EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 6_2: From Digital Circuit to Architecture

Xiaolin Xu
Department of ECE
Northeastern University

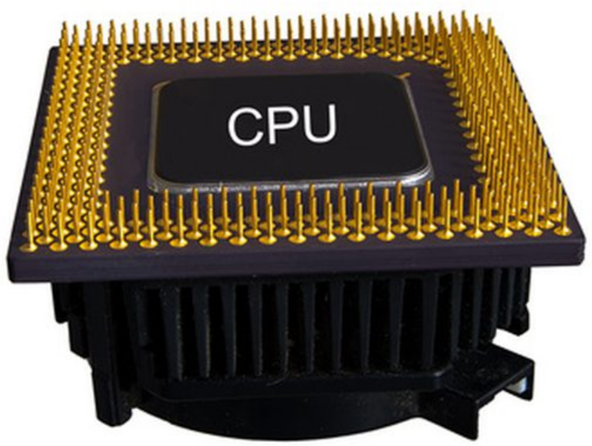
From Digital Circuit to Architecture

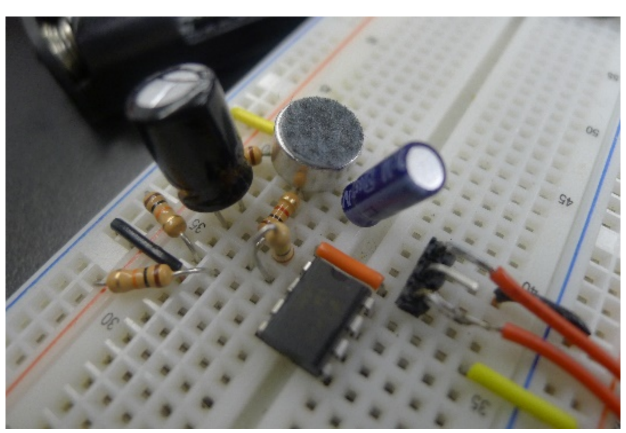
- Jumping up a few levels of abstraction
- **Architecture**: programmer's view of computer
  - Defined by instructions & operand locations
- **Microarchitecture:** how to implement an architecture in hardware (we will learn this later)

Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

Who Controls the HWs and How to?

Windows, Linux, Mac OS, FreeBSD, redhat, solaris, Sun Cobalt





Application/software	Programs
Operating Systems (OS)	Drivers
Architecture	**Instructions**
Micro-architecture	**Datapath/controller**
Logic	**Address/memory**
Digital Circuits	**Gates (AND, OR)**
Analog Circuits	Amplifier/filter
Devices	Transistor
Physics	Electron

Architecture and Micro-architecture

- Architecture: programmer's view of computer
  - Defined by instructions & operand locations
- Microarchitecture: how to implement an architecture in hardware

- RISC: Reduced Instruction Set Computer
- CISC: Complex Instruction Set Computer, e.g., x86

First Step to Learn any Computer Architecture

- Language!
- Computer's language are called ***instructions***
- **Instruction set**
  - The language of computer hardware
  - Used by all programs on a computer

Machine Language

- The main part of Instruction Set Architecture (ISA)
- Computer hardware only understands ‘1’ and ‘0’
- So, all the instruction set are encoded as binary numbers — the machine language

Machine Language

- The main part of Instruction Set Architecture (ISA)
- Computer hardware only understands ‘1’ and ‘0’
- So, all the instruction set are encoded as binary numbers — the machine language

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Assembly Language

- **Instructions:** commands in a computer's language
  - **Assembly language:** human-readable format of instructions
  - **Machine language:** computer-readable format (1's and 0's)
- **MIPS** architecture:
  - Developed by John Hennessy and his colleagues at Stanford and in the 1980's.
  - Used in many commercial systems, including Silicon Graphics, Nintendo, and Cisco

Once you've learned one architecture, it's easy to learn others

Co-inventor of RISC and MIPS

- President of Stanford University
- Professor of Electrical Engineering and Computer Science at Stanford since 1977
- Coinvented the Reduced Instruction Set Computer (RISC) with David Patterson
- Developed the MIPS architecture at Stanford in 1984 and cofounded MIPS Computer Systems
- As of 2004, over 300 million MIPS microprocessors have been sold

MIPS

- **Our focus**
  - **Microprocessor without Interlocked Pipeline Stages**
  - Developed by John Hennessy and his colleagues at Stanford in 1980's
  - Used in many commercial systems
  - Similar as RISC-V
- Underlying design principles:
  - Simplicity favors regularity
  - Make the common case fast
  - Smaller is faster
  - Good design demands good compromises

MIPS vs. Lab Computer Registers

- MIPS has:
  - 32 32-bit registers
  - Usually registers are faster than memory
- MIPS called "32-bit architecture" because it operates on 32-bit data
- Lab computer has four/4 8-bit registers
  - 16 bit instruction words
  - 8-bit architecture because datapath is 8 bits

Architecture Design Principles

- **Underlying design principles, as articulated by Hennessy and Patterson:**
  1. Simplicity favors regularity
  2. Make the common case fast
  3. Smaller is faster
  4. Good design demands good compromises

Instruction Example: Addition

C Code

```
a = b + c;
```

MIPS assembly code

```
add a, b, c
```

- **add:** mnemonic indicates operation to perform
- **b, c:** source operands (on which the operation is performed)
- **a:** destination operand (to which the result is written)

Instruction Example: Subtraction

C Code

```
a = b - c;
```

MIPS assembly code

```
sub a, b, c
```

- **sub:** mnemonic indicates operation to perform
- **b, c:** source operands (on which the operation is performed)
- **a:** destination operand (to which the result is written)

Instruction Example: Subtraction

Similar to addition - only mnemonic changes

C Code

```
a = b - c;
```

MIPS assembly code

```
sub a, b, c
```

- **sub:** mnemonic indicates operation to perform
- **b, c:** source operands (on which the operation is performed)
- **a:** destination operand (to which the result is written)

Design Principle 1

Simplicity favors regularity

- Consistent instruction format
- Same number of operands (two sources and one destination)
- easier to encode and handle in hardware

Multiple Instructions

- More complex code is handled by multiple MIPS instructions.

C Code

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

MIPS assembly code

```
add t, b, c  # t = b + c
sub a, t, d  # a = t - d
```

Design Principle 2

Make the common case fast

- MIPS includes only simple, commonly used instructions
- Hardware to decode and execute instructions can be simple, small, and fast
- More complex instructions (that are less common) performed using multiple simple instructions

Design Principle 2

Make the common case fast

- MIPS is a ***reduced instruction set computer* (RISC)**, with a small number of simple instructions
- Other architectures, such as Intel's x86, are ***complex instruction set computers* (CISC)**

- **Tradeoffs**
  -
  -
  -

Design Principle 2

Make the common case fast

- MIPS is a ***reduced instruction set computer* (RISC)**, with a small number of simple instructions
- Other architectures, such as Intel's x86, are ***complex instruction set computers* (CISC)**

- **Tradeoffs**
  - **CISC has implement highly complex instructions — high cost (overhead)**
  - **RISC achieves the same with MANY small instructions — low cost**
  - **What more? Faster VS. Slower! — Rarely used instruction consumes HW!**

Operands

- Operand location: physical location in computers
  - Registers
  - Memory
    - Large but?
  - Constants (also called immediates)

- Why a computer needs physical location?

Operands

- Operand location: physical location in computer
  - Registers
  - Memory
  - Constants (also called immediates)
- Why a computer needs physical location?
  - Computer only calculates using 1s and 0s, not a, b, c
  - It needs to know where to find the variables

Operands: Registers

- MIPS has 32 32-bit registers
- Registers are faster than memory
- MIPS called "32-bit architecture"
  - Because it operates on 32-bit data

Register

- $ before name
- Some registers used for specific purposes:
  - $0 always holds the constant value 0
  - the saved registers, $s0-$s7, used to hold variables
  - the temporary registers, $t0 - $t9, used to hold intermediate values during a larger computation

Design Principle 3

Smaller is Faster

- MIPS includes only a small number of registers

Operands: Registers

- Registers:
  - `$` before name
  - Example: `$0`, “register zero”, “dollar zero”
- Registers used for specific purposes:
  - `$0` always holds the constant value 0.
  - the *saved registers*, $s0-$s7, used to hold variables
  - the *temporary registers*, `$t0 - $t9,` used to hold intermediate values during a larger computation
  - Discuss others later

MIPS Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address