Quick Reference to the P machine Instructions

Datatypes	
a	Address
b	Boolean
c	Character
i	Integer
r	Real
N	Numeric = i, r
T	Type = a, b, c, i, r

		Conditions	**Result**
Arithmetic instructions			
add N	STORE[SP-1] := STORE[SP-1] $+_N$ STORE[SP] SP := SP – 1	(N, N)	(N)
sub N	STORE[SP-1] := STORE[SP-1] $-_N$ STORE[SP] SP := SP – 1	(N, N)	(N)
mul N	STORE[SP-1] := STORE[SP-1] $*_N$ STORE[SP] SP := SP – 1	(N, N)	(N)
div N	STORE[SP-1] := STORE[SP-1] $/_N$ STORE[SP] SP := SP – 1	(N, N)	(N)
neg N	STORE[SP] := $-_N$ STORE[SP]	(N)	(N)
Logical instructions			
and	STORE[SP-1] := STORE[SP-1] *and* STORE[SP] SP := SP – 1	(b, b)	(b)
or	STORE[SP-1] := STORE[SP-1] *or* STORE[SP] SP := SP – 1	(b, b)	(b)
not	STORE[SP] := *not* STORE[SP]	(b)	(b)
Comparison instructions			
equ T	STORE[SP-1] := STORE[SP-1] $=_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
geq T	STORE[SP-1] := STORE[SP-1] $\geq_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
leq T	STORE[SP-1] := STORE[SP-1] $\leq_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
les T	STORE[SP-1] := STORE[SP-1] $<_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
grt T	STORE[SP-1] := STORE[SP-1] $>_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
neq T	STORE[SP-1] := STORE[SP-1] $\neq_T$ STORE[SP] SP := SP – 1	(T, T)	(b)
Store and load instructions			
ldo T q	SP := SP + 1 STORE[SP] := STORE[q]	Type(q) = a	(T)
ldc T q	SP := SP + 1 STORE[SP] := q	Type(q) = T	(T)

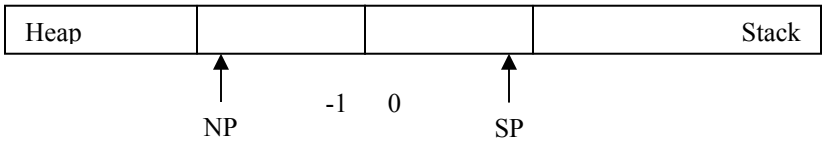
Input of a charactervalue:

- *ldc c 'a'*
- *ldc c ASCII-value (e.g.: for exotic characters of escape characters)*

ind T	STORE[SP] := STORE[STORE[SP]]	(a)	(T)
sro T q	STORE[q] := STORE[SP] SP := SP – 1	Type(q) = a, (T)	

sto T	STORE[STORE[SP – 1]] := STORE[SP] SP := SP – 2	(a, T)	
Conditional and unconditional branches			
ujp *label*	PC := *label*		
fjp *label*	if STORE[SP] = false then PC := *label* fi SP := SP - 1	(b)	
Indexed branch			
ixj *label*	PC := STORE[SP] + *label* SP := SP - 1	(i)	
Computation of indexed address			
ixa q	STORE[SP – 1] := STORE[SP – 1] + STORE[SP] * q SP := SP - 1	(a, i) Type(q) = i	(a)
Increment and decrement			
inc T q	STORE[SP] := STORE[SP] + q	(T) and Type(q) = i	(T)
dec T q	STORE[SP] := STORE[SP] - q	(T) and Type(q) = i	(T)
Boundary check			
chk p q	if(STORE[SP] < p) *or*(STORE[SP] > q) then error('value out of range') fi	(i), Type(p) = Type(q) = i	(i)
Instructions for dynamic arrays			
dpl T	SP := SP + 1 STORE[SP] := STORE[SP – 1]	(T)	(T, T)
ldd q	SP := SP + 1 STORE[SP] := STORE[STORE[SP – 3] + q]	(a, T_1, T_2) Type(q) = i	(a, T_1, T_2, i)
sli T_2	STORE[SP – 1] := STORE[SP] SP := SP - 1	(T_1, T_2)	(T_2)

Memory organization: the memory organization has been slightly changed with regard to the original P-machine specifcation found in Compiler Design by Reinhard Wilhelm and Dieter Maurer.



No maximum sizes for Stack or Heap are defined. They are simply bounded by the size of a standard positive integer on your platform.

Dynamic memory			
new	if NP – STORE[SP] ≤ EP then error('store overflow') else NP := NP – STORE[SP] STORE[STORE[SP – 1]] := NP SP := SP – 2 Fi	(a,i)	

Definition: *base*(p,a) = **if** p = 0 **then** a **else** *base*(p – 1, STORE[a + 1]) **fi.**

Loading and storing for difference in nesting depths		
lod T p q	SP := SP + 1 STORE[SP] := STORE[base(p, MP) + q)]	
lda p q	SP := SP + 1 STORE[SP] := base(p, MP) + q	

str T p q	STORE[base(p, MP) + q] := STORE[SP] SP := SP - 1	
Instructions for calling and entering procedures		
mst p	STORE[SP + 2] := base(p, MP) STORE[SP + 3] := MP STORE[SP + 4] := EP SP := SP + 5	*Static link* *Dynamic link* *The parameters can now ben evaluated starting from STORE[SP + 1]*
cup p *label*	MP := SP – (p + 4) STORE[MP + 4] := PC PC := *label*	*p is the storage requirement for the parameters* *save return address* *branch to procedure start*
ssp p	SP := MP + p –1	*p size of static part of data area*
sep p	EP := SP + p if EP ≥ NP then error (‘store overflow’) fi	*p max. depth of local stack* *Check for collision of stack and heap*
ent p q	SP := MP + q – 1 EP := SP + p if EP ≥ NP then error(‘store overflow’) fi	*q data area size* *p max. depth of local stack* *Collision of stack and heap*
Return from function procedures and proper procedures		
retf	SP := MP PC := STORE[MP + 4] EP := STORE[MP + 3] if EP ≥ NP then error(‘store overflow’) fi	*Function result in the local stack* *Return branch* *Restore EP*
retp	SP := MP – 1 PC := STORE[MP + 4] EP := STORE[MP + 3] if EP ≥ NP then error(‘store overflow’) fi MP := STORE[MP + 2]	Proper procedure with no results Return branch Restore EP Dynamic link
Block copy instructions		
movs q	for i := q – 1 down to 0 do STORE[SP + i] := STORE[STORE[SP] + i] Od SP := SP + q - 1	(a)

movd q	for i = 1 to STORE[MP + q + 1] do
	  STORE[SP + i] := STORE[STORE[MP + q]
	  + STORE[MP + q + 2] + i – 1]
	od
	STORE[MP + q] := SP + 1 – STORE[MP + q + 2]
	SP := SP + STORE[MP + q + 1]	

Instructions for procedures		
smp p	MP := SP – (p + 4)	
cupi p q	STORE[MP + 4] := PC PC := STORE[base(p, STORE[MP + 2]) + q]	*Return address*
mstf p q	STORE[SP + 2] := STORE[base(p, MP) + q + 1] STORE[SP + 3] := MP STORE[SP + 4] := EP SP := SP + 5	
Halt instruction		
hlt		*Halts the P machine*

Input/Output instructions		
in i in r in c in b	SP := SP + 1 STORE[SP] := *value*	*value = input from stdin* *Booleanvalue: either t or f*
out r i	SP := SP – 2	*condition: (r,i) → the real at STORE[SP –1] is written with precision found at STORE[SP]* *Output is written to stdout*
out i out r out c out b out a	SP := SP – 1	 *The real is written with standard precision.* *STORE[SP] is written to stdout*
Conversion instruction		
conv T_1 T_2	precondition: Type(STORE[SP]) = T_1 postcondition: Type(STORE[SP]) = T_2	*Converts the element at STORE[SP]. Conversions follow the standard C++-conversion-rules.*