INSTRUCTION SET ARCHITECTURE

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Computer Organization

- Classic components of a computing system
Computer Organization

- Classic components of a computing system
Computer Organization

- Classic components of a computing system
Computer Organization

- Classic components of a computing system
Computer Organization

- Classic components of a computing system

- Algorithm

- Output
  - Monitor
  - Projector

- CPU
  - Intel Core i7

- Input
  - Keyboard
  - Mouse
  - Joystick
  - Microphone

- Memory
  - RAM
Classic components of a computing system

- CPU
- Memory
- Input
- Output

Algorithm
Instruction Set Architecture

- The key to program/use a microprocessor
  - The language of the hardware defines the hardware/software interface
  - ISA is a contract between software and hardware
Instruction Set Architecture

- The key to program/use a microprocessor
  - The language of the hardware defines the hardware/software interface
  - ISA is a contract between software and hardware
  - Stored-program concept (von Neumann)
Instruction Set Architecture

- A program (in say, C) is compiled into an executable that is composed of machine instructions.
- Java programs are converted into portable bytecode that is converted into machine instructions during execution (just-in-time compilation).
A program (in say, C) is compiled into an executable that is composed of machine instructions

Java programs are converted into portable bytecode that is converted into machine instructions during execution (just-in-time compilation)
Data Representation

- Smallest unit of representing information in conventional computers is **bit**
  - Only two states: 0 and 1
Data Representation

- Smallest unit of representing information in conventional computers is **bit**
  - Only two states: 0 and 1
- Multibit representation units are used to increase the number of states
  - Every group of 8 bits is called a **byte** representing 256 states
Data Representation

- Smallest unit of representing information in conventional computers is bit
  - Only two states: 0 and 1

- Multibit representation units are used to increase the number of states
  - Every group of 8 bits is called a byte representing 256 states
  - Multiple bytes form a word
    - 4-byte word or
    - 8-byte word in more modern processors
Data Conversion

- Decimal is the most human-friendly base for presenting numbers
  - Example: 8163

- Convert decimal to binary (machine-friendly)
  - Through a series of divisions
  - Example: 1111111100011
Data Conversion

Decimal is the most human-friendly base for presenting numbers
- Example: 8163

Convert decimal to binary (machine-friendly)
- Through a series of divisions
- Example: 1111111100011

Find the binary representation of \(8163\) through a series of divisions by 2.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>4081</th>
<th>2040</th>
<th>1020</th>
<th>510</th>
<th>255</th>
<th>127</th>
<th>63</th>
<th>31</th>
<th>15</th>
<th>7</th>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remainder</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Answer: \(1111111100011\) \(\text{bin}\)
Data Conversion

- Decimal to Hexadecimal
  - Example: 8163

Find the hexadecimal representation of 8163 through a series of divisions by 16.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Answer: \(1FE3_{\text{hex}}\)
Data Conversion

- Decimal to Octal
  - Example: 8163

Find the hexadecimal representation of $8163$ through a series of divisions by 8.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1020</td>
<td>3</td>
</tr>
<tr>
<td>127</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Answer: $17743_{\text{oct}}$
Conversion To Decimal

- From Binary (1111111100011)

\[ 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 + 1 \times 2^8 + 1 \times 2^9 + 1 \times 2^{10} + 1 \times 2^{11} + 1 \times 2^{12} = 8163 \]
Conversion To Decimal

- From Binary (1111111100011)
  \[ 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 + 1 \times 2^8 + 1 \times 2^9 + 1 \times 2^{10} + 1 \times 2^{11} + 1 \times 2^{12} = 8163 \]

- From Hexadecimal (1FE3)
  \[ 3 \times 16^0 + E \times 16^1 + F \times 16^2 + 1 \times 16^3 = 3 \times 16^0 + 14 \times 16^1 + 15 \times 16^2 + 1 \times 16^3 = 8163 \]
Conversion To Decimal

- **From Binary (1111111100011)**
  \[1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 + 1 \times 2^8 + 1 \times 2^9 + 1 \times 2^{10} + 1 \times 2^{11} + 1 \times 2^{12} = 8163\]

- **From Hexadecimal (1FE3)**
  \[3 \times 16^0 + E \times 16^1 + F \times 16^2 + 1 \times 16^3 = 3 \times 16^0 + 14 \times 16^1 + 15 \times 16^2 + 1 \times 16^3 = 8163\]

- **From Octal (17743)**
  \[3 \times 8^0 + 4 \times 8^1 + 7 \times 8^2 + 7 \times 8^3 + 1 \times 8^4 = 8163\]
Instruction Set Architecture

- keep the hardware simple — the chip must only implement basic primitives and run fast

- keep the instructions regular — simplifies the decoding/scheduling of instructions

- MIPS instruction set architecture
  - Other examples are ARM, x86, IBM power, etc.

- Complex vs. simple instructions
  - Which one is better?
Example MIPS Instruction

- **C code**
  - High level language
  
  ```c
  a = b + c;
  ```

- **Assembly code**
  - Human friendly machine instruction
  
  ```asm
  add a, b, c  # a is the sum of b and c
  ```

- **Machine code**
  - Hardware friendly machine instruction
  
  `0000001000110010010000000100000`
Translate the following C code to assembly

\[ a = b + c + d + e; \]
Translate the following C code to assembly

```
a = b + c + d + e;
```

Assembly

```
add a, b, c
add a, a, d
add a, a, e
add a, b, c
add f, d, e
add a, a, f
```
Translate the following C code to assembly

```
a = b + c + d + e;
```

Assembly

```
add  a, b, c
add  a, a, d
add  a, a, e
add  a, b, c
add  f, d, e
add  a, a, f
```

Translate this one

```
f = (g + h) – (i + j);
```
Translate this one

\[
f = (g + h) - (i + j);
\]

Assembly

```
add f, g, h
sub f, f, i
sub f, f, j
```

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

In summary

- Operations are not necessarily associative and commutative
- More instructions than C statements
- Usually fixed number of operands per instruction