C335 Computer Structures

Computer Abstractions and Technology (Part #1)

Dr. David R. Surma

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann, Dr. Liqiang Zhang and other sources
Computer Abstractions and Technology

- What is computer architecture?
- What forces drive computer architecture?
- Performance
Below the Program

Applications software

Hardware

Systems software

- Operating system – supervising program that interfaces the user’s program with the hardware (e.g., Linux, Mac OS, Windows)
  - Handles basic input and output operations
  - Allocates storage and memory
  - Provides for protected sharing among multiple applications
- Compiler – translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute
Below the Program, Con’t

- High-level language program

Compiler – translates programs written in a high-level language (e.g., C/C++, Java) into instructions that the hardware can execute.
Below the Program, Con’t

- High-level language program (in C)
  ```c
  swap (int v[], int k)
  {
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
  }
  ```

- Assembly language program (for MIPS)
  ```assembly
  swap:    sll $2, $5, 2
          add $2, $4, $2
          lw $15, 0($2)
          lw $16, 4($2)
          sw $16, 0($2)
          sw $15, 4($2)
          jr $31
  ```

- Machine (object, binary) code (for MIPS)
  ```
  000000 00000 00101 0001000010000000
  000000 00100 00010 0001000000100000
  ```
Advantages of Higher-Level Languages?

- Higher-level languages …
  - Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …)
  - Improves programmer productivity – more understandable code that is easier to debug and validate
  - Improves program maintainability
  - Allows programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
  - Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine

- As a result, very little programming is done today at the assembler level
Why bother to learn assembly language?

- “The difference between mediocre and star programmers is that star programmers understand assembly language, whether or not they use it on a daily basis.”

- “Assembly language is the language of the computer itself. To be a programmer without ever learning assembly language is like being a professional race car driver without understanding how your carburetor works. *To be a truly successful programmer, you have to understand exactly what the computer sees when it is running a program.* Nothing short of learning assembly language will do that for you. Assembly language is often seen as a black art among today's programmers - with those knowing this art being more productive, more knowledgeable, and better paid, even if they primarily work in other languages.” – Jonathan Bartlett
**QUESTION:** In Spring 2001, tens of thousands of dot.com workers were laid off.

- How many of them were making car payments on a Jaguar?
- How many of them knew assembly language?

**ANSWER:**

- How many of them were making car payments on a Jaguar?  
  - Many of them.
- How many of them knew assembly language?  
  - Just a few of them.

The used car lots of Silicon Valley were full of repossessed Jaguars (according to a news story in 2001).
Under the Covers

The BIG Picture

- Same components for all kinds of computers
  - Desktop, server, embedded

- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

datapath + control = processor (CPU)
Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously
LCD screen: picture elements (pixels)

- Mirrors content of frame buffer memory
Opening the Box

Capacitive multitouch LCD screen

3.8 V, 25 Watt-hour battery

Computer board
Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

A5 processor
Chip Manufacturing Process

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch
The Instruction Set Architecture (ISA)

- Instructions
  - The words of a computer’s language are called instructions

- Instruction set
  - The vocabulary of a computer’s language is called its *instruction set*

- Instruction Set Architecture (ISA)
  - The set of instructions a particular CPU implements is an Instruction Set Architecture.
The Instruction Set Architecture (ISA)

The interface description separating the software and hardware
What is “Computer Architecture/Structure”?

- **Computer Architecture** = *Instruction Set Architecture (ISA)*
  - the one “true” language of a machine
  - *boundary* between hardware and software
  - the hardware’s *specification*; defines “what” a machine does;

  + **Machine Organization**
    - the “guts” of the machine; “how” the hardware works; the *implementation*; must obey the ISA abstraction

- We will explore both, and more!
QUESTION: Do all processor chips have the same architecture?

ANSWER: No. Each family of processor chip (MIPS, ARM, PIC, SPARC, Alpha, Motorola, Intel, et al.) has its own architecture.
QUESTION: Does your understanding of computers depend on which Assembly Language / ISA you study?

ANSWER: No. A well-designed modern assembly language, ISA is best, but any one is OK.
The MIPS ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Special

- 3 Instruction Formats: all 32 bits wide

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jump target</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 - R31</td>
</tr>
</tbody>
</table>

- PC
- HI
- LO

Program counter
HI and LO are special registers used with multiply and division
### The MIPS ISA

- **3 Instruction Formats:** all 32 bits wide

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
</table>
| R-type

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
</table>
| I-type

<table>
<thead>
<tr>
<th>OP</th>
<th>jump target</th>
</tr>
</thead>
</table>
| J-type

**For “R-type” instructions:**

- **OP** = Operation Code (opcode)
- **rs** and **rt** are source registers
- **rd** is the destination register
- **sa** is used with shift and rotate instructions
- **funct** is for instructions that share an opcode
The MIPS ISA – Example R-type instruction

- 3 Instruction Formats: all 32 bits wide

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R-type</td>
</tr>
<tr>
<td>OP</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
<td>I-type</td>
</tr>
<tr>
<td>OP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>J-type</td>
</tr>
</tbody>
</table>

add $s1, $s2, $s3  \hspace{1cm} Meaning: \hspace{0.5cm} $s1 = $s2 + $s3
Below the Program

- **High-level language program (in C)**
  
  ```c
  swap (int v[], int k); //swap v[k] and v[k+1]
  ...
  ```

- **Assembly language program (for MIPS)**
  
  ```assembly
  swap:  sll $2, $5, 2
          add $2, $4, $2
          lw $15, 0($2)
          lw $16, 4($2)
          sw $16, 0($2)
          sw $15, 4($2)
          jr $31
  ```

- **Machine (object) code (for MIPS)**
  
  ```
  0000 00 0000 00101 0001000010000000
  0000 00 0100 00010 0001000001000000
  1000 11 00010 01111 0000000000000000
  1000 11 00010 10000 0000000000000100
  1010 11 00010 10000 0000000000000000
  1010 11 00010 01111 0000000000000100
  0000 00 11111 00000 0000000000010000
  ```
Input Devices Input Object Code

000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000

Diagram:
- Processor
  - Control
  - Datapath
- Memory
- Devices
  - Input
  - Output
Object Code Stored in Memory

Processor

Control

Datapath

Memory

Devices

Input

Output

000000 00000 00101 0010000100000000
000000 00100 00010 0010000001000000
100111 00010 01111 0000000000000000
100111 00010 10000 0000000000000001
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000010
000000 11111 00000 0000000000001000
Control fetches an instruction from memory
Control decodes the instruction to determine what to execute
Datapath executes the instruction as directed by control.
What Happens Next?

Processor

Memory

000000 00000 00101 0001000010000000
000000 00100 00010 0001000001000000
100111 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000010000

Devices

Input

Output
Control fetches the *next* instruction from memory
At program completion the data to be output resides in memory.
Output Devices Output Data

Processor
- Control
- Datapath

Memory

Devices
- Input
- Output

0000010010100000000000000000000000000000
0000000010011110000000000000100
000000111110000000000001000
000000111111000000000000000000000000000000