Principles of Computers
17th Lecture

Pavel Ježek, Ph.D.
pavel.jezek@d3s.mff.cuni.cz

http://d3s.mff.cuni.cz/~jezek
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

### CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegStackTop</td>
<td>Flags</td>
<td>(SP)</td>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Instruction set:

- LOAD x
- STORE x
- ADD
- SUB
- MUL
- AND
- OR
- NOT
- SHR
- SHL
- ...  
- PUSH (store on memory [call] stack)
- POP (load from memory [call] stack)
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

| ?   | R15 |
| ?   | R14 |
| ?   | R13 |
| ?   | R12 |
| ?   | R11 |
| ?   | R10 |
| ?   | R9  |
| ?   | R8  |
| ?   | R7  |
| ?   | R6  |
| ?   | R5  |
| ?   | R4  |
| ?   | R3  |
| ?   | R2  |
| ?   | R1  |
| ?   | R0  |

RegStackTop
Flags
(SP)
IP

memory

Disassembled machine code

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-nary operation takes n registers from top)

**CPU registers**

```
<table>
<thead>
<tr>
<th>RegStackTop</th>
<th>Flags</th>
<th>(SP)</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>R15</td>
<td></td>
<td>IP</td>
</tr>
<tr>
<td>?</td>
<td>R14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Disassembled machine code**

```
LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x
```
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x

RegStackTop
Flags (SP)
IP

memory
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

```
<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Flags:
- RegStackTop
- (SP)
- IP

Memory:
- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

**CPU registers**

```
| Stack Top | R15 | R14 | R13 | R12 | R11 | R10 | R9  | R8  | R7  | R6  | R5  | R4  | R3  | R2  | R1  | R0  |
```

**IP**

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x

```
RegStackTop
Flags
(SP)
IP
```
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

Memory

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x

RegStackTop
Flags
(SP)
IP
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-nary operation takes n registers from top)

CPU registers

- ?
- R15
- R14
- R13
- R12
- R11
- R10
- R9
- R8
- R7
- R6
- R5
- R4
- R3
- R2
- R1
- R0
- RegStackTop
- Flags
- (SP)
- IP

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x

memory
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
</table>

RegStackTop: 1
Flags: 1
(SP): 6
IP: 4

memory

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-nary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RegStackTop
Flags
(SP)
IP

memory

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD 2</td>
</tr>
<tr>
<td>LOAD 4</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>LOAD 1</td>
</tr>
<tr>
<td>LOAD 1</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>MUL</td>
</tr>
<tr>
<td>LOAD $F7</td>
</tr>
<tr>
<td>NOT</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>STORE x</td>
</tr>
</tbody>
</table>

RegStackTop

Flags (SP)

IP

memory
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

- R15
- R14
- R13
- R12
- R11
- R10
- R9
- R8
- R7
- R6
- R5
- R4
- R3
- R2
- $F7
- 2
- 1
- 0

RegStackTop
Flags
(SP)
IP
IP

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x

memory
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th>RegStackTop</th>
<th>Flags</th>
<th>(SP)</th>
<th>IP</th>
</tr>
</thead>
</table>

memory

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- LOAD 2
- LOAD 4
- ADD
- LOAD 1
- LOAD 1
- ADD
- LOAD $F7
- NOT
- MUL
- ADD
- STORE x

memory
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RegStackTop
Flags
(SP)
IP

memory

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-nary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>16</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

RegStackTop
Flags
(SP)
IP

memory

• LOAD 2
• LOAD 4
• ADD
• LOAD 1
• LOAD 1
• ADD
• LOAD $F7
• NOT
• MUL
• ADD
• STORE x

• Load/store architecture
• Loads have implicit target (register on top)
• Stores have implicit source (register on top)
• Arithmetic instructions have all operands implicit (n-nary operation takes n registers from top)
### Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

```plaintext
<table>
<thead>
<tr>
<th>CPU registers</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>R15</td>
<td>LOAD 2</td>
</tr>
<tr>
<td>R14</td>
<td>LOAD 4</td>
</tr>
<tr>
<td>R13</td>
<td>ADD</td>
</tr>
<tr>
<td>R12</td>
<td>LOAD 1</td>
</tr>
<tr>
<td>R11</td>
<td>LOAD 1</td>
</tr>
<tr>
<td>R10</td>
<td>ADD</td>
</tr>
<tr>
<td>R9</td>
<td>LOAD $F7</td>
</tr>
<tr>
<td>R8</td>
<td>NOT</td>
</tr>
<tr>
<td>R7</td>
<td>MUL</td>
</tr>
<tr>
<td>R6</td>
<td>ADD</td>
</tr>
<tr>
<td>R5</td>
<td>STORE x</td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td></td>
</tr>
</tbody>
</table>
```

- IP
- RegStackTop
- Flags
- (SP)
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th>IP</th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>8</th>
<th>16</th>
<th>22</th>
</tr>
</thead>
</table>

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x

global data: ...

memory

address x
Stack Machine

- Load/store architecture
- Loads have implicit target (register on top)
- Stores have implicit source (register on top)
- Arithmetic instructions have all operands implicit (n-ary operation takes n registers from top)

CPU registers

<table>
<thead>
<tr>
<th></th>
<th>R15</th>
<th>R14</th>
<th>R13</th>
<th>R12</th>
<th>R11</th>
<th>R10</th>
<th>R9</th>
<th>R8</th>
<th>R7</th>
<th>R6</th>
<th>R5</th>
<th>R4</th>
<th>R3</th>
<th>R2</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegStackTop</td>
<td>Flags</td>
<td>(SP)</td>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LOAD 2
LOAD 4
ADD
LOAD 1
LOAD 1
ADD
LOAD $F7
NOT
MUL
ADD
STORE x

global data:

address x

memory
How To Write/Compile Complex Expressions for Stack Machine?

\[ x := 1 + a + F1(b + c) \times F2(d + e + F3(2, 3), f + F4(4) + g) + F5(5) + x; \]
Complex Expressions in Stack Machine’s Assembler (”.NET CIL Code Here”)

\[ x := 1 + a + F1(b + c) \times F2(d + e + F3(2, 3), f + F4(4) + g) + F5(5) + x; \]

- Instruction reference:
  - `ldc.i4.X` = load constant value \( X \) as 4 byte integer = immediate
  - `ldsfld a` = load from static field \( a \) = load from global variable \( a \)
  - `stsfld x` = store to static field \( x \) = store to global variable \( x \)

- Arguments to functions passed on top of register stack
- Return values from functions stored on top of register stack
Complex Expressions in Stack Machine’s Assembler (.NET CIL Code Here)

\[ x := 1 + a + F1(b + c) \times F2(d + e + F3(2, 3), f + F4(4) + g) + F5(5) + x; \]

- Instruction reference:
  - `ldc.i4.X` = load constant value \( X \) as 4 byte integer = immediate
  - `ldsfld a` = load from static field \( a \) = load from global variable \( a \)
  - `stsfld x` = store to static field \( x \) = store to global variable \( x \)

- Arguments to functions passed on top of register stack
- Return values from functions stored on top of register stack

**Pros (advantages):**
- Easy to write by humans
- Easy to read by cross compilers
  \( \rightarrow \) implemented by virtual machines (.NET’s CLR, JVM)
  \( \rightarrow \) allows easier implementation of JIT compiler from IL code (CIL / Java bytecode) to target CPU’s machine code (e.g. x86)

**Cons (disadvantages):**
- Hard to generate effective code for modern CPUs (modern compilers are be able to generate more effective code for general register architecture CPUs)
  \( \rightarrow \) not implemented by typical real CPUs