1. Introduction

The P-Machine is a virtual machine usually implemented using an interpreter. The difference between the P-Machine and most other computer architectures is the P-Machine relies heavily on a stack. The P-Machine also supports additional data types not ordinarily available on conventional computers.

The P-Machine defines storage for registers, data, and instructions as shown in Figure 1. Registers, data, and instructions are discussed in succeeding sections.

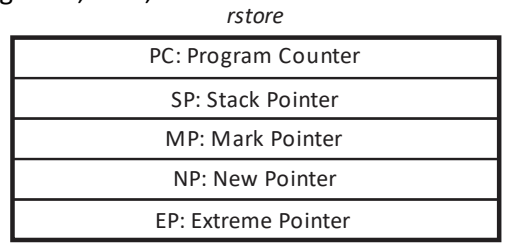

Figure 1. (a) Register Store

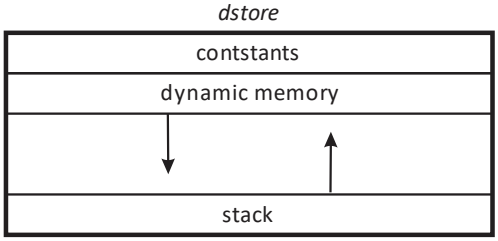

Figure 1. (b) Data Store

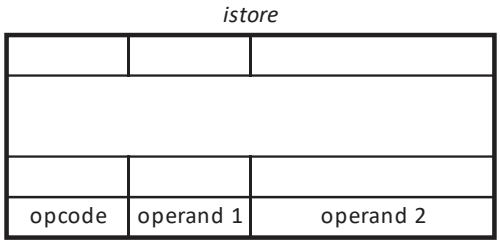

Figure 1 (c) Instruction Store

Figure 1. P-Machine Memory

2. Registers

The P-Machine has five registers: they are:

1. PC, the program counter
2. SP, the stack pointer
3. MP, the mark pointer
4. NP, the new pointer
5. EP, the extreme stack pointer

2.1. Program Counter Register (PC)

The program counter register contains the index of the current instruction being executed. Instructions are stored in array *istore*. Each element of the array occupies 32 bits and contains three fields, an operation code and two operands. The format of P-Machine instructions is shown in Figure 2. P-Machine programs are limited to 32768 instructions. Instructions are stored in array *istore* starting with element zero.

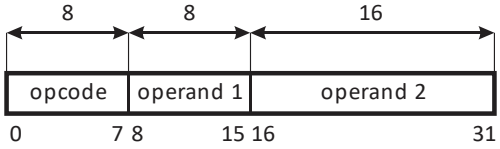

Figure 2. P-Code Instruction Format

The program counter is a 32-bit integer.

2.2. Stack Pointer (SP)

The stack pointer register contains the index of the element on top of the stack. Note that the stack is entirely contained in the data store. The data store is entirely composed of elements, each occupying eight (8) bytes. The stack pointer is an index of array *dstore*.

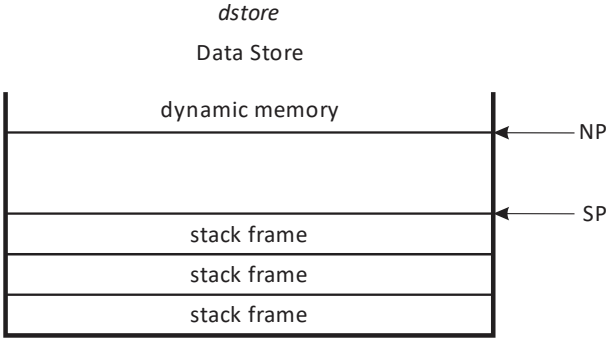

Figure 3. Anatomy of the Data Store

The anatomy of the data store is shown in Figure 3. The stack and heap grow toward each other as a P-Code program executes. Each time a procedure or function is called, another stack frame is allocated on the stack. As procedures and functions return to their callers, stack frames are removed from the stack.
The stack pointer marks the maximum extent of the stack.

The stack pointer register is a 32-bit integer.

2.3. New Pointer (NP)

The new pointer register marks the maximum extent of the heap. The new pointer register is an index of array *dstore*. Each time standard function new is called, storage for data referenced by the pointer returned by function new is allocated from the heap and the new pointer is moved toward the stack.
If the value of the stack pointer exceeds that of the new pointer the stack and heap have collided. This situation is an error and the P-Machine terminates the P-Code program.

2.4. Mark Pointer (MP)

The mark pointer register marks the beginning of a stack frame and it, too, contains an index of array *dstore*. The anatomy of a stack frame is shown in Figure 4. A stack frame is divided into sections, the mark, parameters, locals, and computation stack. The mark pointer register contains the index of the return value in the stack mark.
The mark pointer register is a 32-bit integer.

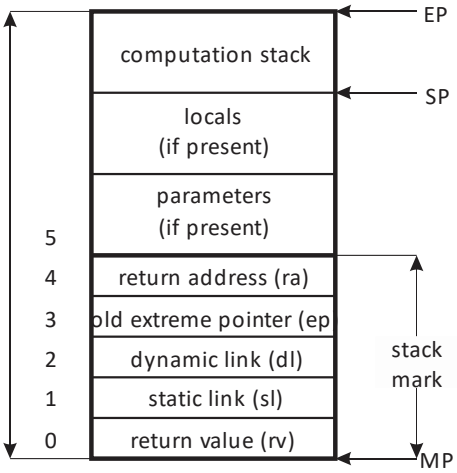

Figure 4. Anatomy of a Stack Frame

2.5. Extreme Pointer (EP)

The extreme pointer register marks maximum extent of the frame. The value of the extreme pointer relative to the mark pointer is the size of the stack mark, parameters, local variables, and largest computation stack. The stack will not grow beyond the value stored in the extreme pointer during the execution of a function or procedure.

3. Stack Frame

The anatomy of a stack frame is shown in Figure 4. A stack frame is allocated each time a procedure or function is called and removed when the subprogram returns. Each region of the frame is discussed below.

3.1. Return Value (rv)

The return value is storage reserved for the return value of a function: the value returned by a function is stored here. The return value is a single element of array *dstore* and occupies eight (8) bytes. Eight bytes are sufficient to store any of the P-Machine types.

3.2. Static Link (sl)

The static link references the frame on the stack associated with the immediately enclosing procedure or function. For example, if procedure A encloses procedure B, then the static link in the frame for procedure B references the frame for procedure A.

The static link is an index into array *dstore*. The static link occupies a single element of array *dstore*.

```
program main;
    var j:integer;
    procedure p;
        var i:integer;
        procedure q;
        begin
                if i>0 then
                begin
                        i:=i-1;
                        j:=j+1;
                        q
                end
        end{q};
    begin{p}
        i:=2;
        q;
    end{p};
begin{main}
    j:=0;
    p
end{main}.
```

Figure 5. Program main

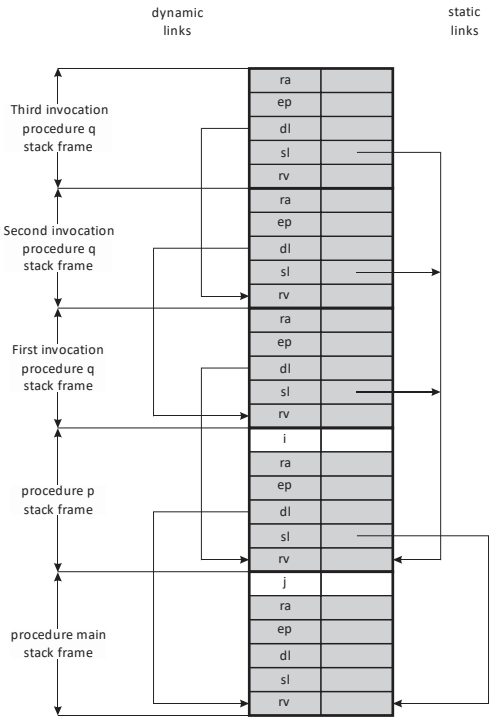

Figure 6. Static and Dynamic Links

Consider program main in Figure 5. The diagram in Figure 6 is a snapshot of the stack on the third (recursive) invocation of procedure q. Note the static links from all three invocations of procedure q reference the frame associated with the invocation of procedure p. Procedure p encloses procedure q. Non local references to variable i allocated in procedure p are accessed via the static link in the frame for procedure q.
The static link references the frame of the enclosing procedure. Procedure p encloses procedure q and static links in all three invocations of procedure q reference procedure p's frame.

3.3. Dynamic Link (dl)

The dynamic link references the caller's frame. For example, if procedure F called procedure G, then the dynamic link in procedure G's frame references the frame of procedure F.

The dynamic link is an index of array *dstore*. The dynamic link occupies a single element of array *dstore*.
Dynamic links illustrated in Figure 6 refer to the calling subprogram. Program main is called by the operating system so its dynamic link is not shown. Only procedures and functions of program main have valid dynamic links.

The dynamic link in procedure p's frame refers to program main.

The dynamic link in the first invocation of procedure q refers to the frame allocated for procedure p. Procedure p called procedure q.

Dynamic links in subsequent invocations of procedure q refer to the frame allocated for the previous invocation of procedure q. Procedure q is recursive: it calls itself.

3.4. Old Extreme Pointer (ep)

The old extreme pointer contains the value of the extreme pointer register. The value of the extreme pointer register is saved in the stack mark when a procedure or function is called and restored when the subprogram returns.

The old extreme pointer occupies a single element of array *dstore*.

3.5. Return Address (ra)

The return address contains the address of the instruction following the procedure call in the calling procedure. The return address is the next instruction to execute after the subprogram returns. The return address is an index of array *istore*. The return address occupies a single element of array *dstore*.

3.6. Parameters

Parameters are stored immediately above the stack mark. Parameters are optional. Procedures or functions without parameters require no storage for such.

The first parameter is located five elements from the mark pointer.
Parameters having type Boolean, character, integer, real, or set occupy one element in array *dstore*.

```
program params;
    var j:integer;
    procedure addparams
    (var i:integer);
begin{addparams}
    i:=i+1
end{addparms};
begin{params}
    j:=1;
    addparams(j)
end{params}.
```

Figure 7. Program params

iaddr	P-Code	Operand 1	Operand 2	Comment
0	ent	sp	5	Start procedure addparams Allocate storage for parameter i and the stack mark
1	ent	ep	3	Reserve 3 elements for the computation stack
2	lva	0	5	Load the value of parameter i: it is an address. Parameter i will be used as the target in the store indirect instruction in element 7 of array *iaddr*.
3	lva	0	5	Load the value of parameter i, an address, in preparation to obtain the value of variable i.
4	ind	i		Load indirect. Obtain the value of variable i
5	ldc	i	1	Load an integer constant 1.
6	adi			Add the two integers on top of the stack.

iaddr	P-Code	Operand 1	Operand 2	Comment
7	sti	i		Assign the sum to the location whose address is in parameter i. Assign the sum to variable j.
8	rtn	p		Return to the caller
9	ent	sp	5	Start program params Allocate storage for variable j and the stack mark
10	ent	ep	6	Reserve 6 elements for the computation stack
11	lda	0	5	Load the address of variable j
12	ldc	i	1	Load an integer constant 1.
13	sti	i		Assign the integer constant to variable j.
14	mst	0		Allocate storage for procedure addparams' stack mark
15	lda	0	5	Load the address of variable j just above the mark in the parameters section, passing variable j by reference.
16	cup	1	0	Call procedure addparams, having one argument.
17	rtn	p		Return to the caller
18	mst	0		**Execution starts here**. Create program params' stack mark.
19	cup	0	9	Call program params
20	stp			Stop

Figure 8. Annotated P-Code listing of program params

Consider program params in Figure 7 and the annotated P-Code listing of program params in Figure 8. Instructions for procedure addparams occupy elements 0 through 8 of array *istore*. Code for program main is stored in elements 9 through 17 of array *istore*. The program prologue, that creates the stack mark for program main, is stored in elements 18 and 19 of array *istore*.

Note that storage for one parameter is allocated by the entry (ent) instruction located in element 0 of array *istore*. The address of variable j is passed to procedure addparams by the instruction in element 15 of array *istore*.

3.7. Locals

Storage for variables local to the subprogram is allocated above parameters. Temporary variables allocated by the compiler may also be allocated in the local storage section. Procedures or functions without local variables require no storage for locals.

Consider program locals in Figure 9 and the corresponding listing of P-Code instructions in Figure 10. Variable i is local to procedure addlocal. Storage for variable i is allocated by the entry (ent) instruction in element 0 of array *istore*. Instructions stored in elements 0 through 7 implement procedure addlocal. Program locals is stored in elements 8 through 12 of array *istore* and the program prologue that creates the stack mark for program locals is stored in elements 13 and 14 of array *istore*.

```
program locals;
    procedure addlocal;
        var i:integer;
    begin{addlocal}
        i:=i+1
    end{addlocal};
begin{locals}
    addlocal
end{locals}.
```

Figure 9. Program locals

iaddr	P-Code	Operand 1	Operand 2	Comment
0	ent	sp	5	Start procedure addlocal Allocate storage for variable i and the stack mark
1	ent	ep	3	Reserve 3 elements for the computation stack
2	lda	0	5	Load the address of variable i in preparation to store a value.
3	lvi	0	5	Load the value of variable i
4	ldc	i	1	Load an integer constant 1.
5	adi			Add the two integers on top of the stack.
6	sti	i		Assign the sum to variable i.
7	rtn	p		Return to the caller
8	ent	sp	4	Start program locals Allocate storage for the stack mark
9	ent	ep	5	Reserve 5 elements for the computation stack
10	mst	0		Allocate storage for procedure addlocals' stack mark
11	cup	0	0	Call procedure addlocal.
12	rtn	p		Return to the caller
13	mst	0		**Execution starts here**. Create program main's stack mark.
14	cup	0	8	Call program local
15	stp			Stop

Figure 10. Annotated P-Code listing of program locals

3.8. Computation Stack

Storage for the computation stack is allocated in this region. The amount of storage required for the most complex computation in the subprogram is computed during compilation and used to determine the number of elements in the computation stack.

The purpose of reserving storage for computation in advance is to improve execution performance. Every time the stack pointer is incremented, it must be checked against the new pointer to prevent a collision. The stack cannot grow into the heap. By reserving the entire storage required for computation in the subprogram prologue, the P-Machine is not required to check the value of the new pointer every time the stack pointer is incremented. The extreme pointer delimits the maximum extent of the computation stack.

4. P-Machine register codes

P-Machine registers are assigned numeric codes in Table 1.

Table 1. P-Machine Registers

Register Mnemonic	Value (Hex)	Description
sp	00	Stack Pointer
ep	01	Extreme Pointer
mp	02	Mark Pointer
pc	03	Program Counter
np	04	New Pointer

5. P-Machine type codes

P-Machine types are assigned numeric codes in Table 2.

Table 2. P-Machine Types

Type	Value (Hex)	Description
a	00	Address
b	01	Boolean
c	02	Character
i	03	Integer
r	04	Real
s	05	String
t	06	Set
p	07	Procedure
x	08	Any of the above types

6. Instruction mnemonic and standard function codes

P-Machine instructions are assigned numeric codes in tables 3, 7, 9, 11, 13, 15, 17, 19, and 21. Notes discussing the function of P-Code instructions follow the tables that contain their numeric codes.

Standard function codes for the call standard procedure (csp) P-Code instruction are found in table 5.

Table 3. Subprogram Linkage P-Machine Operations

Mnemonic	P-Code (hex)	Operand 1	Operand 2	Operation on stack Before	Operation on stack After	Description
cup	00	*argsize*	*iaddr*	empty	empty	Call User Procedure
csp	01		*stdfunction*	function specific		Call Standard Procedure
ent	02	*register*	*amount*	empty	empty	Entry
mst	03		*level*	empty	empty	Mark Stack
rtn	04	*type*		empty	empty	Return

Table 4. Subprogram Linkage P-Machine Operation Notes

Operand Symbol	Description
argsize	*argsize* specifies the size of the arguments passed to the procedure called. *argsize* is specified in elements of array *dstore*. *argsize* is an unsigned integer in the range 0 to 255.
iaddr	*iaddr* is the instruction address that marks the first instruction of the procedure called. *iaddr* is an index of array *istore*. *iaddr* is an unsigned integer in the range 0 to 32767.
stdfunction	*stdfunction* is the index of a Pascal standard procedure or function. Indexes of standard procedures and functions are given in Table 5.
register	*register* is an integer value representing one of two registers, the stack pointer (sp) or the extreme pointer (ep). When coded as operand 1 of the entry instruction, sp represents a zero (0) and ep represents a (1).
amount	*amount* is the amount to be added to the value of the mark pointer register and assigned to the register in operand 1 of the entry instruction.

Table 4. Subprogram Linkage P-Machine Operation Notes (continued)

Operand Symbol	Description
level	*level* specifies which stack frame contains variables not local to the current subprogram. *level* specifies the number of static links to traverse before arriving at that stack frame allocated when the immediately enclosing procedure was called. *level* is computed by adding one to the level of the calling procedure minus the level of the called procedure.
type	*type* is a P-Machine type. P-Machine types are listed in Table 2.

6.1. Call User Procedure (cup)

Note: Variable *p* is operand 1 and variable *q* is operand 2 of a p-code instruction.

1. *mp***:=***sp***-(***p***+***4***);**
   The mark pointer is assigned a value relative to the stack pointer, *sp*. If no arguments are passed to the subprogram being invoked then the stack pointer references the location assigned to the return address (ra) in the stack mark. The mark pointer must reference the position (0) assigned to the return value (rv) in the stack mark. The relative address of the return value is numerically four (4) positions less than the return address. Please refer to figure 11.
2. *dstore***[***mp***+4].***a***:=***pc***;**
   Assign the return address to its location in the stack mark.
3. *pc***:=***q***;**
   Assign the starting address of the subprogram being called to the program counter, *pc*.

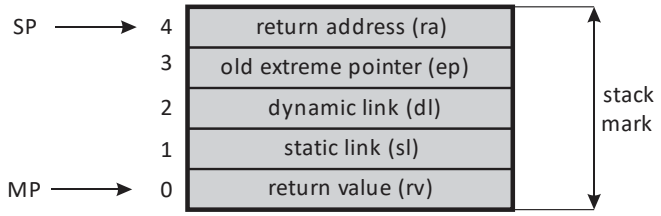

Figure 11. Subprogram invocation with no arguments.

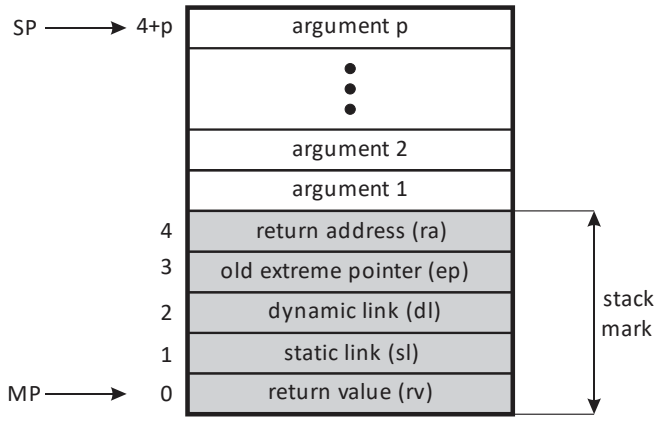

Figure 12. Subprogram invocation with *p* arguments.

6.2. Call Standard Procedure (csp)

Call Standard Procedure is used to call standard procedures and functions listed in Table 5.

6.3. Entry (ent)

The Entry operation is used to allocate storage on the current stack frame for local variables and for the computation stack.

6.4. Mark Stack (mst)

The Mark Stack instruction allocates storage for the stack mark, saves the value of the old extreme pointer, assigns the value the dynamic link, and computes the value of the static link.

```
dstore[sp+2].vm:=base(p);     {The static link is set to p minus the level of the called
                               subprogram + 1}
dstore[sp+3].vm:=mp;          {Assign the value of the dynamic link}
dstore[sp+4].vm:=ep;          {Save the value of the old extreme pointer}
sp:=sp+5;                     {Allocate storage for the stack mark just created.}
```

Figure 13. Mark Stack implementation

6.5. Return (rtn)

The Return instruction restores the sp, ep, mp, and pc registers.

```
if p>0 then sp:=mp else sp:=mp-1;   {Put the return value on top of the stack if
                                     if the subprogram was a function}
pc:=store[mp+4].vm;                 {Assign the pc to the return address}
ep:=store[mp+3].vm;                 {Restore the extreme pointer}
mp:=store[mp+2].vm;                 {Restore the mark pointer and deallocate
                                     the previous stack frame}
```

Figure 14. Return implementation

Table 5. Standard Functions

Mnemonic	Opcode (hex)	Operation on stack Before	After	Description
rdb	00	(a,f)	empty	Read Boolean
rdc	01	(a,f)	empty	Read Character
rdi	02	(a,f)	empty	Read Integer
rdr	03	(a,f)	empty	Read Real
rln	04	(f)	empty	Read Line
wrb	05	(w,b,f)	empty	Write Boolean
wrc	06	(w,c,f)	empty	Write Character
wri	07	(w,i,f)	empty	Write Integer
wre	08	(d,w,r,f)	empty	Write Real, Exponential Format
wrf	09	(d,w,r,f)	empty	Write Real, Fixed Format
wrs	0A	(a,f)	empty	Write String, *a*=address of string
wrt	0B	(t,f)	empty	Write Set
wln	0C	(f)	empty	Write Line
sqt	0D	(r)	sqrt(r)	Square Root
ln	0E	(r)	ln(r)	Natural Logarithm
exp	0F	(r)	exp(r)	Exponentiation

Table 6. Standard Function Notes

Symbol	Description
(x_1,x_2,x_3,x_4)	Operands on the stack are enclosed in parentheses and separated by commas. The leftmost operand, x_1, is on top of the stack. Subsequent operands, x_2, x_3, and x_4 are on the stack, under x_1, in the order listed. For example, element x_4 is fourth from the top of the stack.
(a,f)	Operand a represents the address of a variable and operand f represents the file to be read. For example, standard function rdb reads a Boolean value, true or false, from file f and stores a zero or a one, depending on which value was read, into the Boolean value whose address "a", is on top of the stack.
(t,f)	Set value t is formatted and written to file f.
(w,b,f)	Boolean value b is right justified in a field of width w in file f.
(w,c,f)	Character value c is right justified in a field of width w in file f.
(w,i,f)	Integer value i is right justified in a field of width w in file f.

(d,w,r,f)	Real value r is right justified in a field of width w having d fraction digits in file f.
(r)	Operand r is a real value on top of the stack

Table 6. Standard Function Notes (continued)

Symbol	Description
sqrt(r)	sqrt(r) is the square root of r.

Table 7. Comparison P-Machine Operations

Mnemonic	P-Code (hex)	Operand 1	Operand 2	Operation on stack Before	After	Description
equ	05	*type*		(x_1,x_2)	b	Equality comparison, b:=x_2=x_1
neq	06	*type*		(x_1,x_2)	b	Inequality comparison, b:=x_2<>x_1
grt	07	*type*		(x_1,x_2)	b	Greater than comparison, b:=x_2>x_1
geq	08	*type*		(x_1,x_2)	b	Greater than or equal comparison, b:=x_2>=x_1
les	09	*type*		(x_1,x_2)	b	Less than comparison, b:=x_2<x_1
leq	0A	*type*		(x_1,x_2)	b	Less than or equal comparison, b:=x_2<=x_1

Table 8. Comparison P-Machine Operation Notes

Operand Symbol	Description
(x_1,x_2)	Operand x_1 is on top of the stack and operand x_2 is next to the top of the stack
(x_1,x_2)	Operands x_1 and x_2 are values having the same type. Comparisons compare two values having the same type
b	The value b is the Boolean result of comparing two values of the same type
type	*type* is a P-Machine type. P-Machine types are listed in Table 2.

Table 9. Arithmetic P-Machine Operations

Mnemonic	P-Code (hex)	Operand 1	Operand 2	Operation on stack Before	After	Description
adi	0B			(i_1,i_2)	i_3	Integer addition, i_3:=i_2+i_1
sbi	0C			(i_1,i_2)	i_3	Integer subtraction, i_3:=i_2-i_1
ngi	0D			(i_1)	i_2	Integer sign inversion, i_2:=-i_1
mpi	0E			(i_1,i_2)	i_3	Integer multiplication, i_3:=i_2*i_1
dvi	0F			(i_1,i_2)	i_3	Integer division, i_3:=i_2 **div** i_1

Table 9. Arithmetic P-Machine Operations (continued)

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
mod	10			(i_1,i_2)	i_3	Modulo, i_3:=i_2 **mod** i_1
abi	11			(i_1)	i_2	Integer absolute value, i_2:=|i_1 |
sqi	12			(i_1)	i_2	Integer square, i_2:=i_1^2

Table 9. Arithmetic P-Machine Operations (continued)

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
inc	13	*i-type*		(x)	x+1	Increment
dec	14	*i-type*		(x)	x-1	Decrement
adr	15			(r_1,r_2)	r_3	Real addition, r_3:=r_2+r_1
sbr	16			(r_1,r_2)	r_3	Real subtraction, r_3:=r_2-r_1
ngr	17			(r_1)	r_2	Real sign inversion, r_2:=-r_1
mpr	18			(r_1,r_2)	r_3	Real multiplication, r_3:=r_2*r_1
dvr	19			(r_1,r_2)	r_3	Real division, r_3:=r_2/r_1
abr	1A			(r_1)	r_2	Real absolute value, r_2:=|r_1|
sqr	1B			(r_1)	r_2	Real square r_2:=r_1^2

Table 10. Arithmetic P-Machine Operation Notes

Operand Symbol	Description
(i_1,i_2)	i_1 is on top of the stack and i_2 is next to the top of the stack
(r_1,r_2)	r_1 is on top of the stack and r_2 is next to the top of the stack
i_1, i_2, i_3	i_1, i_2, and i_3 are integers.
r_1, r_2, r_3	r_1, r_2, and r_3 are real numbers.
i-type	Integral type including *a*-address, *b*-boolean, *c*-character, and *i*-integer.

Table 11. Boolean P-Machine Operations

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
	(hex)			Before	After	
ior	1C			(b_1,b_2)	b_3	Inclusive OR, b_3:=b_2∪b_1
and	1D			(b_1,b_2)	b_3	AND, b_3:=b_2∩b_1
xor	1E			(b_1,b_2)	b_3	Exclusive OR, b_3:=b_2⊕b_1
not	1F			(b_1)	b_2	Complement, b_2:=**not** b_1

Table 12. Boolean P-Machine Operation Notes

Operand Symbol	Descriptions

(b_1,b_2)	b_1 is on top of the stack and b_2 is next to the top of the stack
b_1, b_2, b_3	b_1, b_2, and b_3 are Boolean values.

Table 13. Set P-Machine Operations

Mnemonic	P-Code (hex)	Operand 1	Operand 2	Operation on stack Before	Operation on stack After	Description
inn	20			(t,i)	b	Set membership, b:=i∈t
uni	21			(t_1,t_2)	t_3	Set union, t_3:=t_2∪t_1
ntr	22			(t_1,t_2)	t_3	Set intersection, t_3:=t_2∩t_1
dif	23			(t_1,t_2)	t_3	Set difference, t_3:=t_2∩ **not** t_1
cmp	24			(t_1)	t_2	Complement, t_2:=**not** t_1
sgs	25			(i)	t	Generate singleton set

Table 14. Set P-Machine Operation Notes

Operand Symbol	Description
(t,i)	t is a bit-vector representing a set of 64 elements numbered 0 to 63 from most significant bit to least significant bit. i is an integer value between 0 and 63. If the ith bit is set, Boolean value b is assigned true, otherwise it is assigned false.
(t_1,t_2)	t_1 is on top of the stack and s_2 is next to the top of the stack
t_1, t_2, t_3	t_1, t_2, and t_3 are sets having a maximum of 64 elements.

Table 15. Jump P-Machine Operations

Mnemonic	P-Code (hex)	Operand 1	Operand 2	Operation on stack Before	Operation on stack After	Description
ujp	26		*iaddr*	empty	empty	Unconditional jump
xjp	27		*iaddr*	(i)	empty	Indexed jump
fjp	28		*iaddr*	(b)	empty	False jump
tjp	29		*iaddr*	(b)	empty	True jump

Table 16. Jump P-Machine Operation Notes

Operand Symbol	Description
iaddr	*iaddr* is an instruction address in the range 0 to 32767.
(i)	i is an integer offset from *iaddr*. The indexed jump instruction is used to implement case statements
b	b is a Boolean value

Table 17. Conversion P-Machine Operations

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
	(hex)			Before	After	
flt	2A			(i)	r	Float top of stack
flo	2B			(x,i)	(x,r)	Float next to top of stack
trc	2C			(r)	i	Truncate
rnd	2D			(r)	i	Round
chr	2E			(i)	c	Convert to character
ord	2F			(x)	i	Convert to integer

Table 18. Conversion P-Machine Operation Notes

Operand Symbol	Description
i	i is an integer value
r	r is a real value
(x,i)	x is a value of any type on top of the stack and i is an integer value next to the top of stack.

Table 19. Program Termination P-Machine Operations

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
	(hex)			Before	After	
stp	30			empty	empty	Stop

Table 20. Program Termination P-Machine Operation Notes

Operand Symbol	Description
stp	stp is always the last instruction in a P-Machine Program

Table 21. Data Reference P-Machine Operations

Mnemonic	P-Code	Operand 1	Operand 2	Operation on stack		Description
	(hex)			Before	After	
lda	31	*level*	*offset*	empty	a	Load address of data
ldc	32	*type*	*cindex*	empty	x	Load constant
ldi	33	*type*		(a)	x	Load indirect
lva	34	*level*	*offset*	empty	a	Load value (address)
lvb	35	*level*	*offset*	empty	b	Load value (Boolean)
lvc	36	*level*	*offset*	empty	c	Load value (character)
lvi	37	*level*	*offset*	empty	i	Load value (integer)
lvr	38	*level*	*offset*	empty	r	Load value (real)
lvs	39	*level*	*offset*	empty	s	Load value (string)
lvt	3A	*level*	*offset*	empty	t	Load value (set)
sti	3B	*type*		(x,a)	empty	Store indirect
ixa	3C		*stride(q)*	(a_1,i)	a_2	Compute indexed address $a_2 := q*i+a_1$

Table 22. Data Reference P-Machine Operation Notes

Operand Symbol	Description
level	*level* references the frame in which the value sought is found. When the value of *level* is zero, the value sought is in the frame on top of the stack. When the value of level is one or greater, the value sought is one or more frames distant in the stack following the static link. For example, if the value of level was two, the value sought is in that frame that can be found following the static link twice.
offset	*offset* is the amount added to the mark pointer register to obtain the desired value. Values are stored in elements of array *dstore*. Recall that the mark pointer register always contains a valid index of array *dstore*.
type	*type* is a P-Machine type. P-Machine types are listed in Table 2.
cindex	*cindex* is a constant index. A P-Machine program contains tables of integer, real, set, and string constants. If the value of type is i (4), r (5), s(6) or t(7), the value of the constant is in the associated constant table. Boolean and character constants are coded in operand 2 directly.
stride	*stride* is the number of words occupied by an element of the array referenced