displayed for Virtual Examines

```
>>> E/G 0
----
      G 00000000  98765432
      --------------------
```

4.5 INITIALIZE Command

SYNTAX: I<CR>

SEMANTICS: A CPU system initialize is performed.

RESPONSE: <CR><LF><INPUT-PROMPT>

4.6 HALT Command

SYNTAX: H<CR>

SEMANTICS: The CPU ISP will stop instruction execution after completing the execution of the instruction being executed when the console presents the HALT request to the CPU.

RESPONSE: <CONTENTS OF CPU PC><CR><LF><INPUT-PROMPT>.

4.7 LOAD Command

SYNTAX: L[<QUALIFIER-LIST>]<SP><FILE-SPECIFICATION><CR>

SEMANTICS: The Load command is used to read file data from the console's load device to main memory, or to the writable control store (WCS). If no qualifier is given with the Load command, physical main memory is loaded.

QUALIFIERS: 1) /S:<ADDRESS>

The 'START' Qualifier is used to specify a starting address for the load. If no 'START' Qualifier is given, the console will start loading at Address 0.

2) /WCS
The 'WCS' Qualifier is used to specify that the writable control store is to be loaded.

3) /P or /V
The 'Physical' or 'Virtual' Qualifier is used to force either physical or Virtual main memory as the destination of the load.

NOTE

If no qualifier for address space (WCS, Physical, Virtual), the destination of the LOAD is physical main memory.

4.8 MICROSTEP Command

SYNTAX: M[<SP><COUNT>]<CR>

SEMANTICS: The CPU is allowed to execute the number of MICRO-instructions indicated by <COUNT>. If no <COUNT> is specified, one instruction is performed, and the console enters 'SPACE-BAR-STEP' mode. (See Section 4.9.1)

The Console enters 'Program I/O' Mode immediately before issuing the Step, and re-enters 'Console I/O' Mode as soon as the Step completes. The ISP may be restarted by typing 'C', and will continue executing the current instruction. Typing an 'N' will cause the ISP to finish the current instruction before halting.

RESPONSE: <CR><LF><TAB> "MICRO PC= <CONTENTS OF MICRO PC>"

4.9 NEXT Command

SYNTAX: N[<SP><COUNT>]<CR>

SEMANTICS: The CPU is allowed to execute the number of MACRO-instructions indicated by <COUNT>. If no <COUNT> is specified, one instruction is

performed, and the console enters 'SPACE-BAR-STEP' mode. (See Section 4.9.1)

The Console enters 'Program I/O' Mode immediately before issuing the Step, and re-enters 'Console I/O' Mode as soon as the Step completes.

RESPONSE: <CR><LF><TAB> "HALTED AT <CONTENTS OF PC>"

4.9.1 SPACE-BAR-STEP Feature

1. Each time a 'NEXT' or MICROSTEP'' command with no <COUNT> argument is given to the console, the Step is executed and then the console may enter "SPACE-BAR STEP" mode. Each depression of the SPACE-BAR will cause 1 Step of the flavor currently enabled (Micro Cycle, Instruction).

2. A 'NEXT ' or 'MICROSTEP' command with an argument will not enable the Space-Bar feature.

   E.G. "NEXT 2" will cause 2 instructions to be executed, then the console will prompt for another command.

3. Exiting "SPACE-BAR STEP" mode - Type any character except "SPACE" to exit "SPACE-BAR STEP" mode.

4.10 REPEAT Command

SYNTAX: R<SP><CONSOLE COMMAND>

SEMANTICS: 'R <CONSOLE COMMAND>' causes the console to repeatedly execute the <CONSOLE COMMAND> specified until execution is terminated by a Control-C (^C) (see Section 5.1). Any valid console command may be specified for <CONSOLE COMMAND> with the exception of the 'repeat' command.

RESPONSE: <dependent upon command specified>

4.11 START Command

SYNTAX: S<SP><ADDRESS><CR>

or

S/WCS<SP><ADDRESS><CR>

SEMANTICS: 1) 'START <ADDRESS>'
The start command performs the equivalent of the

following sequence of console commands:

1. A CPU system 'initialize' is performed.
2. <ADDRESS> is deposited into the CPU program counter (PC).
3. A 'continue' is issued to begin CPU ISP execution.

RESPONSE: <CR><LF>

The console enters 'Program I/O' mode.

2) 'S/WCS <ADDRESS>'
1. <ADDRESS> is deposited to the Micro-PC.
2. The CPU microsequencer begins execution.

RESPONSE: <CR><LF><INPUT-PROMPT>

.12 Test Command

SYNTAX: T<CR>

SEMANTICS: The console subsystem will execute a self test, checking to insure its own integrety.

QUALIFIERS: /D

The 'Diagnose' Qualifier is used to cause CPU microdiagnostics to be run upon sucessful completion of the self test.

4.13 Binary Load Command

SYNTAX: X[<QUALIFIER-LIST>]<SP><ADDRESS><SP><COUNT><CR>

QUALIFIERS: /P,/WCS

SEMANTICS: The console will prepare to receive a string of binary data to be loaded into the address space specified by the <QUALIFIER-LIST>. Once the command has been parsed, the console will cease to echo input bytes received. The first byte of data is a CHECKSUM of the ASCII characters which comprised the command string, and will not be loaded into memory nor will <COUNT> be decremented. If the checksum does not compare, the console will respond with an error message,

re-enable echo of received characters, issue its input prompt and await another command. This will prevent inadvertent entry into a mode where the console is accepting the next several thousand input characters as data with the only way out being to turn power off.

Once the console has verified the byte checksum of the input string, it will deposit the data WORD by WORD into memory. If the <COUNT> was odd, the last byte will be byte 0 of the last word. As the console is depositing the data it is also adding the words together to form a checksum, and examining (if possible) the data it just stored to assure data integrity. If the <COUNT> is odd, the last word added to the checksum will have 0's in the high byte.

Once the <COUNT> is exhausted, the final two bytes transmitted will be the WORD CHECKSUM of all the data, with the low byte sent first. The console will compare the checksum and respond with an error message the received and computed checksums don't match. In any case, the console will re-enable echo, issue an input prompt, and await the next command.

5.0 Commands Performed While CPU is Running

Depending upon implementation, console commands may require that the CPU ISP be halted in order for the command to be executed. However, some console implementations may allow execution of certain commands while the ISP is running. The only restriction upon this is that the console should guarantee to have no effect upon the program currently running should an operation be performed which results in an error.

If a particular command may not be executed while the CPU is running, the console will respond with an error printout.

5.1 Control Characters & Special Characters

This section lists the control characters and special characters recognized by the console adaptor, and describes their function.

CONTROL-C(^C) Causes the suspension of all repetitive console operations such as:

1. Successive operations as a result of a /next qualifier.
2. Repeated command executions as a result of a 'repeat' command.

CONTROL-O(^O) Suppresses/enables console terminal output (toggle). Console terminal output is always enabled at the next console input prompt.

CONTROL-P(^P) Enters 'Console Mode' (if key switch is not LOCKED.) Characters typed are now fielded by the console, not the ISP.

CONTROL-U(^U) ^U typed before a line terminator causes the deletion of all characters typed since the last line terminator. The console echoes: '^U<CR><LF>'

CARRIAGE RETURN<CR> Terminates a console command line.

5.2 COMMAND QUALIFIERS AND DEFAULTS

Qualifiers are used within a command to specify the type or addressing and the length of data arguments (Sections 5.2.1 & 5.2.2). Qualifiers may be typed in any order. Defaults are applied by the console when a command does not contain a qualifier specifying

address-type or data-length. (Section5.3)

Certain commands permit an address argument to be defaulted (Section 5.4). The <DEFAULT-ADDRESS> used by the console is the next address following the last virtual, physical, or register address accessed by an examine or deposit command. Note that the next address is dependent upon data-length, since a byte reference updates the <Default-Address> by 1, while a long-word reference updates the <Default-Address> by 4.

The '/N' qualifier allows an examine or deposit command to operate on more than one address (Section 5.7).

5.2.1 Qualifiers for Address-Type

Qualifiers for address-type are used within a command line to specify the type of an address argument as either a Virtual memory or I/O space, Physical memory or I/O space, General Register, Processor Internal Register (VAX only), Machine Dependent Internal Register, or Writeable Control Store. The qualifiers for the respective types of addresses are: '/V', '/P', '/G', '/I', '/M' ,'/WCS'.

NOTE

> Virtual Addresses that reference non-existent or non-resident pages will cause the console to abort execution of the console command that referenced the virtual address. In each case, an appropriate error message will be displayed on the console terminal.

Example: To examine virtual address 1234, an operator would type:

E/V 1234

The Console will display the physical address corresponding to virtual address 1234, and the contents of that address.

5.2.2 Defaults for Address Types

The console till remember the last address qualifier typed, and use that address qualifier as the default address space for sucessive commands. When the console powers up, the default address space is [redacted]sical.

5.2.3 Qualifiers for Data Length

Qualifiers for data length are used within a command line to specify the length of a data quantity assovuated with the command. Data length may be specified as eithe byte, word, or long word (VAX only) by means of the'/B', '/W', or '/L'.

Example: The following command will display the byte at address 1233.

'E/B 1233'

5.2.4 Defaults for Data-Length

The console remembers the last data length qualifier which was typed. That data length is used as the default data length for sucessive console commands until a new data length is specified. The console initially uses a default data length of word for PDP-11's and long word for VAX's.

5.3 Default-Address Facility

Each time an Examine or Deposit command is executed, the console computes the address of the next memory location following the location referenced by the Examine or Deposit. The address of the next memory location is termed the <DEFAULT-ADDRESS>, since an examine command that does not specify an address will reference the next address by default. (See example below). The console computes the <DEFAULT-ADDRESS> as follows:

<DEFAULT-ADDRESS>=<Address used by last Examine or Deposit>+n

where "n" is 1 for byte references
2 for word references
4 for long-word references

The following example shows a sequence of console commands, and the value taken by the default address "after" each command is executed.

Example of default address facility: (All numbers are Hex)

COMMAND	ADDRESS USED	<DEFAULT-ADDRESS> AFTER EXECUTION
E/B 2341	2341	2342
E/W	2342 (USES<DEFAULT-ADDRESS>)	2344
E/L	2344 (USES<DEFAULT-ADDRESS>)	2348
E R0	GENERAL REGISTER 0	GENERAL REGISTER 1(R1)
E/G E	GENERAL REGISTER E(SP)	GENERAL REGISTER F(PC)
E	PC (USES<DEFAULT-ADDRESS>)	GENERAL REGISTER 0(R0)

Note that the <DEFAULT-ADDRESS> is R0 following a PC reference.

5.4 Specifying the <DEFAULT-ADDRESS> in a Command

The symbol '+' can be used as an address argument in a Deposit or Examine command, to represent the <DEFAULT-ADDRESS>. This symbol is provided to permit depositing to successive locations, without having to type the address argument after the first deposit.

Example: To 'TOGGLE-IN' a program starting at address 123456, the following deposit commands can be used:

D 123456 <DATA>

```
D     +     <DATA>

D     +     <DATA>

          etc.
```

Each Deposit command, after the first, puts the <DATA> into the next successive memory location.

5.5 Special Notation for Last Address

The last (Virtual, Physical, General Register, Internal Register, or WCS) address referenced via an examine or deposit command is denoted by an asterisk (*). The <Last-Address> may be used as an argument to an examine or deposit command by typing an asterisk in lieu of the address argument.

Example:

'E 1234'

will display the contents of location 1234.

If the next command issued is:

'D * 356'

The console will deposit the number 356 into address 1234.

5.6 Special Notation For 'Preceeding Address'

The symbol '-' (minus sign) may be used as an <ADDRESS> in a deposit or examine command to specify the location immediately preceeding the last location referenced.

5.7 Use Of 'Last Data' As An Address Argument

The symbol '@' may be used as an <ADDRESS> in a deposit or examine command. The last <DATA> examined or deposited will be used as the address.

5.8 The /N Qualifier

The /Next qualifier is provided to permit examine and deposit commands to operate on multiple sequential addresses. The Syntax of the /Next qualifier is:

<SLASH>N[:<COUNT>]

The <COUNT> argument specifies the number of additional executions of th command to be performed after the initial execution. The default value fo <COUNT> is one.

Example #1 The command:

E/B 1230/N:2

is evaluated by the console as follows:

1. The console initially evaluates the command and applies ar applicable default values.

2. The command with applied defaults is executed. The consol displays the contents of location 1230, and updates th <DEFAULT-ADDRESS> to 1231.

3. The /Next switch is now evaluated by the console. The consol repeats the command operation the number of times indicated b the <COUNT> argument. Each execution uses the <DEFAULT-ADDRESS for its address argument and updates the <DEFAULT-ADDRESS afterwards. In the above example, locations 1231 and 1232 ar successively displayed. The final value of the <DEFAULT-ADDRESS will be 1233.

Example #2 If the command:

E/N:2

is issued following the command in example #1, the contents o locations 1233, 1234, and 1235 will be displayed. Since the examin command does not contain an address argument, the initial executio of the command will use the current <DEFAULT-ADDRESS> which was 123 following the command in example #1.

NOTE

When using /Next qualifier to examine or deposit multiple CPU general registers, the 'next' register after the program counter (PC) is defined to be R0.

Example #3 The command:

E/N:5/G D

will display the contents of R13, SP, PC, R0, R1 and R2 in tha order.

6.Ø CONSOLE COMMUNICATION WITH THE OPERATING SYSTEM

The console adaptor's terminal, in addition to being the console adaptor's input device, also serves as the operating system operator's terminal. The console adaptor is said to be in 'Program I/O Mode' when the console terminal is being used as the operator's terminal. The console adaptor is in 'console I/O Mode' when the console terminal is being used to perform traditional console panel functions, and CPU hardware test and debug functions (the functions defined in sections 2 thru 5 of this specification).

6.1 Console I/O Mode

When the console adaptor is in 'Console I/O' mode, the console terminal serves as the operator interface to the console adaptor's console panel functions defined by this specification. All console terminal input is interpreted by the console adaptor, and appropriate console adaptor functions are invoked. Console terminal input is not passed to any CPU ISP-LEVEL software. The console will not accept any output from ISP-Level software running in the CPU. This implies that operator terminal output and system communication with the console's load device (if implemented) are disabled while the console is in console I/O mode.

6.2 Program I/O Mode

When the console adaptor is in 'Program I/O' mode, console terminal input is passed, character by character, to CPU ISP-LEVEL software. All validity checking, etc. is performed by the CPU software. The console adaptor operates transparently with respect to the CPU software. All terminal output from the CPU software is passed directly to the console terminal.

6.3 Console I/O Escape to Program I/O Mode

The console I/O escape sequence causes the console adaptor to transition from console I/O mode to program I/O mode. The console I/O escape sequence is the console command:

'C<CR>'

NOTE

The console commands: 'S', 'M', and 'N' also enable program I/O mode.

6.4 Program I/O Escape to Console I/O Mode

The program I/O escape sequence causes the console adaptor to transition from program I/O mode to console I/O mode. The Program I/O escape sequence is a Control-P("P).

NOTE

Control-P is not recognized if the console power switch is in the 'REMOTE DISABLE' or 'LOCAL DISABLE' position.

7.0 Operating System communication with console load device

This section was condensed from the 11/780 console spec, and does not really reflect how a TU58 would be used. Once the TU58 interface is clearer, this section will be updated to reflect it's functionality.

The operating system (OS) must be able to read and write the console sub-system's load device (TU58, Floppy disc, etc.) To acnieve this functionality, the following set of commands will be supported by the console software:

A. Write Sector - OS supplies track, sector, and 128 bytes of data. Console returns status upon completion of write.

B. Read Sector - OS supplies track and sector, console returns 128 bytes of data, and status of Read operation.

C. Read Status - Console returns load device status

D. Write sector with deleted data mark - OS supplies track and sector. (no data required). Console returns status upon completion of the write.

E. Cancel Function - Console aborts current load devicefunction.

The following functions will not be directly available to the OS:

Empty Silo, Fill Silo, Read error register, initialize.

While the OS initiated load device functions are in progress, operator terminal I/O is not disabled. Terminal I/O may be interspersed with load device I/O.

Once a Function is initiated, no other load device commands will be issued by the OS until the function is complete. The only exception is the command 'Cancel Function', which may be issued at any time.

The functions described in this document will only be available to the OS when the console is in 'Program I/O' mode. (i.e. the console terminal is being used as the system operator's terminal.)

NOTE

In the following protocols, two hardware side-effects are implied:

1. Each time the OS loads the 'Transmit Buffer' (TXDB), the 'TX Ready' bit in the 'Transmit Status Register' (TXCS), is automatically cleared. 'TXDB' is only loaded by the OS, and only when 'TX Ready' is set. 'TX Ready' is explicitly set by the console when the console is ready to accept another transfer thru 'TXDB'.

2. Each time the OS reads the 'Receiver Buffer' (RXDB), the 'RX DONE' bit in the 'Receiver Status Register' (RXCS) will automatically clear. 'RXDB' is only read by the OS, and only when 'RX DONE' is set. 'RX DONE' is explicitly set by the console each time the console has loaded 'RXDB' with a character for the OS.

7.1 Load Device Function Protocol

A. Write Sector/Write Deleted Data Sector

1. the OS puts 'write sector' or 'write-deleted-data sector' command into 'TXDB'.
2. Console takes write command, and sets 'TX Ready' in 'TXCS'.
3. The OS puts a 'Sector #' into 'TXDB'.
4. Console takes sector # and sets 'TX Ready'.
5. The OS puts a 'Track #' into 'TXDB'.
6. Console takes track # and sets 'TX Ready'.
7. The OS puts a byte of data into 'TXDB'.
8. Console accepts a byte of data and sets 'TX Ready'. Steps 7 & 8 are done 128 times for write sector. Steps 7 & 8 are skipped for write deleted data sector.
9. Console initiates Load Device write.
10. Write completes.
11. Console sends 'Function Complete' message. The 'Function Complete' message consists of loading RXDB Bits 8-11 with a select code of '2', and Bits 0-7 with the load device status byte. (see Sec 6.5.3 for a definition of the status byte)
12. The Os receives the 'Function Complete' message.

B. Read Sector

1. The OS puts 'Read Sector' command into TXDb.
2. Console takes read command, and sets 'TX Ready' in TXCS.
3. The OS puts a sector # into TXDB.

4. Console takes sector # and sets 'TX Ready'.

5. The OS puts a track # into TXDB.

6. Console takes track # and sets 'TX Ready'.

7. Console initiates read.

8. Read completes.

9. Console sends 'Function Complete' message. The 'Function Complete' message consists of a select code of '2' in Bits 8-11 of RXDB, and a Status Byte in Bits 0-7 of RXDB. (see Sec. 6.5.3 for Status Byte definition.)

10. The OS receives 'Function Complete'.

11. Console puts one byte of data in RXDB, and sets 'RX Done'.

12. The OS accepts one byte of data from RXDB.

Steps 11 and 12 are done 128 times. When the 128th byte is accepted by the OS, the read is complete.

NOTE

If a load deviceerror occurs on step 8, steps 11 and 12 will be skipped.

C. Read Status

1. The OS puts 'Read Status' command in TXDB.

2. Console takes 'Read Status' command, and sets 'TX Ready' in TXCS.

3. Console gets Status from last load device function performed.

4. Console puts 'Function Complete', with the Status, into 'RXDB' and sets 'RX Done'. (see Sec. 6.5.3 for Status definition)

5. The OS reads the load device status.

D. Terminate Function

1. The OS puts 'Cancel Function' command in TXDB.

2. Console takes 'Cancel Function'.

3. Console terminates Function in progress, if any.

4. Console sets 'TX Ready' in TXCS.

7.2 Miscellaneous Console Communications

The console software will support certain additional functional communications from the Operating System :

Software Communication Codes

1) Warm Restart Boot Command - The console will boot the VAX 11/780.

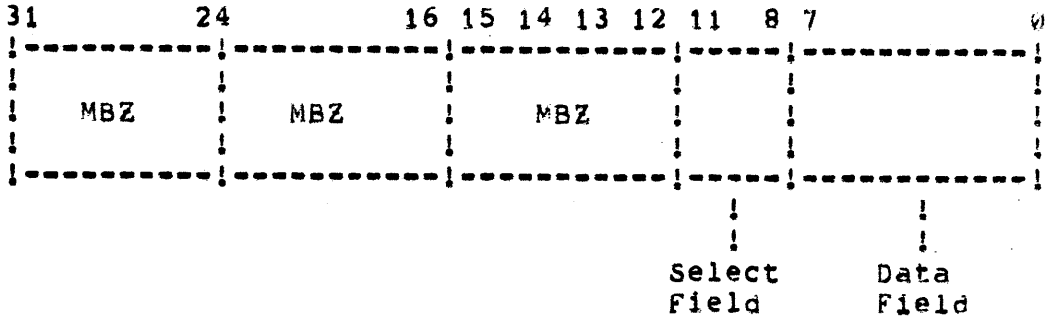
2) Clear Warm-Start and Cold-Start Flags - The operating system issues these codes when the OS has restarted/rebooted successfully. The console clears the associated flags.

NOTE

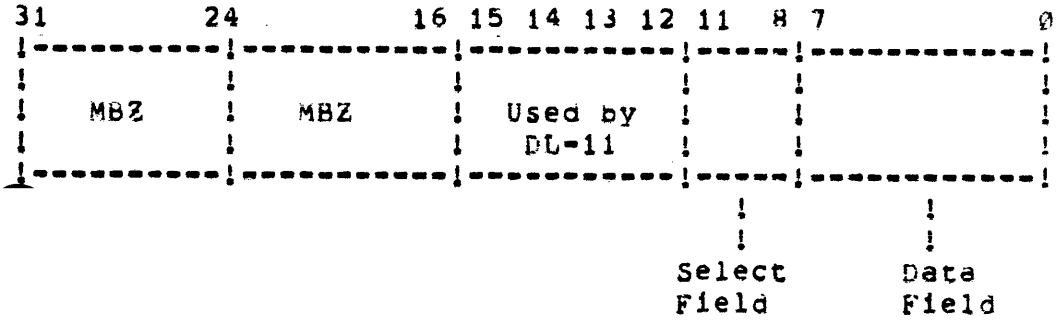
The 'cold' and 'warm' restart flags are used by the console to prevent infinite loops when a warm restart results in a CPU error halt.

7.3 Communication Register Formats & Select Codes

TXDB



RXDB



Select Field Values (in Hex)

Select Code	Device	Data Field Values
0	Operator's Terminal	0 thru 7F ! ASCII Data
1	Drive 0 (Data)	0 thru FF = Binary Data
2	Function Complete	(Status)
9	Drive 0 (Command)	0 = Read Sector 1 = Write Sector 2 = Read Status 3 = Write Deleted Data Sector 4 = Car 5 = Protocol Error
F	Misc. Communication	1 = Software Done 2 = Boo 3 = Cle 4 = Cle

NOTE

Code 5 (Protocol Error), is sent by the console when one of the following occurs:

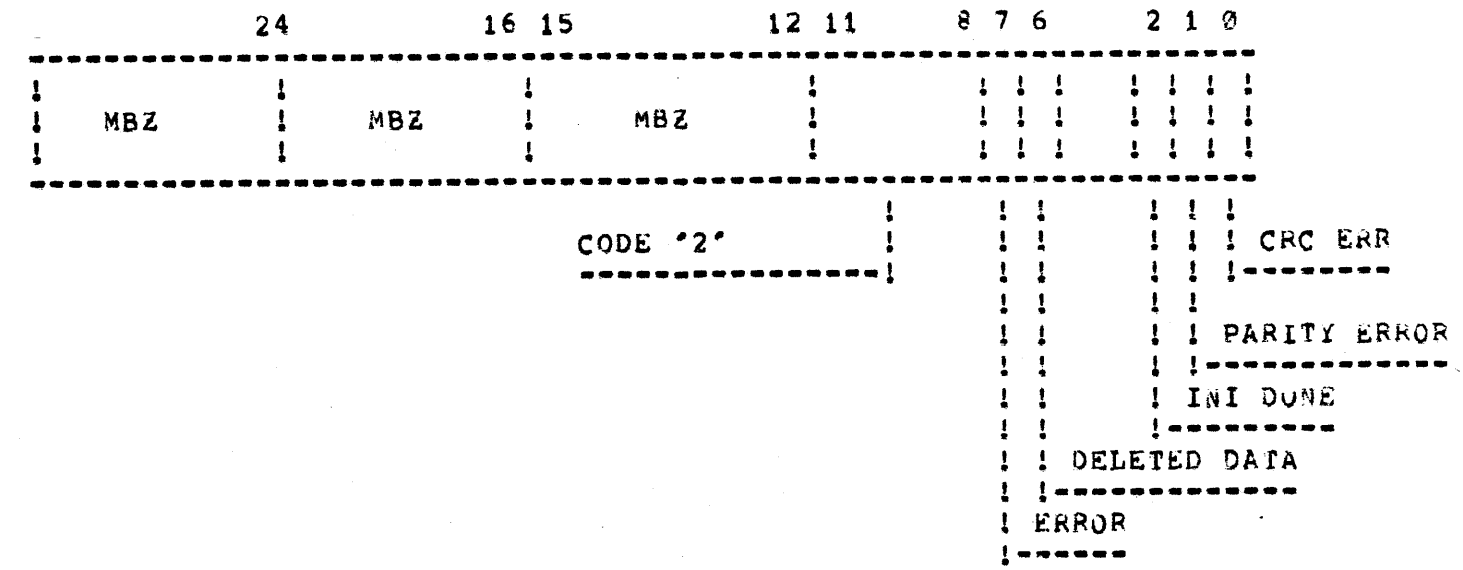
1) Another load device command (except for Cancel Function) is issued by the OS before a previous command is completed.

2) The console gets a 'Drive Ø (DATA)' when expecting a command.

7.3.1 Status Byte Definition

The Status Byte is used by the OS to determine the success or failure of a Read or Write operation. The Status Byte is sent to the OS at the completion of a Read, Write, or Read Status operation. The Select code is always 'Function Complete' (code 2). The Status Bit assignments are as follows:

RXDB



NOTE

The Status Bit assignments are identical to those supplied by the Floppy controller, excepting Bit 7. Bit 7 corresponds to Bit 15 of the Floppy's 'RXCS' Register.

8.0. ERRORS

All error messages have the following format:

?<ERROR NUMBER>[<SP><error message>]

Each distinct error message is assinged a number which is the same for all implementations. Certain implementations may also include an optional english message indicating the nature of the error. As with the command abbreviations, error numbers are unique and new errors must have numbers assigned to them by the architecture group.

8.1 ERROR LIST

Errors are T,B,S,

8.2 11/780 CONSOLE ERRORS

1.0 SYNTACTIC ERRORS

?'<ASCII STRING>' IS INCOMPLETE
The <ASCII STRING> is not a complete console command.

?'<ASCII STRING>' IS INCORRECT
The <ASCII STRING> is not recognized as part of a console command.

?FILE NAME ERR
A <FILENAME> given with a 'LOAD' or '@' command cannot be translated to RAD50.

2.0 COMMAND GENERATED ERRORS

?FILE NOT FOUND
A <FILENAME> given with a 'LOAD' or '@' command does not match any file on the current Floppy disc. This error can also be generated by a 'HELP' or 'BOOT' command if the help file or boot file is missing from the Floppy.

LT,

?CAN'T FIND MICMON.SYS
Generated when a 'TEST' command is issued and the Micro-diagnostic monitor file is missing from the current Floppy.

?CAN'T FIND WCSMON.SYS
Generated by a 'WCS' command when the control-store debugger file is missing from the current Floppy.

?NO CPU RESPONSE
The console timed out while waiting for a response from a STAR CPU micro-routine.

?CPU NOT IN CONSOLE WAIT LOOP, COMMAND ABORTED
A console command that requires the assistance of the STAR CPU was issued when the CPU is not in the console service loop.

?CPU CLOCK STOPPED, COMMAND ABORTED
A console command that requires the CPU clock to be running was issued with the clock stopped.

?IND-COM ERR
An indirect command file error was detected. This error is generated if:

1) An indirect command line exceeds 80 characters.

2) An indirect command line does not end with <CARRIAGE-RETURN> <LINE FEED>.

?CHM ERR
A change-mode instruction was attempted from the interrupt stack.

INT PENDING
This is not actually an error, but indicates that an error was pending at the time that a console-requested halt was performed.

3.0 MICRO-ROUTINE ERRORS

The console uses various micro-code routines in the STAR CPU's control store to perform console functions. The following errors are generated by micro-routine failures:

?MEM-MAN FAULT, CODE=XX
A virtual examine or deposit caused an error in the memory management micro-routine. 'XX' is a one byte error code supplied by the memory

management routine. See 'Star machine Check/Fault/Halt Spec' by C. Mathis for the definition of the error codes.

?MICRO-MACHINE TIME OUT
Indicates that the VAX 11/780 micro-machine has failed to strobe interrupts within the maximum time period allowed.

?MIC-ERR ON CONSOLE FUNCTION
An unspecified error occured while servicing a console request. Referencing non-existent memory will cause this error.

?INT-REG ERR
An error occured while referencing one of the STAR CPU internal (processor) registers. Specifying a register address that is too large will cause this error.

?MICRO-ERROR, CODE=XX
An unrecognized micro-error occured. 'XX' is the one-byte error code returned by the micro-routine.

4.0 CPU FAULT GENERATED ERROR MESSAGES

?INT-STACK INVALID
The STAR CPU interrupt stack was marked invalid.

?CPU DOUBLE-ERR HALT
The STAR CPU has done a 'Double Error Halt'.

?ILL I/E VECTOR
An illegal Interrupt/Exception vector was encountered by the STAR CPU.

?NO USR WCS
An Interrupt/Exception vector to WCS was encountered, and no WCS exists.

NOTE

See 'STAR Machine Check/Fault/Halt Spec', C. Matnis, for further information on these errors.

?MICRO-MACHINE TIME-OUT
Indicates that the VAX 11/780 micro-machine has failed to strobe interrupts within the maximum

time period allowed.

5.0 MESSAGES GENERATED BY FLOPPY ERRORS

?FLOPPY ERROR, CODE=X
The console Floppy driver detected an error. 'X' is an error code with the following meaning: ('X' is always in HEX).

CODE 0 -Floppy hardware error. (CRC, Parity, or a Floppy Firmware detected error).

CODE 1 -An 'Open' failed to find tne file specified.

CODE 2 -The Floppy driver queue is full.

CODE 3 -A Floppy sector was referenced that is out of the legal range of sector numbers.

?FLOPPY NOT READY
The console floppy drive failed to become ready while booting.

?FLOPPY ERROR ON BOOT
A console floppy error was detected while attempting a console boot.

?NO BOOT ON FLOPPY
The console attempted to boot from a flopoy that does not contain a valid boot block.

6.0 MESSAGES RELATING TO VERSION COMPATIBILITY

? Remote ACCESS NOT SUPPORTED
Printed when the console mode switch enters a 'Remote' position, and remote software support is not included in the console.

? WARNING - WCS EPLA VERSION MISMATCH
The microcode in WCS is not compatible with FPLA. This message is printed on eacn ISP start or continue, but no other action is taken by the console.

? FATAL - WCS PCS VERSION MISMATCH
The microcode in PCS is not compatible with that in WCS. ISP start and continue are disabled by the console.

7.0 CONSOLE-GENERATED ERRORS

? TRAP = 4 , RESTARTING CONSOLE
The console took a time-out trap. Console will restart.

? UNEXPECTED TRAP
The console trapped to an unused vector. Console reboots on insertion of console floppy, after Control-C is typed.

? Q-BLOCKED
Console's terminal output queue is blocked ; console will reboot.

APPENDIX A

LIST OF MACHINES AND THEIR COMMAND SETS

This is T.B.S.

Section 6 SOFTWARE ISSUES

1. Map Register Bit 15 incorporated in the high word of Unibus Mapping Registers to perform a cache Bypass is not implemented in the 11/68 since it is unnecessary. All Unibus references perform direct accesses to memory and bypasses cache on the 11/68.

2. All trap related functions in the Memory Management has been eliminated. We found no one who finds it useful.

3. The System I/D register 17 777 764 is not implemented since its function is to be performed by the new instruction MFPT.

4. System Size Register is not implemented since memory sizing is to be performed by all operating systems. This results in all non-existant memory traps (11/70 Vector 114) to perform TIMEOUT traps to Vector 4 (consistent with all other PDP-11's with exception of 11/70/74) with appropriate bits set in the CPU Error Register.

5. The RH68 is compatible with operation of the RH70 with the exception of bus parity on the internal bus (PPBI).

6. The 11/68 implements write buffering on CPU writes. Control is provided through MMR3 bit <6> for operations which require synchronization between write operations and processor operations. See full description of the operation of write buffering in the description of MMR3 Bit <6>.

7. Format of bits contained in Error and Diagnostic Registers reflect 11/68 processor specific functions. These include Cache Control, Cache/Memory System Error, Cache/Memory Maintenance Registers.

/mmm