COP 3402 Systems Software

Virtual Machines
as instruction
interpreters

Outline

1. Virtual machines as software interpreters
2. P-code: instruction set architecture
3. The instruction format
4. Assembly language

Virtual Machine: P-code

The Pseudo-code machine is a software (virtual) machine that implements the instruction set architecture of a computer.

P-code was implemented in the 70s to generate intermediate code for Pascal compilers.

Another example of a virtual machine is the JVM (Java Virtual Machine) whose intermediate language is commonly referred to as Java bytecode.

The P-machine Instruction format (PM/0)

The ISA of the PM/0 has 22 instructions and the instruction format has three components <op, l, m>:

OP is the operation code.

L indicates the lexicographical level.

M depending of the opcode it indicates:

- A number (instructions: LIT, INT).

- A program address (instructions: JMP, JPC, CAL).

- A data address (instructions: LOD, STO)

- The identity of the operator OPR(i.e. OPR 0, 2 (ADD) or OPR 0, 4 (MUL)).

Virtual Machine: P- code

The interpreter of the P-machine(PM/0) consists of:

A store named **“stack”** organized as a stack.

A **“code”** store that contains the instructions.

The CPU has four registers:

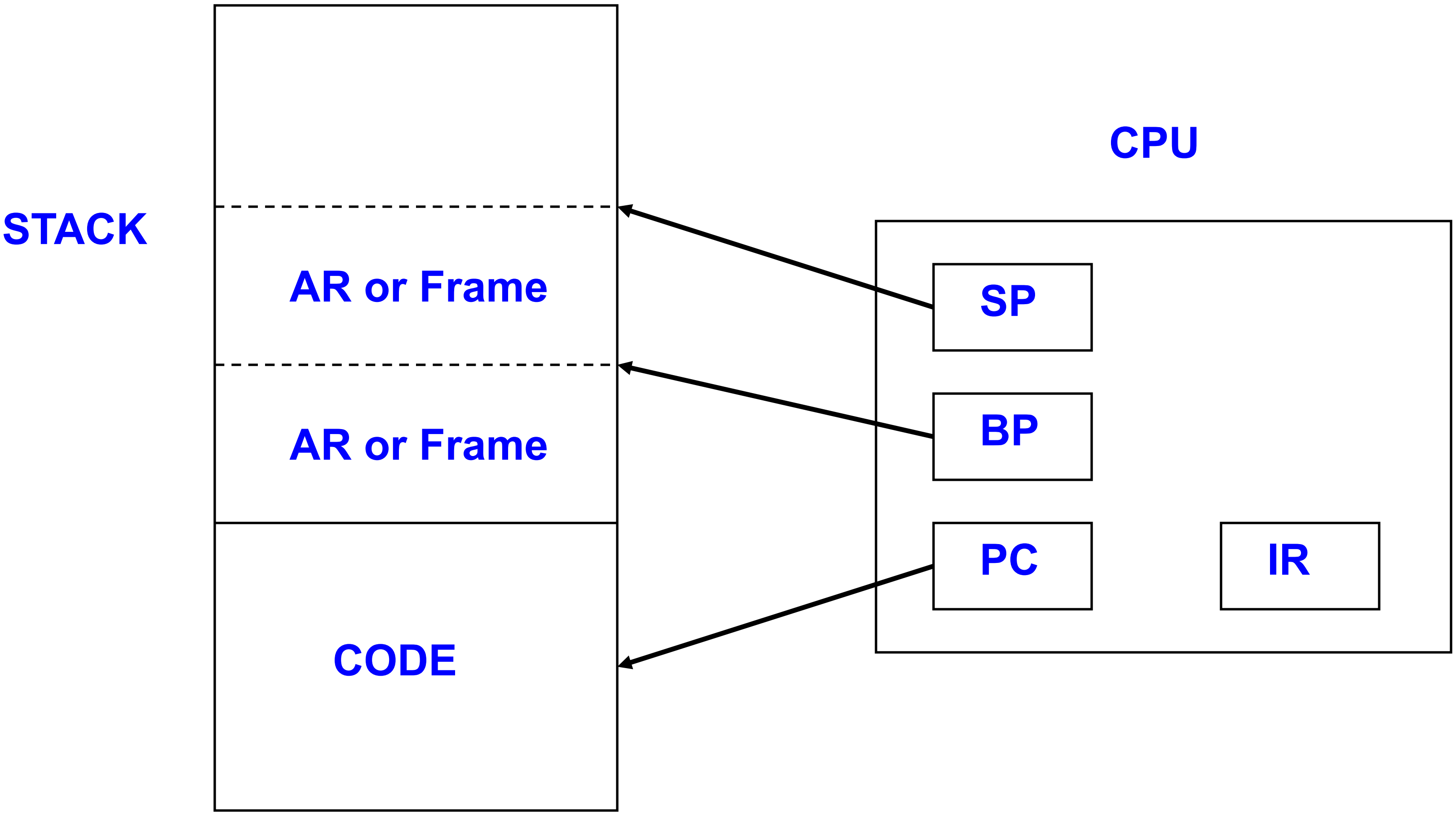
Register **“bp”** points to the base of the current **activation record (AR)** in the stack

Register **“sp”** points to the top of the stack

A program counter or instruction pointer (**pc**)

An instruction register (**ir**).

Virtual Machine: P- code

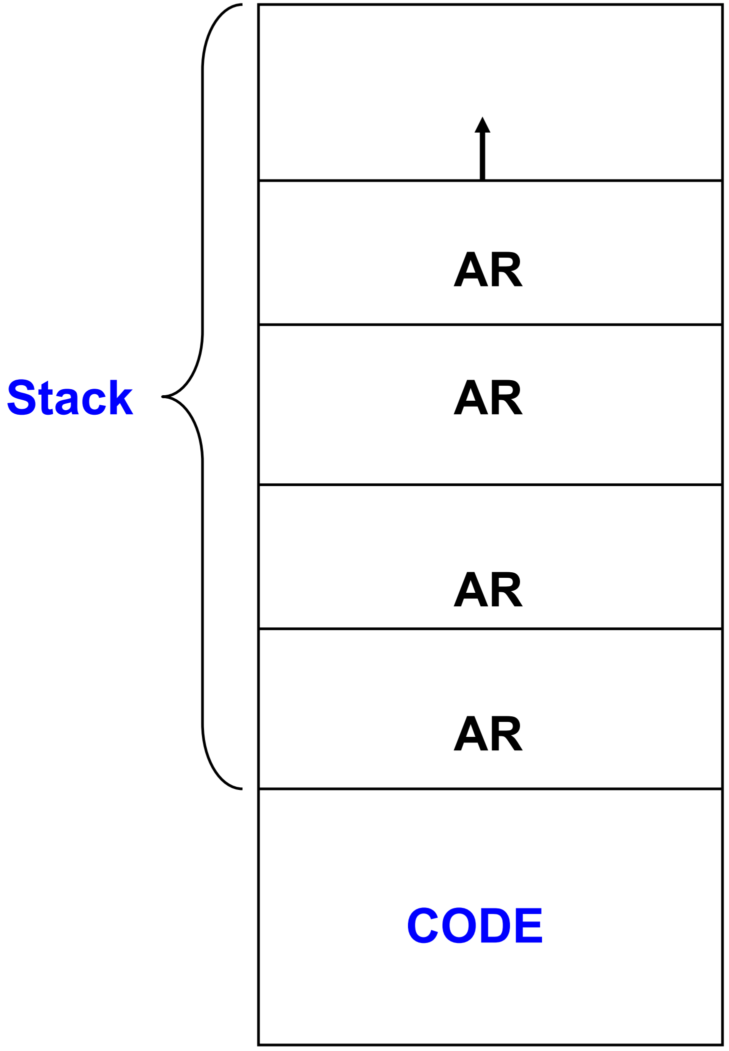


Activation Records (AR)

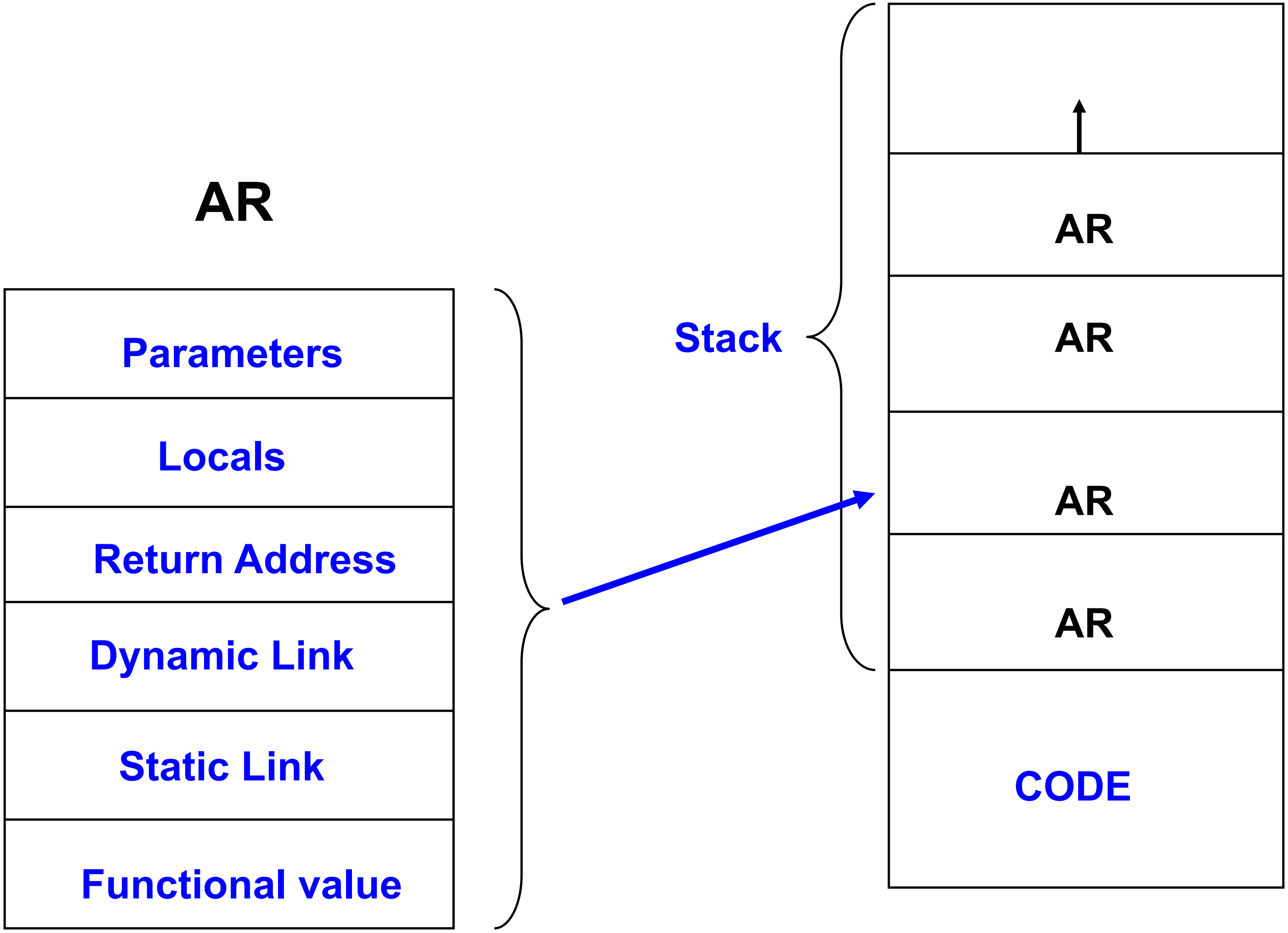
What is an activation record?

•Activation record or stack frame is the name given to a data structure which is inserted in the stack, each time a procedure or function is called.

•The data structure contains information to control sub-routines program execution.



Activation records (AR)



AR

Parameters

Locals

Return Address

Dynamic Link

Static Link

Functional value

Stack

AR

AR

AR

AR

CODE

Activation records (AR)

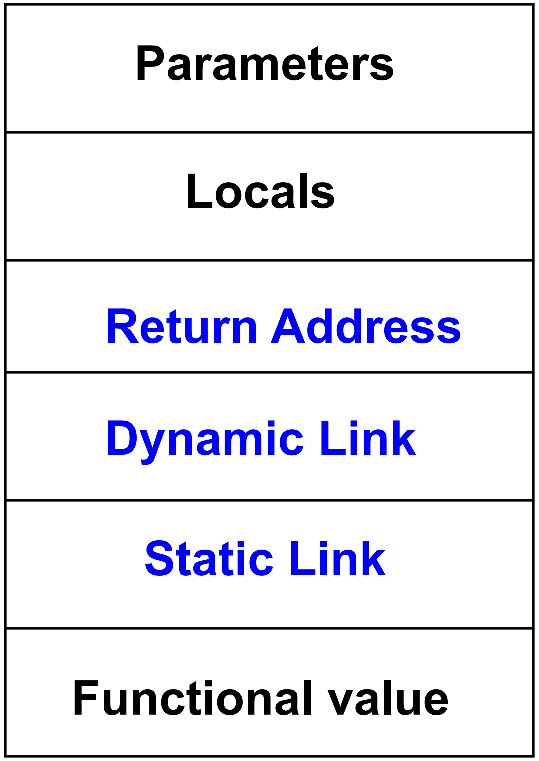
Control Information:

Return Address: Points, in the code segment, to the next instruction to be executed after termination of the current function or procedure.

Dynamic Link: Points to the previous stack frame

Static Link: Points to the stack frame of the procedure that statically encloses the current function or procedure

AR



Activation records (AR)

Functional value: Location to store the function return value.

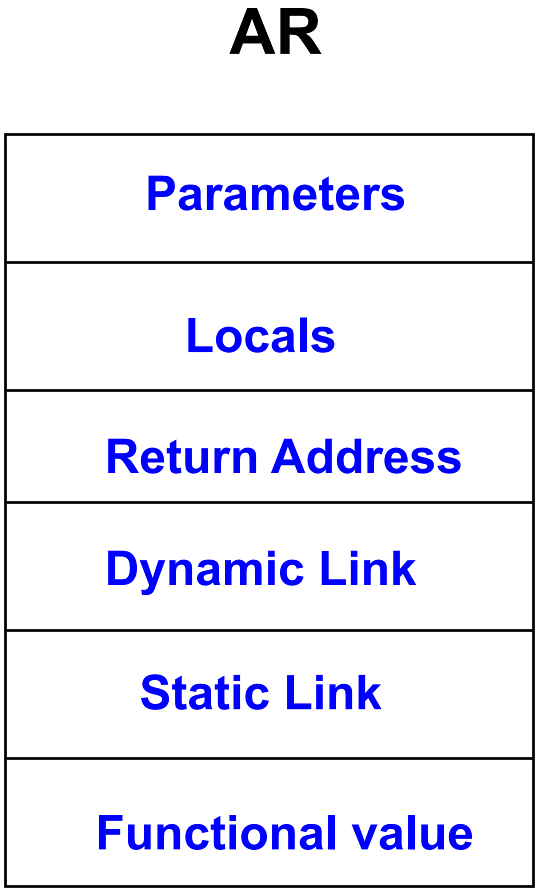
Parameters: Space reserved to store the actual parameters of the function.

Locals: Space reserved to store local variables declared within the procedure.

Return Address: Points, in the code segment, to the next instruction to be executed after termination of the current function or procedure.

Dynamic Link: Points to the previous stack frame

Static Link: Points to the stack frame of the procedure that statically encloses the current function or procedure



AR

Parameters

Locals

Return Address

Dynamic Link

Static Link

Functional value

Back to the P-machine!!
Instruction cycle

The machine has two cycles known as fetch and execute.

Fetch cycle:
In the fetch cycle an instruction is fetch from the code store (ir ← code[pc]) and the program counter is incremented by one (pc ← pc + 1).

Execute cycle:
In this cycle ir.op indicates the operation to be executed. In case ir.op = OPR then the field ir.m is used to identified the operator and execute the appropriate arithmetic or logical instruction

opcode

↓

P-machine ISA

01 - **LIT 0, M** → Push constant value (literal) **M** onto stack

02 – **OPR (to be defined in the next slide)**

03 – **LOD L, M** → Push from location at offset **M** in frame **L** levels down.

04 – **STO L, M** → Store in location at offset **M** in frame **L** levels down.

05 – **CAL L, M** → Call procedure at M (generates new block mark and pc = **M**).

06 – **INC 0, M** → Allocate **M** locals (increment sp by M), first three are **SL**, **DL**, **RA**.

07 – **JMP 0, M** → pc = **M**;

08 – **JPC 0, M** → Jump to M if top of stack element is 0
and decrement sp by one.

09 – **WRT 0, 0** → (print (stack[sp]) and sp ← sp – 1

opcode

↓

P-machine ISA

02 - OPR:

RTN	**0,0**	→ **Return operation** (i.e. return from subroutine)
OPR	**0,1**	→ **NEG** (- stack[sp])
OPR	**0,2**	→ **ADD** (sp← sp – 1 and stack[sp] ← stack[sp] + stack[sp + 1])
OPR	**0,3**	→ **SUB** (sp← sp – 1 and stack[sp] ← stack[sp] - stack[sp + 1])
OPR	**0,4**	→ **MUL** (sp← sp – 1 and stack[sp] ← stack[sp] * stack[sp + 1])
OPR	**0,5**	→ **DIV** (sp← sp – 1 and stack[sp] ← stack[sp] div stack[sp + 1])
OPR	**0,6**	→ **ODD** (stack[sp] ← stack mod 2) or ord(odd(stack[sp]))
OPR	**0,7**	→ **MOD** (sp← sp – 1 and stack[sp] ← stack[sp] mod stack[sp + 1])
OPR	**0,8**	→ **EQL** (sp← sp – 1 and stack[sp] ← stack[sp] = =stack[sp + 1])
OPR	**0,9**	→ **NEQ** (sp← sp – 1 and stack[sp] ← stack[sp] != stack[sp + 1])
OPR	**0,10**	→ **LSS** (sp← sp – 1 and stack[sp] ← stack[sp] < stack[sp + 1])
OPR	**0,11**	→ **LEQ** (sp← sp – 1 and stack[sp] ← stack[sp] <= stack[sp + 1])
OPR	**0,12**	→ **GTR** (sp← sp – 1 and stack[sp] ← stack[sp] > stack[sp + 1])
OPR	**0,13**	→ **GEQ** (sp← sp – 1 and stack[sp] ← stack[sp] >= stack[sp + 1])

P-machine ISA

opcode

↓

```
01 -  LIT    0, M →  sp ← sp +1;
                     stack[sp] ← M;

02 – RTN   0, 0 →   sp ← bp -1;
                    pc ← stack[sp + 3];
                    bp ← stack[sp + 2];

03 – LOD   L, M → sp ← sp +1;
                  stack[sp] ← stack[ base(L) + M];

04 – STO   L, M →  stack[ base(L) + M] ← stack[sp];
                    sp ← sp -1;
```

P-machine ISA

opcode
↓

```
05 - CAL   L, M → stack[sp + 1]  ←  base(L);          /* static link (SL)
                  stack[sp + 2]  ← bp;                /*  dynamic link (DL)
                  stack[sp + 3]  ← pc                 /*  return address (RA)
                  bp ← sp + 1;
                  pc ← M;

06 – INC   0, M → sp ← sp + M;

07 – JMP  0, M →  pc = M;

08 – JPC  0, M →  if stack[sp] == 0 then  pc ← M;
                 sp ← sp - 1;

09 – WRT 0, 0  →  print (stack[sp]);
                  sp ← sp – 1;
```

P-machine: Code generation

In this example “ functional value field” is not considered

Programming example using PL/0

```
const n = 13;          /* constant declaration
var i,h;               /* variable declaration
procedure sub;
  const k = 7;
  var j,h;
  begin                   /* procedure
    j:=n;                 /* declaration
    i:=1;
    h:=k;
  end;
begin  /* main starts here
  i:=3;
  h:=9;
  call sub;
end.
```

P-code for the program on the left

```
 0 jmp 0 10
 1 jmp 0 2
 2 inc 0 5
 3 lit 0 13
 4 sto 0 3
 5 lit 0 1
 6 sto 1 3
 7 lit 0 7
 8 sto 0 4
 9 opr 0 0
10 inc 0 5
11 lit 0 3
12 sto 0 3
13 lit 0 9
14 sto 0 4
15 cal 0 2
16 opr 0 0
```

Running a program on PM/0

	pc	bp	sp	stack
Initial values	0	1	0	0 0 0 0 0
0 jmp 0, 10	10	1	0	0 0 0 0 0
10 inc 0, 5	11	1	5	0 0 0 0 0
11 lit 0, 3	12	1	6	0 0 0 0 0 3
12 sto 0, 3	13	1	5	0 0 0 3 0
13 lit 0, 9	14	1	6	0 0 0 3 0 9
14 sto 0, 4	15	1	5	0 0 0 3 9
15 cal 0, 2	2	6	5	0 0 0 3 9| 1 1 16
2 inc 0, 5	3	6	10	0 0 0 3 9| 1 1 16 0 0
3 lit 0, 13	4	6	11	0 0 0 3 9| 1 1 16 0 0 13
4 sto 0, 3	5	6	10	0 0 0 3 9| 1 1 16 13 0
5 lit 0, 1	6	6	11	0 0 0 3 9| 1 1 16 13 0 1
6 sto 1, 3	7	6	10	0 0 0 1 9| 1 1 16 13 0
7 lit 0, 7	8	6	11	0 0 0 1 9| 1 1 16 13 0 7
8 sto 0, 4	9	6	10	0 0 0 1 9| 1 1 16 13 7
9 opr 0, 0	16	1	5	0 0 0 1 9

code

```
0 jmp  0 10
1 jmp  0 2
2 inc  0 5
3 lit  0 13
4 sto  0 3
5 lit  0 1
6 sto  1 3
7 lit  0 7
8 sto  0 4
9 opr  0 0
10 inc 0 5
11 lit 0 3
12 sto 0 3
13 lit 0 9
14 sto 0 4
15 cal 0 2
16 opr 0 0
```

P-machine: Code generation (Regs)

Programming example using PL/0

```
const n = 13;        /* constant declaration
var i,h;             /* variable declaration
procedure sub;
  const k = 7;
  var j,h;
  begin
    j:=n;              /* procedure
    i:=1;              /* declaration
    h:=k;
  end;
begin  /* main starts here
  i:=3;
  h:=9;
  call sub;
end.
```

P-code for the program on the left

```
0 jmp   0 0 10
1 jmp   0 0 2
2 inc   0 0 6
3 lit   0 0 13
4 sto   0 0 4
5 lit   0 0 1
6 sto   0 1 4
7 lit   0 0 7
8 sto   0 0 5
9 opr   0 0 0
10 inc  0 0 6
11 lit  0 0 3
12 sto  0 0 4
13 lit  0 0 9
14 sto  0 0 5
15 cal  0 0 2
16 sio  0 0 3
```

Program on PM/0 with registers

		pc	bp	sp	stack
Initial values		0	1	0	0 0 0 0 0
0 jmp	0, 0, 10	10	1	0	0 0 0 0 0
10 inc	0, 0, 6	11	1	6	0 0 0 0 0
11 lit	0, 0, 3	12	1	6	0 0 0 0 0
R0 = 3, R1 =0, R2 = 0, etc.					
12 sto	0, 0, 4	13	1	6	0 0 0 0 3
13 lit	0, 0, 9	14	1	6	0 0 0 0 3
14 sto	0, 0, 5	15	1	6	0 0 0 0 3 9
15 cal	0, 0, 2	2	7	6	0 0 0 0 3 9
2 inc	0, 0, 6	3	7	12	0 0 0 0 3 9 | 0 1 1 16
3 lit	0, 0, 13	4	7	12	0 0 0 0 3 9 | 0 1 1 16
4 sto	0, 0, 4	5	7	12	0 0 0 0 3 9 | 0 1 1 16 13
5 lit	0, 0, 1	6	7	12	0 0 0 0 3 9 | 0 1 1 16 13
6 sto	0, 1, 4	7	7	12	0 0 0 0 1 9 | 0 1 1 16 13
7 lit	0, 0, 7	8	7	12	0 0 0 0 1 9 | 0 1 1 16 13
8 sto	0, 0, 5	9	7	12	0 0 0 0 1 9 | 0 1 1 16 13 7
9 rtn	0, 0, 0	16	1	6	0 0 0 0 1 9
16sio	0, 0, 3	17	1	6	0 0 0 0 1 9

code	
0 jmp	0 0 10
1 jmp	0 0 2
2 inc	0 0 6
3 lit	0 0 13
4 sto	0 0 4
5 lit	0 0 1
6 sto	0 1 4
7 lit	0 0 7
8 sto	0 0 5
9 opr	0 0 0
10 inc	0 0 6
11 lit	0 0 3
12 sto	0 0 4
13 lit	0 0 9
14 sto	0 0 5
15 cal	0 0 2
16 sio	0 0 3