Lecture 2: PM/0 Virtual Machine
Outline

• Virtual Machines as software interpreters
• P-code: Instruction Set Architecture
• Instruction Format
• Assembly Language
Virtual Machine: P-code

• The Pseudo-code machine is a software (virtual) machine that implements the instruction set architecture of a stack-based 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 ISA of the PM/0 has 24 different instructions.

The instruction format has three components <op, l, m>:

• **op** Operation Code (op or opcode)

• **l** Lexicographical Level (level)

• **m** Modifier indicates depending on op (mnemonic)
  
  o **Number**  LIT, INT
  
  o **Program Address**  JMP, JPC, CAL
  
  o **Data Address**  LOD, STO
  
  o **Identity of the operator**  OPR
Virtual Machine: P-code

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

- **stack** a store organized as a stack
- **code** a store that contains the instructions
- **CPU** with four registers:
  - **bp** points to the base of the current Activation Record (AR) in the stack
  - **sp** points to the top of the stack
  - **pc** program counter or instruction pointer
  - **ir** instruction register
Virtual Machine: P-code

- Stack
- AR
- AR
- Code
- CPU
  - sp
  - bp
  - pc
  - ir
Activation Records (AR)

- **Activation Record** or **Stack Frame**: data structure that push onto stack, each time a procedure/function is called
- AR contains all information necessary to control the execution of the subroutine
Activation Records (AR)

The order of FV, SL, DL, RA is consistent with + 1 in the base function (useful for the implementation of the PM/0 machine).
Activation Records (AR)

Control Information

- **Return Address** points to the next instruction of the **caller** to be executed after returning from the **callee**, that is, the current function/procedure

- **Dynamic Link** points to the base of the previous AR, that is, the AR of the caller

- **Static Link** points to the AR of the procedure/function that statically encloses the callee

Note that the procedure/function that statically encloses the callee is not necessarily the caller.

For instance, A statically encloses B and B calls itself recursively.
Accessing Values in Activation Records

How to compute the base of activation record \( L \) levels down

```c
int base( int level, int b ) {
    while (level > 0) {
        b = stack[ b + 1 ];
        level--;
    }
    return b;
}
```

The order of FV, SL, DL, RA is consistent with the +1 in the base function.
Activation Records (AR)

Control Information

- **Functional Value** is the location storing the return value of the callee
- **Parameters** are the locations storing the parameters of the callee passed by the caller
- **Locals** are the locations storing the local variables declared within the callee
Instruction Cycle

The instruction cycle consists of two steps:

• Fetch Cycle
  • an instruction is fetched from the code store
    \[ ir \leftarrow \text{code}[ \text{pc} ]; \]
  • the program counter is incremented by one
    \[ \text{pc} \leftarrow \text{pc} + 1; \]

• Execute Cycle
  • \text{ir.op} indicates the operation to be executed
  • When the opcode \text{ir.op} equals 02 (OPR) or 09 (SIO), then the modifier \text{ir.m} further identifies the instruction
Informal Description of ISA

03 LOD L M  Get value at offset M in frame L levels down and push it

opcode mnemonic level modifier
## Informal Description

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 LIT 0 M</td>
<td>Push value M onto stack</td>
</tr>
<tr>
<td>02 OPR 0 M</td>
<td>Perform return (RET) or arithmetic/logical operation (ADD, EQL, ...)</td>
</tr>
<tr>
<td>03 LOD L M</td>
<td>Get value at offset M in frame L levels down and push it</td>
</tr>
<tr>
<td>04 STO L M</td>
<td>Pop stack and insert value at offset M in frame L levels down</td>
</tr>
<tr>
<td>05 CAL L M</td>
<td>Call procedure at M (generates new stack frame)</td>
</tr>
<tr>
<td>06 INC 0 M</td>
<td>Allocate M locals on stack</td>
</tr>
<tr>
<td>07 JMP 0 M</td>
<td>Jump to M</td>
</tr>
<tr>
<td>08 JPC 0 M</td>
<td>Pop stack and jump to M if value is equal to 0</td>
</tr>
<tr>
<td>09 SIO 0 0</td>
<td>Pop stack and print out value</td>
</tr>
<tr>
<td>09 SIO 0 1</td>
<td>Read in input from user and push it</td>
</tr>
<tr>
<td>09 SIO 0 2</td>
<td>Halt the machine</td>
</tr>
</tbody>
</table>
Formal Definition of ISA

01 LIT 0 M  Push value M onto stack

    sp ← sp + 1;
    stack[ sp ] ← M;

03 LOD L M  Get value in frame L levels down at offset M and push it

    sp ← sp + 1;
    stack[ sp ] ← stack[ base(L, bp) + M ];

04 STO L M  Pop stack and insert value in frame L levels down at offset M

    stack[ base(L, bp) + M ] ← stack[ sp ];
    sp ← sp - 1;
Call procedure at \( M \) (generates new stack frame)

\[
\begin{align*}
\text{stack}[\ sp + 1 \] & \leftarrow 0; \quad \text{\quad // functional value (FV)} \\
\text{stack}[\ sp + 2 \] & \leftarrow \text{base}( L, \ bp ); \quad \text{\quad // static link (SL)} \\
\text{stack}[\ sp + 3 \] & \leftarrow \ bp; \quad \text{\quad // dynamic link (DL)} \\
\text{stack}[\ sp + 4 \] & \leftarrow \ pc; \quad \text{\quad // return address (RA)}
\end{align*}
\]

\[
\begin{align*}
\text{bp} & \leftarrow \ sp + 1; \\
\text{pc} & \leftarrow M;
\end{align*}
\]

Why + 2 and not + 1?

The order of FV, SL, DL, RA is consistent with the + 1 in the base function.

Allocate \( M \) locals on stack

\[
\text{sp} \leftarrow \ sp + M;
\]
Formal Definition

07 JMP 0 M       Jump to M

pc ← sp + M;

08 JPC 0 M       Pop stack and jump to M if value is equal to 0

if ( stack[ sp ] == 0 ) then { pc ← M; }
sp ← sp - 1;
Recall that when the opcode is equal to 09 (mnemonic SIO), the operation to be executed is further determined by the modifier \( M \):

**0 OUT** Pop stack and print out value

```plaintext
print( stack[ sp ] );
sp <- sp - 1;
```

**1 INP** Read in input from user and push it

```plaintext
sp <- sp + 1;
read( stack[ sp ] );
```

**2 HLT** Halt the machine (your virtual machine stops)

```plaintext
halt;
```
Recall that when the opcode is equal to 02 (mnemonic OPR), the operation to be executed is further determined by the modifier $M$. 

Formal Definition

The operation 02 OPR 0 M
The only operation with no argument

0 RTN Return from function or procedure

\[
\begin{align*}
sp & \leftarrow bp - 1; \\
pc & \leftarrow stack[sp + 4]; \quad // \text{return address (RA)} \\
bp & \leftarrow stack[sp + 3]; \quad // \text{dynamic link (DL)}
\end{align*}
\]
2 OPR 0 M

Operations with one argument

M

1 NEG stack[ sp ] ← - stack[ sp ];

6 ODD stack[ sp ] ← stack[ sp ] mod 2;
Formal Definition

Operations with two arguments:

for all operations below, perform first

\[ \text{sp} \leftarrow \text{sp} - 1; \]

2. ADD \( \text{stack[sp]} \leftarrow \text{stack[sp]} + \text{stack[sp + 1]}; \)
3. SUB \( \text{stack[sp]} \leftarrow \text{stack[sp]} - \text{stack[sp + 1]}; \)
4. MUL \( \text{stack[sp]} \leftarrow \text{stack[sp]} \times \text{stack[sp + 1]}; \)
5. DIV \( \text{stack[sp]} \leftarrow \text{stack[sp]} \div \text{stack[sp + 1]}; \)
6. MOD \( \text{stack[sp]} \leftarrow \text{stack[sp]} \mod \text{stack[sp + 1]}; \)
8. EQL \( \text{stack[sp]} \leftarrow \text{stack[sp]} == \text{stack[sp + 1]}; \)
9. NEQ \( \text{stack[sp]} \leftarrow \text{stack[sp]} != \text{stack[sp + 1]}; \)
10. LSS \( \text{stack[sp]} \leftarrow \text{stack[sp]} < \text{stack[sp + 1]}; \)
11. LEQ \( \text{stack[sp]} \leftarrow \text{stack[sp]} \leq \text{stack[sp + 1]}; \)
12. GTR \( \text{stack[sp]} \leftarrow \text{stack[sp]} > \text{stack[sp + 1]}; \)
13. GEQ \( \text{stack[sp]} \leftarrow \text{stack[sp]} \geq \text{stack[sp + 1]}; \)
Programming example using PL/0

```plaintext
const n = 13;  
var i,h;       
procedure sub;  
    const k = 7;  
    var j,h;  
    begin  
        j:=n;  
        i:=1;  
        h:=k;  
    end;  
begin  
    i:=3;  
    h:=0;  
    call sub;  
end;
```

P-code

<table>
<thead>
<tr>
<th>Line</th>
<th>OP</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>jmp</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>jmp</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>inc</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>lit</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>sto</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>lit</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>sto</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>lit</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>sto</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>opr</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>inc</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>lit</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>sto</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>lit</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>sto</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>cal</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>sio</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
## Running a program on PM/0

<table>
<thead>
<tr>
<th>Initial values</th>
<th>pc</th>
<th>bp</th>
<th>sp</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  jmp 10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10 inc 6</td>
<td>11</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>11 lit 3</td>
<td>12</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>12 sto 4</td>
<td>13</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>13 lit 0</td>
<td>14</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>14 sto 5</td>
<td>15</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>15 cal 2</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2 inc 6</td>
<td>3</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3 lit 13</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4 sto 4</td>
<td>5</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5 lit 1</td>
<td>6</td>
<td>7</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>6 sto 4</td>
<td>7</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>7 lit 7</td>
<td>8</td>
<td>7</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>8 sto 5</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9 opr 0</td>
<td>16</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>16 sio 2</td>
<td>17</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>