System Software: The Virtual Machine

Andrew Harn

University of Central Florida

Virtual Machines

- A machine implemented by software
- Used to run intermediate code
  - Very portable
  - Can track and test ISA
- Made in the 70s, referred as P-code machines (PM) for Pascal compilers
- A more modern example would be the JVM which takes in Java bytecode as the input

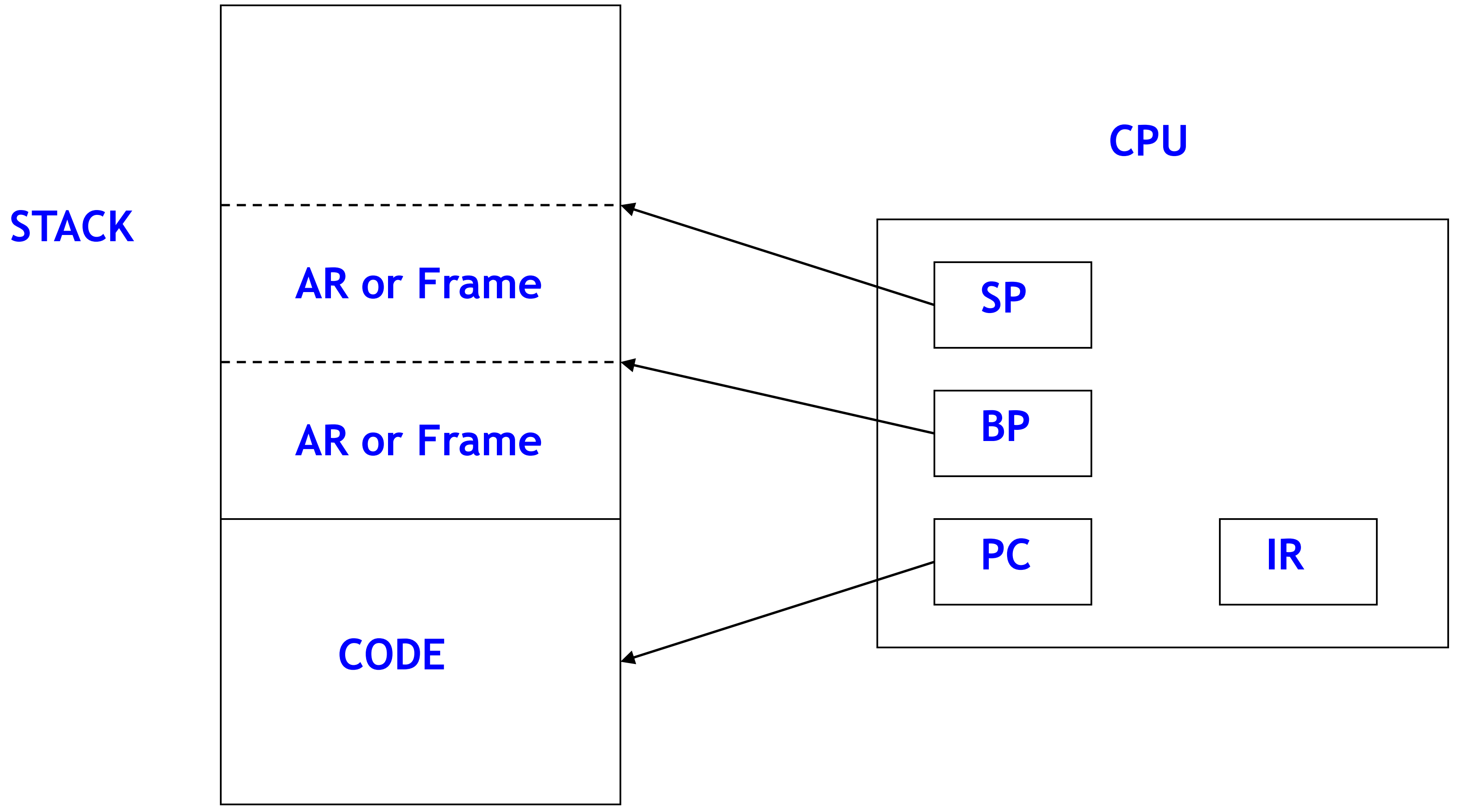
The P-Machine Instruction Format (PM/0)

- The ISA of the PM/0 has 22 instructions, each in the format of:
  - **<OP, L, M>**
  - **OP** – The operation code
  - **L** – The lexicographical level
  - **M** – Depends on the oPCode, it could be:
    - A number (**LIT, INT**)
    - A program address (**JMP, JPC, CAL**)
    - A data address (**LOD, STO**)
    - Identity of the operator **OPR**

The P-Code Machine

- The Interpreter if PM/0 consists of a stack of activation records (AR) on top of the underlying code of the program
- An **activation record (AR)** is simply a data structure used to store data of a sub-routine in order to organize program execution
- In addition to the stack, the machine has a CPU with four registers:
    - **Base Pointer (BP)** points to the base of the current AR in the stack
    - **Stack Pointer (SP)** points to the top of the stack
    - **Program Counter (PC)**
    - **Instruction Register (IR)**

Virtual Machine: With CPU



Parts of the Activation Record (AR)

- **Functional Value** points to the memory location to store the function return value
- **Parameters & Locals** are Space used to store parameters and local variables of the procedure
- **Return Address** points to the line of code for the next instruction to be execution after the termination of the current function/procedure (when the AR is no loner needed)
- **Dynamic Link** points to the previous stack frame
- **Static Link** points to the frame of the procedure that encloses the current procedure
  - **Relevant only when procedure can hold other procedures**

Instruction Cycle

- As with the VN Machine, the P-Machine has the fetch and execute step for the instruction cycle
- **Fetch Step** – An instruction is fetched from the code (IR ← code[PC]), and the program counter is incremented (PC ← PC+1)
- **Execute Step** – In this step, an operation is executed depending on IR.OP. If this value is **OPR** than the IR.M is used to identify what the appropriate arithmetic/logical instruction is to execute.

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 → Allocates **M** locals (add M to SP), 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.

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

P-Machine ISA: OPR

RTN	**0,0**	→ **Return operation** (i.e. return from subroutine)
OPR	**0,1**	→ **NEG** (stack[SP] ← -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] / stack[SP + 1])
OPR	**0,6**	→ **ODD** (stack[SP] ← stack[SP] mod 2)
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

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

```
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;
```

PM/0: Code Generation

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:=0;
  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 STO0 3
13 LIT 0 0
14 STO 0 4
15 CAL 0 2
16 OPR 0 0
```