Chapter 5 - A Closer Look at Instruction Set Architectures

ISA: The programmers view of the machine.
(The set of programmer visible registers, instructions and memory space.)

Organization: How the ISA is actually implemented in hardware.

Instruction set Characteristics / Classification

• Where Operands can be stored
• How Instructions deal with memory
• Number of Operands per instruction
• Instruction Length
• Operation Types
• Addressing Modes
Instruction set Characteristics / Classification

- Where Operands can be stored
  - Memory (Explicit address or Stack)
  - Registers
  - In the Instruction (Immediate)

Memory

Add X

AC ← AC + M[X]

0011 0000 0001 0110

Memory

Registers

AddR R1

AC ← AC + R1

1010 0000 0000 0001

Immediate

Addl 0x78

AC ← AC + 0x78

1011 0000 0111 1000

Registers
**Instruction set Characteristics / Classification**

- Where Operands can be stored
  - Memory (Explicit address or Stack)
  - Registers
  - In the Instruction (Immediate)

<table>
<thead>
<tr>
<th>Memory</th>
<th>Registers</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add X (or Add 0x016)</td>
<td>AddR R1 (or Add R1)</td>
<td>Addl 0x78 (or Add #0x078)</td>
</tr>
<tr>
<td>AC ← AC + M[X]</td>
<td>AC ← AC + R1</td>
<td>AC ← AC + 0x78</td>
</tr>
<tr>
<td>0011 0000 0001 0110</td>
<td>1010 0000 0000 0001</td>
<td>1011 0000 0111 1000</td>
</tr>
</tbody>
</table>

- Memory
- Registers
- Immediate
Instruction set Characteristics / Classification

- How instructions deal with memory
  The maximum number of memory references allowed per instruction and what instructions are allowed to reference memory.
  - RMW (Read Modify Write memory) instructions
  - Load / Store Architecture

RMW

<table>
<thead>
<tr>
<th>Inc X</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[X] ← M[X] + 1</td>
</tr>
<tr>
<td>1100 0000 0001 0110</td>
</tr>
</tbody>
</table>

Load / Store

Only Load and Store instructions are allowed to reference memory.
(Load data from memory into registers and manipulate it there.
Store back to memory when done)

| Load R1, X                   |
| Add R1, #1                  |
| Store X, R1                 |

(MARIE is not Load / Store)
Instruction set Characteristics / Classification

- Number of operands per instruction

Figure 5.2 Two Possibilities for a 16-Bit Instruction Format
Instruction set Characteristics / Classification

- Number of operands per instruction

![Diagram showing two possibilities for a 16-bit instruction format]

Add R0, R4, R7
R0 ← R4 + R7

Figure 5.2 Two Possibilities for a 16-Bit Instruction Format
Number of Operands

\[ Z = X \times Y + W \times U \]

on a one address ISA (i.e. Marie):

```
LOAD  X  ; AC \leftarrow M[X]
MULT  Y  ; AC \leftarrow AC \times M[Y]
STORE TEMP
LOAD  W
MULT  U
ADD   TEMP
STORE Z
```

on a two address ISA:

```
src1/dst src2
LOAD  R1, X  ; R1 \leftarrow M[X]
MULT  R1, Y  ; R1 \leftarrow R1 \times M[Y]
LOAD  R2, W
MULT  R2, U
ADD   R1, R2
STORE Z, R1
```

on a three address ISA:

```
dst src1 src2
MULT  R1, X, Y  ; R1 \leftarrow M[X] \times M[Y]
MULT  R2, W, U
ADD   Z, R1, R2
```

dst \leftarrow src1 op src2
important for operations where the order matters
i.e. subtract and divide
Number of Operands

\[ Z = X \times Y + W \times U \]

on a one address ISA (i.e. Marie):

\[
\begin{align*}
\text{LOAD} & \quad X ; \ AC \leftarrow \text{M}[X] \\
\text{MULT} & \quad Y ; \ AC \leftarrow AC \times \text{M}[Y] \\
\text{STORE} & \quad \text{TEMP} \\
\text{LOAD} & \quad W \\
\text{MULT} & \quad U \\
\text{ADD} & \quad \text{TEMP} \\
\text{STORE} & \quad Z
\end{align*}
\]

7 instrs, one temp mem loc
7 instr fetch + 7 mem data refs

on a two address ISA:

\[
\begin{align*}
\text{LOAD} & \quad R1, X ; \ R1 \leftarrow \text{M}[X] \\
\text{MULT} & \quad R1, Y ; \ R1 \leftarrow R1 \times \text{M}[Y] \\
\text{LOAD} & \quad R2, W \\
\text{MULT} & \quad R2, U \\
\text{ADD} & \quad R1, R2 \\
\text{STORE} & \quad Z, R1
\end{align*}
\]

6 instrs, 2 registers
6 instr fetch + 5 mem data refs

on a three address ISA:

\[
\begin{align*}
\text{MULT} & \quad R1, X, Y ; \ R1 \leftarrow \text{M}[X] \times \text{M}[Y] \\
\text{MULT} & \quad R2, W, U \\
\text{ADD} & \quad Z, R1, R2
\end{align*}
\]

3 instrs, 2 registers
3 instr fetch + 5 mem data refs

dst ← src1 op src2
important for operations where the order matters
i.e. subtract and divide
Number of Operands

\[ Z = X \times Y + W \times U \]

on a one address ISA (i.e. Marie):

```
LOAD X ; AC ← M[X]
MULT Y ; AC ← AC × M[Y]
STORE TEMP ; Z
LOAD W
MULT U
ADD TEMP ; Z
STORE Z
```

7 instrs, one temp mem loc
7 instr fetch + 7 mem data refs

Accumulator Architecture

on a two address ISA:

```
LOAD R1, X ; R1 ← M[X]
MULT R1, Y ; R1 ← R1 × M[Y]
LOAD R2, W
MULT R2, U
ADD R1, R2
STORE Z, R1
```

6 instrs, 2 registers
6 instr fetch + 5 mem data refs

Register - Memory Architecture

on a three address ISA:

```
MULT R1, X, Y ; R1 ← M[X] × M[Y]
MULT R2, W, U
ADD Z, R1, R2
```

3 instrs, 2 registers
3 instr fetch + 5 mem data refs

dst ← src1 op src2
important for operations where the order matters
i.e. subtract and divide

Memory - Memory Architectures have instructions that have all operands in memory and are RMW.
Number of Operands

\[ Z = X \times Y + W \times U \]

on a one address ISA (i.e. Marie) :

Accumulator Architecture

```
LOAD X ; AC \leftarrow M[X]
MULT Y ; AC \leftarrow AC \times M[Y]
STORE TEMP
LOAD W
MULT U
ADD TEMP
STORE Z
```

7 instrs, one temp mem loc
7 instr fetch + 7 mem data refs

on a two address ISA:  

Register - Memory Architecture

```
LOAD R1, X ; R1 \leftarrow M[X]
MULT R1, Y ; R1 \leftarrow R1 \times M[Y]
LOAD R2, W
MULT R2, U
ADD R1, R2
STORE Z, R1
```

6 instrs, 2 registers
6 instr fetch + 5 mem data refs

on a three address ISA:  

```
MULT R1, X, Y ; R1 \leftarrow M[X] \times M[Y]
MULT R2, W, U
ADD Z, R1, R2
```

3 instrs, 2 registers
3 instr fetch + 5 mem data refs

on a three address Load/Store ISA:

```
LOAD R1, X
LOAD R2, Y
MULT R1, R1, R2
LOAD R2, W
LOAD R3, U
MULT R2, R2, R3
ADD R1, R1, R2
STORE Z, R1
```

8 instrs, 3 registers
8 instr fetch + 5 mem data refs

important for operations where the order matters
i.e. subtract and divide

Memory - Memory Architectures have instructions that have all operands in memory and are RMW.
“Zero” Address Instructions
Stack Architectures

\[ Z = X + Y \] is in “infix” notation
\[ Z = + X Y \] is in “prefix” notation
\[ Z = X Y + \] is in “postfix” notation

Postfix notation is also known as RPN after the polish mathematician Jan Lukasiewicz.

Stack architectures are better represented using postfix notation.

\[ Z = (X \times Y) + (W \times U) \]

becomes

\[ Z = X Y \times W U \times + \]

in postfix notation.

Do parenthesized operations, high precedence operations first
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \times Y \times W \times U \times + \]
on a stack architecture

PUSH X

<table>
<thead>
<tr>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
“Zero” Address Instructions

Stack Architectures

Implementing

\[ Z = X \times Y + W U \times + \]
on a stack architecture

PUSH X
PUSH Y

\[
\begin{array}{c}
Y \\
X \\
\vdots
\end{array}
\]
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \, Y \times \, W \, U \times \, + \]
on a stack architecture

PUSH X
PUSH Y
MULT

\[ X \times Y \]
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \times Y \times W U \times + \]
on a stack architecture

<table>
<thead>
<tr>
<th>PUSH X</th>
<th>( X \times Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH Y</td>
<td></td>
</tr>
<tr>
<td>MULT</td>
<td></td>
</tr>
</tbody>
</table>

\[ : \]
Implementing
\[ Z = X \times Y \times W U \times + \]
on a stack architecture

```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
```

```
U
W
X \times Y

```

```
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \times Y \times W \times U + \]
on a stack architecture

PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \times Y \times W \times U \times + \]
on a stack architecture

```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
......
```

\[ W \times U \]
\[ X \times Y \]
“Zero” Address Instructions
Stack Architectures

Implementing

\[ Z = X \times Y \times W \times U \times + \]
on a stack architecture

```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
ADD
```

\[ X \times Y + W \times U \]
“Zero” Address Instructions
Stack Architectures

Implementing
\[ Z = X \times Y \times W \times U \times + \]
on a stack architecture

- PUSH X
- PUSH Y
- MULT
- PUSH W
- PUSH U
- MULT
- ADD
- POP Z

::
Instruction set Characteristics / Classification

- Instruction Length
  - Long vs Short
    Short instructions use less memory, but limit the number of operands that can be specified, needing more instructions.
  - Fixed vs Variable Length Instructions / Opcodes
    Fixed length are easy to decode, but they may waste space

Figure 5.2 Two Possibilities for a 16-Bit Instruction Format
**Instruction set Characteristics / Classification**

- **Instruction Length**
  - Long vs Short
    Short instructions use less memory, but limit the number of operands that can be specified, needing more instructions.
  - Fixed vs Variable Length Instructions / Opcodes
    Fixed length are easy to decode, but they may waste space

*Figure 5.2 Two Possibilities for a 16-Bit Instruction Format*

*Figure 5.3 Three More Possibilities for a 16-Bit Instruction Format*
Variable Length Instructions / Opcodes (Expanding Opcodes)

- We must be able to determine the number of bits in the Instruction / Opcode from the msbs in the opcode

**Example 5.8** We wish to encode (create an instruction set) that has 15 3-address instructions, 14 2-address instructions, 31 1-address instructions and 16 0-address instructions using only 16 bit instructions. **Note: we are assuming 4-bit register select fields.**

<table>
<thead>
<tr>
<th>0</th>
<th>0000</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
</tbody>
</table>

- 15 3-address codes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 14 2-address codes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 31 1-address codes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 16 0-address codes
Variable Length Instructions / Opcodes (Expanding Opcodes)

- We must be able to determine the number of bits in the Instruction / Opcode from the msbs in the opcode

Example 5.8 We wish to encode (create an instruction set) that has 15 3-address instructions, 14 2-address instructions, 31 1-address instructions and 16 0-address instructions using only 16 bit instructions.

Note: we are assuming 4-bit register select fields.
Variable Length Instructions / Opcodes (Expanding Opcodes)

- We must be able to determine the number of bits in the Instruction / Opcode from the msbs in the opcode

**Example 5.8** We wish to encode (create an instruction set) that has 15 3-address instructions, 14 2-address instructions, 31 1-address instructions and 16 0-address instructions using only 16 bit instructions. Note: we are assuming 4-bit register select fields.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15 3-address codes

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>1111</td>
<td>0000</td>
<td>R1</td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>1111</td>
<td>1101</td>
<td>R1</td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14 2-address codes

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4064</td>
<td>1111</td>
<td>1110</td>
<td>0000</td>
<td>R1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4094</td>
<td>1111</td>
<td>1111</td>
<td>1110</td>
<td>R1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

31 1-address codes

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>65520</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65535</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16 0-address codes