

**CPS104**  
**Computer Organization**  
**Lecture 1**  
**Introduction; Data Types**

**August 24 , 2009**

**Gershon Kedem**

**Slides available on:**  
**<http://kedem.cs.duke.edu/cps104/Lectures.html>**

# CPS104: Computer Organization

**Instructor:** Gershon Kedem [kedem@cs.duke.edu](mailto:kedem@cs.duke.edu)

**Office:** LSRC D342, **Phone:** 660-6555

**Office Hours:** Thu 4:00-5:00, or by appt.

**TA:** Mottaghi, Mohammad [mamad at cs.duke.edu](mailto:mamad@cs.duke.edu)

**Office:** D229 LSRC **Phone:** (919) 660-6597

**Office Hours:** TBA

**UTAs:** TBA

**Text:** **Computer Organization & Design: The Hardware /  
Software Interface (4th edition).**

**Web page:** <http://kedem.cs.duke.edu/cps104/>

**Newsgroup:** [duke.cs.cps104](mailto:duke.cs.cps104)

## More Information

We will use the **Altera DE2 Development Board(?)**

**Fridays (“recitation”) we will be hands on in class**

**You must complete the assigned tasks from recitation**

- **The Quiz is part of the grade.**
- **Yes, we really are meeting 3 times a week for 75min!**

# Course Outline:

- **1.Introduction to Computer Organization.**
  - ◆ What is in the box.
  - ◆ Integer and Floating point representation.
  - ◆ Basic data structures.
- **2.Instruction Set Architecture.**
  - ◆ The MIPS Processor.
  - ◆ Assembly level programming.
  - ◆ Instructions and data types representations.
  - ◆ Addressing, procedure calls and Exceptions.
  - ◆ Linking & Loading.
- **3.Digital Logic:**
  - ◆ Introduction: Digital Gates and Boolean Algebra.
  - ◆ Arithmetic and Logic circuits,
  - ◆ Other Functional Units
  - ◆ Flip-flops, Registers and Tristate drivers

# Course Outline (continue):

- ✱ **4. Single Cycle Per Instruction Processor.**
  - ◆ The Datapath.
  - ◆ Executing Instructions
  - ◆ Control
- ✱ **5. Interrupts.**
- ✱ **6. The Memory Hierarchy.**
  - ◆ Cache Memory.
  - ◆ Virtual Memory and Paging.
- ✱ **7. I/O Devices.**
  - ◆ I/O storage devices.
  - ◆ I/O buses and arbitration
  - ◆ LANs and WANs.
- ✱ **8. Advanced processors:**
  - ◆ Pipelined Processor.
  - ◆ Super-Scalar processor.
- ✱ **9. Advanced Computer Architecture. (If there is time).**
  - ◆ Fast Interconnects
  - ◆ Parallel Machines

# Grading

## \* Grade breakdown

- ◆ Midterm Exam: 15%
- ◆ Final Exam: 50%
- ◆ Homework Assignments 30%
- ◆ Quizzes 5%
- ◆ Class Participation 5%

## \* Homework policy:

- ◆ No sad stories please!
- ◆ No “**cooperation**” on homework (Unless **specified** in the assignment).
- ◆ You **must verify** that an electronic submission was successful!
- ◆ 5% reduction for each day late.
- ◆ 5% credit for early submission.
- ◆ No credit **after the homework was graded and handed back.**

## \* Grades posted on blackboard course-page:

- ◆ Password protected Access
- ◆ Written/email request for changes to grades.

# Homework-0

- ✱ **Send me ([kedem@cs.duke.edu](mailto:kedem@cs.duke.edu)) email message with: your name, year, major and a short description of your computer science / Engineering background.**
- ✱ **Readings:**
  - ◆ **Introduction, (Chapter-1),**
  - ◆ **Data representations. (Chapter-3, sections 3.1-3.3, 3.6).**

# Course Problems

- **Can't make midterm**
  - ◆ Tell us early and we will schedule alternate time
- **Forgot to turn in homework/ Dog ate the computer, network down.....**
  - ◆ **I do not accept phony excuses!**
  - ◆ If you have a legitimate problem, talk to me early (ASAP), **email me a reminder!**
- **What is cheating?**
  - ◆ Studying together in groups **is encouraged**
  - ◆ All written work must be your own. **Programs that are substantially the same as others will receive a grade of 0!**
  - ◆ Common examples of cheating: running out of time on a assignment and then pick up someone else's output, , person asks to borrow solution “just to take a look”, copying an exam question, ...

# Why Do You Have to Take This Course?

- **You want to be a plastic surgeon**
- **You all know how to use knife, needle, thread**
- **To be successful you don't just "cut & paste"**
- **You have to learn about the body**
  - ◆ **Skin**
  - ◆ **Bones**
  - ◆ **Muscles**
- **Different parts of the body require different skills**
  - ◆ **Nose**
  - ◆ **Hands**
  - ◆ **Legs**
- **Who do you want performing your surgery?**

# Why Do You Have to Take This Course?

- You want to be a race car driver
- You all know how to drive
- To be successful you don't just drive
- You must “be in touch with your vehicle”
- You have to learn about the vehicle
  - ◆ Engine
  - ◆ Suspension
  - ◆ Tires
- Is it drag racing, monster trucks, NASCAR, endurance
  - ◆ Different cars
  - ◆ Different style of driving
- Who is going to win the Indy 500, 16 year old or Jeff Gordon?

# Why Do You Have to Take This Course?

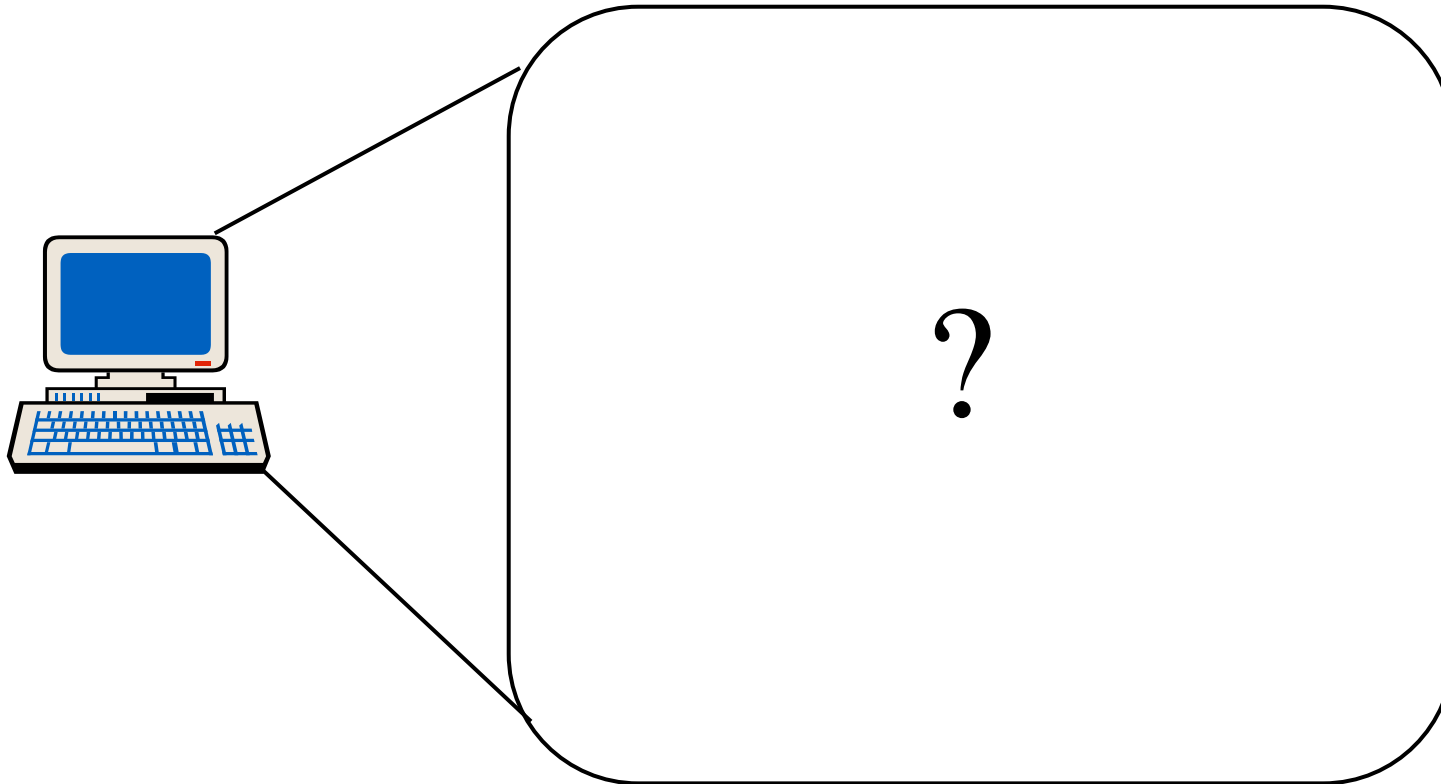
- **You want to be a Computer Scientist**
- **You know how to program (CPS 6, 100)**
- **To be successful you don't just program**
- **You have to understand the machine**
  - ◆ **Hardware: Processor, memory, disk, etc.**
  - ◆ **SW: Operating system, Programming Languages/Compilers**
- **What kind of computer scientist?**
  - ◆ **Databases, networks**
  - ◆ **Scientific computing (motion of planetary bodies, drug development, computational biology, economics, etc.)**
  - ◆ **Games, virtual reality**
  - ◆ **Embedded: Cell phones, mp3 player, cars**
- **Who's code do you want controlling your brakes, airbag, financial transactions? Script kiddie or computer scientist.**

# What You Will Learn

- ✱ **The basic operation of a computer:**  
**What is a computer? What does it do? How does it work?**
  - ◆ primitive operations (instructions)
  - ◆ arithmetic
  - ◆ instruction sequencing and processing
  - ◆ memory
  - ◆ input/output
  - ◆ etc.
- ✱ **Understand the relationship between abstractions**  
**What is done in hardware? What is done in software?**
  - ◆ interface design
  - ◆ high-level program to control signals (SW -> HW)
- ✱ **Software performance depends on understanding underlying HW**

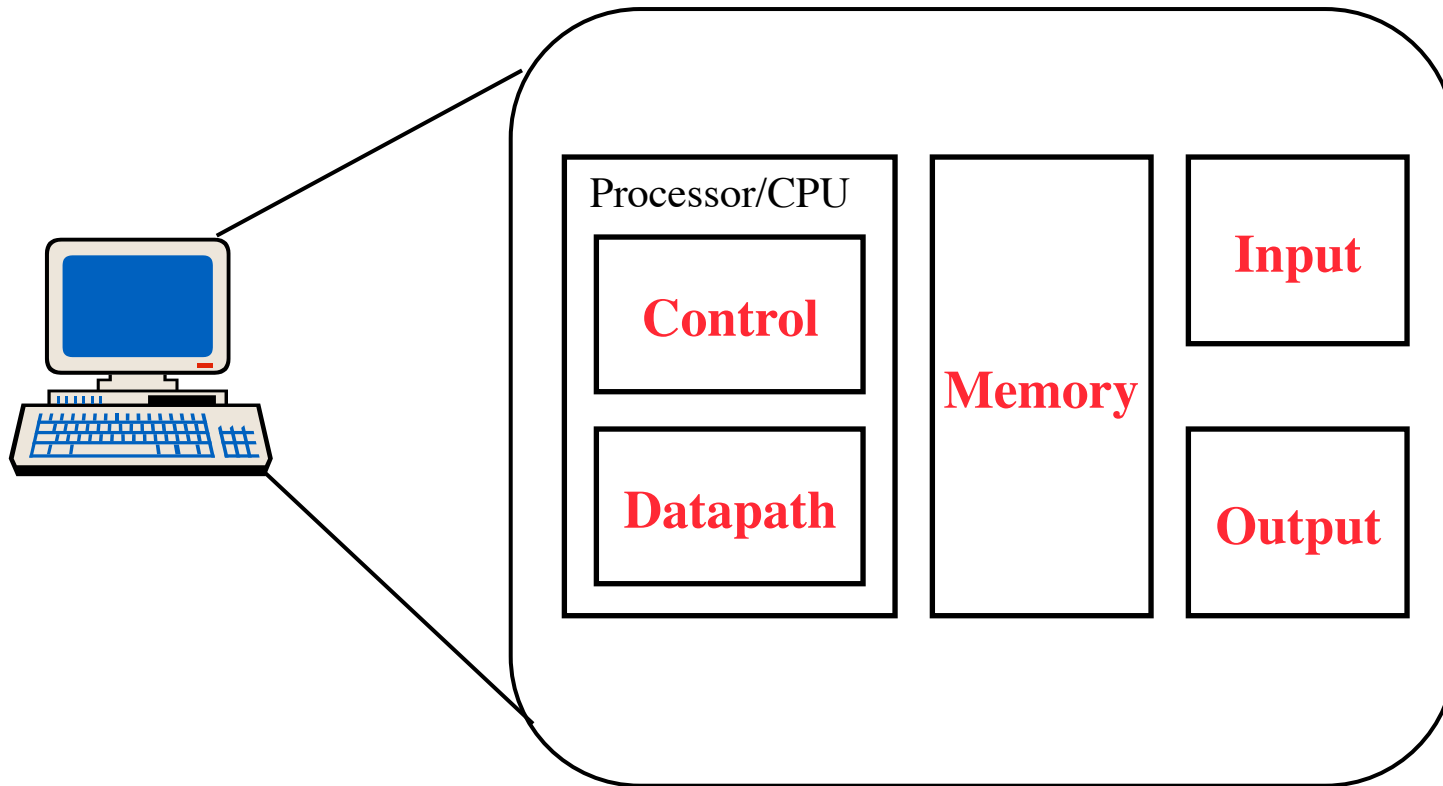
# The Big Picture

- What is inside a computer?
- How does it execute my program?

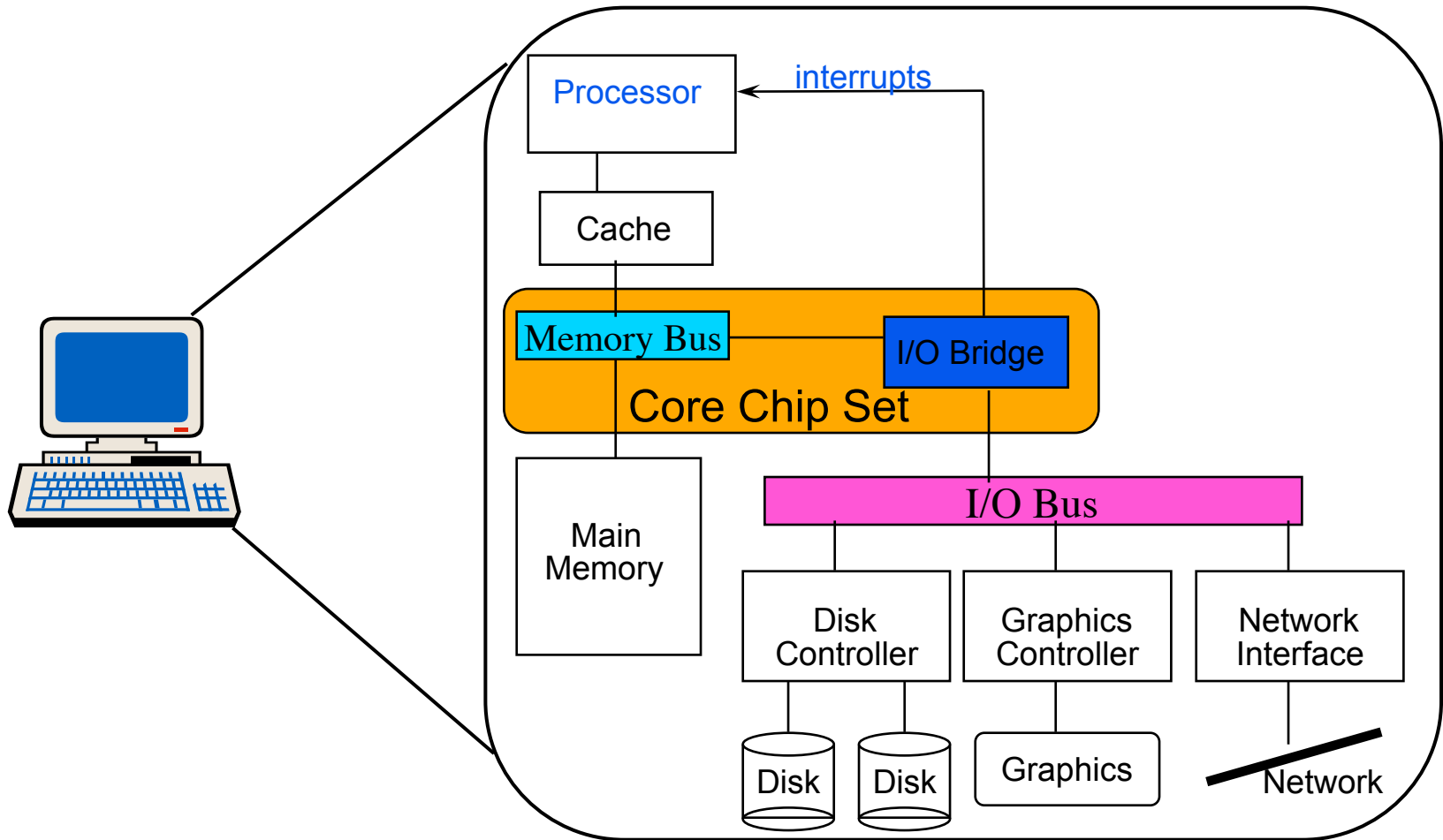


# The Big Picture

- **The Five Classic Components of a Computer**

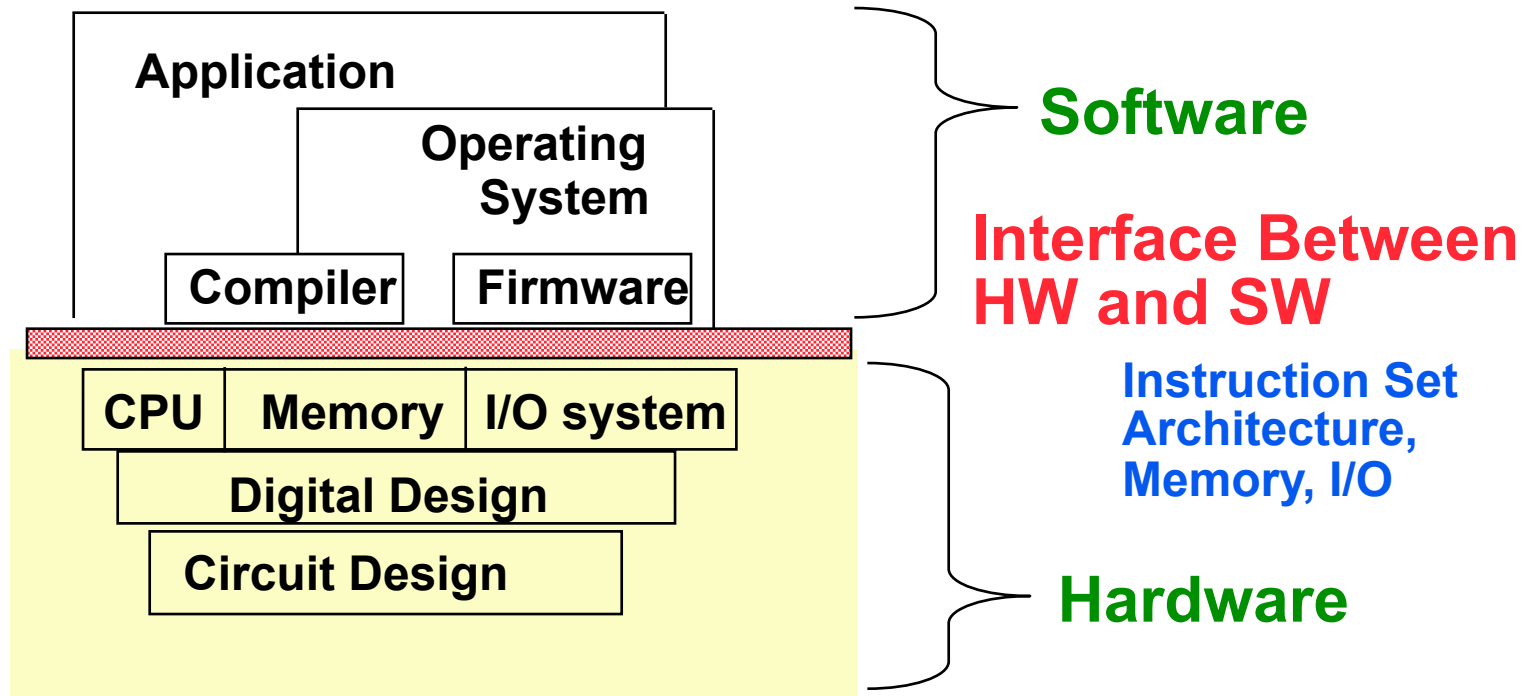


# System Organization



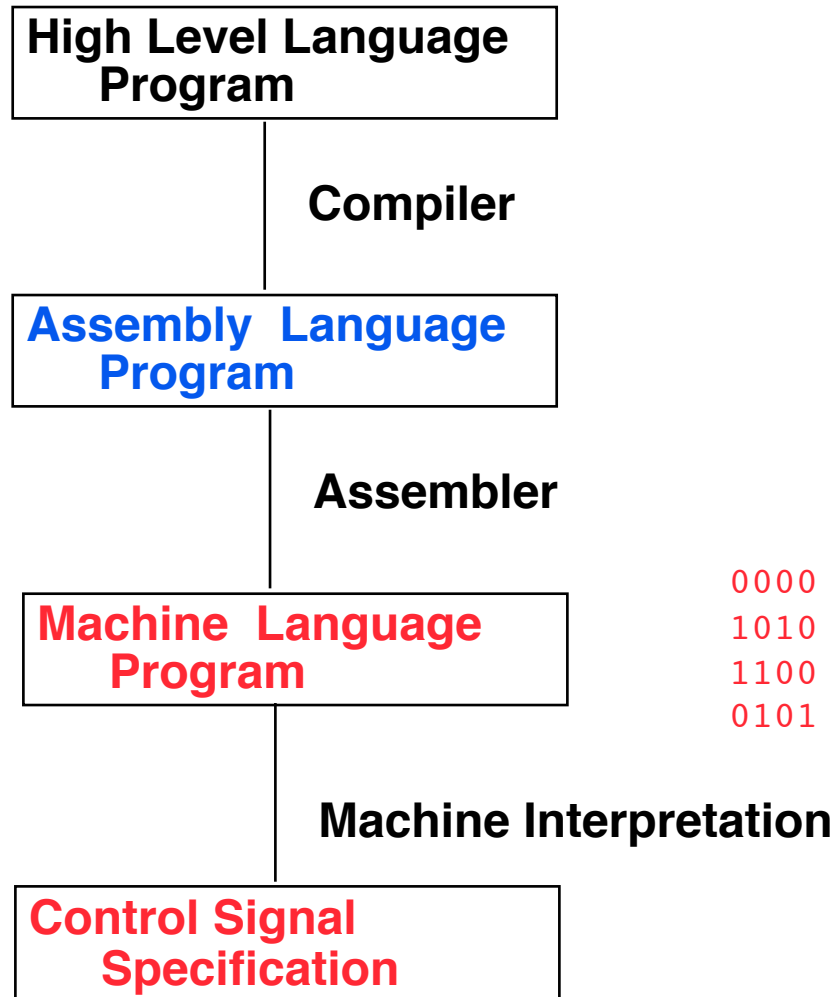
# What is Computer Architecture?

- Coordination of levels of abstraction



- Under a set of rapidly changing *Forces*

# Levels of Representation



```

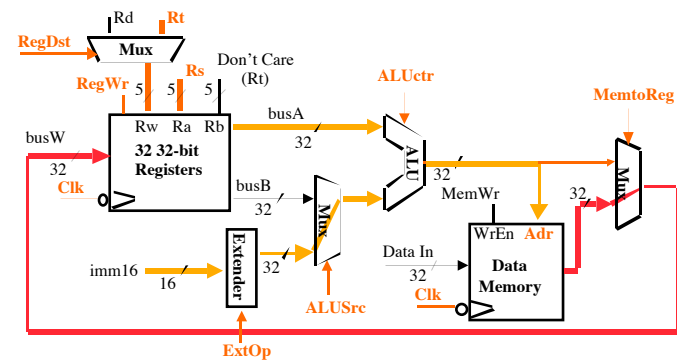
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
  
```

```

lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
  
```

```

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
  
```

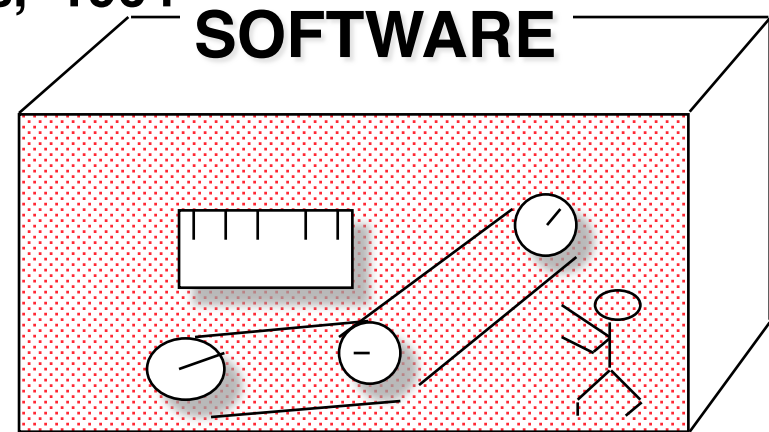


# Instruction Set Architecture

- ... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964

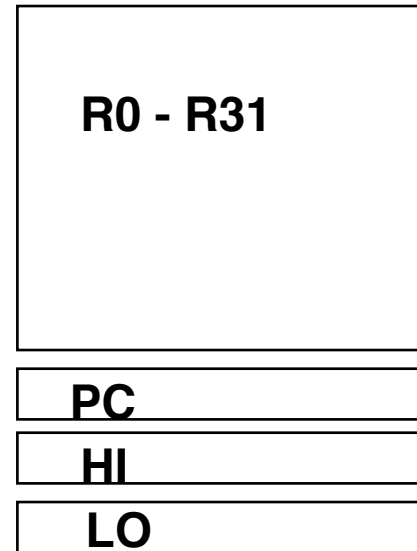
- Organization of Programmable Storage
- Data Types & Data Structures: Encoding & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions



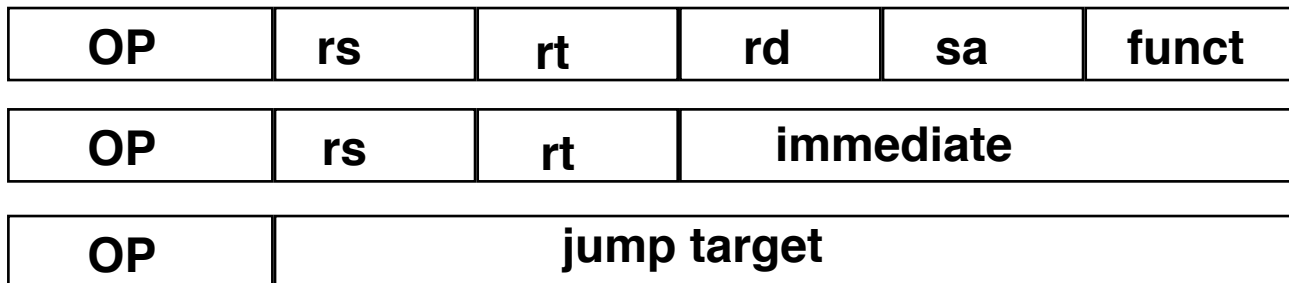
# MIPS I Instruction Set Architecture

## ★ Instruction Categories

- ◆ Load/Store
- ◆ Computational
- ◆ Jump and Branch
- ◆ Floating Point
- ◆ Memory Management
- ◆ Special



## 3 Instruction Formats: all 32 bits wide



# Organization

ISA Level

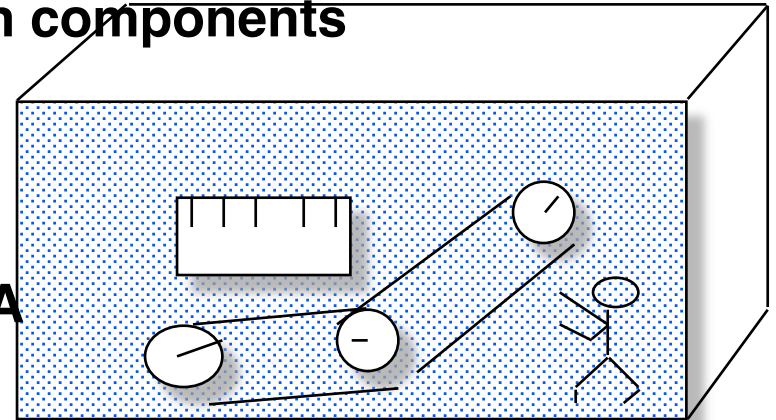
FUs & Interconnect

## *Logic Designer's View*

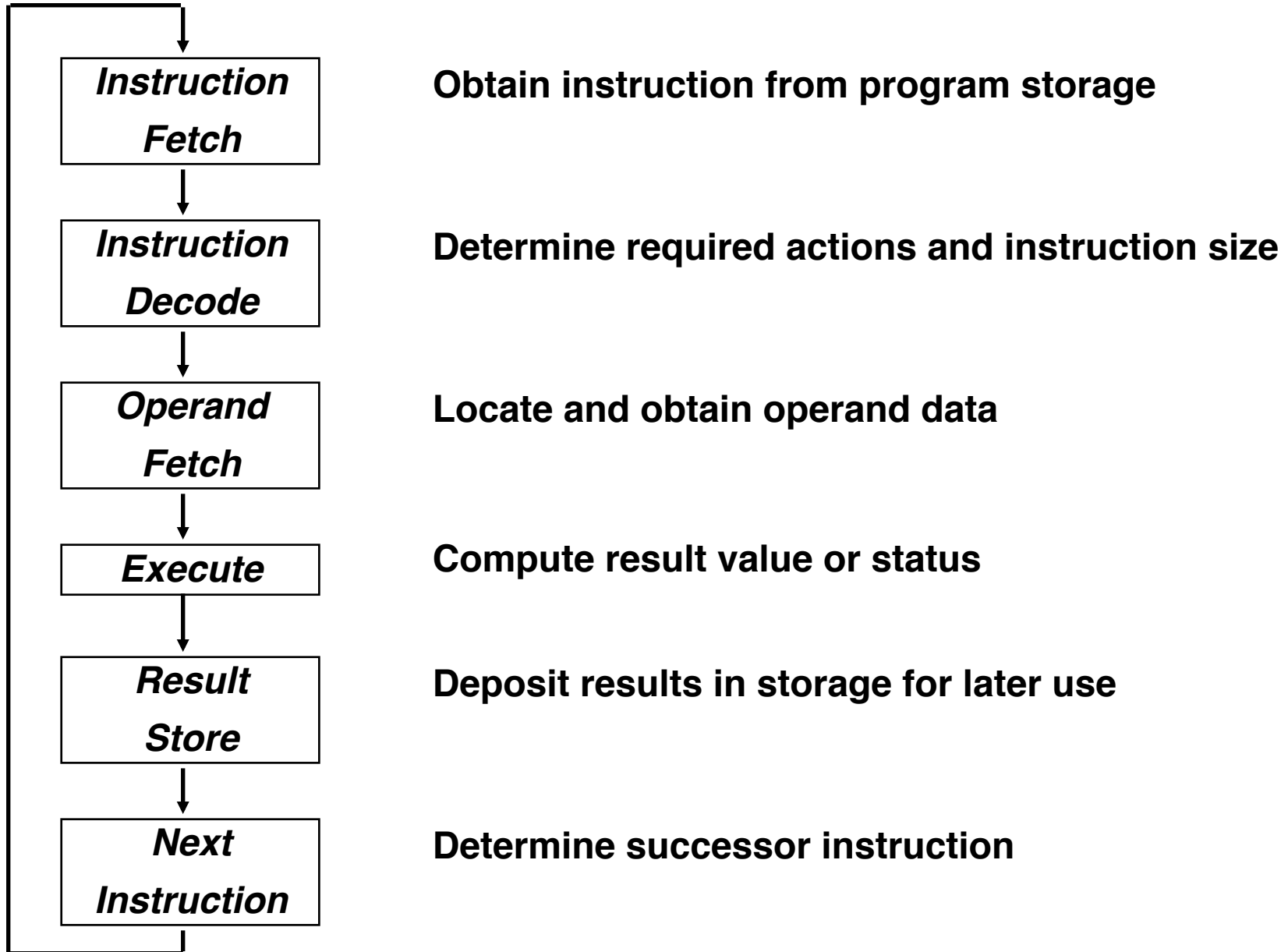
- Capabilities & Performance Characteristics of Principal Functional Units  
(e.g., Registers, ALU, Shifters, Logic Units, ...)
- Ways in which these components are interconnected
- nature of information flows between components
- logic and means by which such information flow is controlled.

Choreography of FUs to realize the ISA

Register Transfer Level Description



# Execution Cycle



# Summary

## Goal

- **Understand basic operation of a computer**

## Why?

- **Software performance is affected/determined by HW capabilities**
- **Future Computer Architects (Processor or System)**

# Summary (Continued)

## Agenda

- **Map “high-level” software to instructions**
- **Instructions are composed of hardware primitives**
  - ◆ how to use them
  - ◆ how to implement them
  - ◆ why a particular primitive
- **Memory for storing instructions and data**
  - ◆ Main memory
  - ◆ Caches
  - ◆ interaction with operating system
- **Input/Output**

# Data Representation

- **Computers store variables (data)**
- **Typically Numbers and Characters or composition of these**
- **We reason about numbers many different ways**
- **The key is to use a representation that is amenable to hardware implementation and is “efficient”**

# Data Representation

- **Question: How do computers store (represent) numbers?**

When you write in a program:

```
int x; float y;  
x= 10;  
y = 0.34;
```

- **What do the variables x and y actually hold?**

# Number Systems

- **A number is a mathematical concept**
  - ◆ 10
- **Many ways to represent a number**
  - ◆ 10, ten, 2x5, X, 100/10, ~~III~~ ~~III~~
- **Symbols are used to create a representation**
- **Which representation is best for addition and subtraction?**
- **Which representation is best for multiplication and division?**

# More Number Systems

- **Humans use decimal (base 10)**

- ◆ digits 0-9 are composed to make larger numbers

$$31 = 3 \cdot 10^1 + 1 \cdot 10^0$$

- ◆ weighted positional notation

- **Addition and Subtraction are straightforward**

- ◆ carry and borrow (today called regrouping)

- **Multiplication and Division less so**

- ◆ can use logarithms and then do adds and subtracts

# Changing Base (Radix)

- **Given 4 positions (4 symbols), what is largest number you can represent?**

# Number Systems for Computers

- Today's computers are built from transistors (Electronic Switches)
- A switch is either **off** or **on**. (Voltage across a switch high or low)
- Need to represent numbers using only **off-on** or **high-low**
  - Use only two symbols!
- **off** and **on** can represent the digits **0** and **1**
  - ♦ A bit can have a value of 0 or 1
- Computer memory is **organize in words**.
- Each computer word has a fixed number of **binary bits**.
- Typical desktop computer uses 32-bit words.
- 32 bit computers operate on **Bytes** (8 bits), **half words** (16 bits), **words** (32-bits) and **double words** (64 bits).

# Binary representation

- weighted positional notation using base 2

$$11_{10} = 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1011_2$$

$$11_{10} = 8 + 2 + 1$$

What is largest number that can be represented, given 4 bits?

# Conversion from Decimal to Binary

- **N** is a positive Integer (in decimal representation)
- $b_i$   $i=0,\dots,k$  are the bits (binary digits) for the binary representation of **N**
- $N = b_k * 2^k + \dots + b_2 * 2^2 + b_1 * 2 + b_0$
- binary representation:  $b_k \dots b_3 b_2 b_1 b_0$
- How do I compute  $b_0$ ?

Compute binary representation of 11?

# Conversion from Decimal

```
i=0;
```

```
while N > 0 do
```

```
     $b_i = N \% 2;$  //  $b_i = \text{remainder}; N \bmod 2$ 
```

```
     $N = N / 2;$  //  $N$  becomes quotient of division
```

```
    i++;
```

```
end while
```

- Replace 2 by **A** and you have an algorithm that computes the **base A** representation for N

# Powers of 2

<u>N</u>	<u>2<sup>n</sup></u>	<u>Binary</u>
0	1	0000000001
1	2	0000000010
2	4	0000000100
3	8	0000001000
4	16	0000010000
5	32	00000100000
6	64	00001000000
7	128	00010000000
8	256	00100000000
9	512	01000000000
10	1024 (1K)	10000000000

# Binary, Octal and Hexidecimal numbers

- **Computers can input and output decimal numbers but they convert them to internal binary representation.**
- **Binary is good for computers, hard for us to read**
  - ◆ **Use numbers easily computed from binary**
- **Binary numbers use only two different digits: {0,1}**
  - ◆ **Example:  $1200_{10} = 0000010010110000_2$**
- **Octal numbers use 8 digits: {0 - 7}**
  - ◆ **Example:  $1200_{10} = 04260_8$**
- **Hexidecimal numbers use 16 digits: {0-9, A-F}**
  - ◆ **Example:  $1200_{10} = 04B0_{16} = 0x04B0$**
  - ◆ **One does not distinguish between upper and lower case**

# Binary and Octal

- Easy to convert Binary numbers To/From Octal.
- Group the binary digits in groups of three bits and convert each group to an Octal digit.

•  $2^3 = 8$

Bin.	Oct.
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

**Example:**

11 000 010 011 001 110 100 111 101 010 101<sub>2</sub>  
3 0 2 3 1 6 4 7 5 2 5<sub>8</sub>

# Binary and Hex

- To convert to and from hex: group binary digits in groups of four and convert according to table
- $2^4 = 16$

Hex	Bin	Hex	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

**Example:**

1100 0010 0110 0111 0100 1111 1101 0101<sub>2</sub>  
C 2 6 7 4 F D 5<sub>16</sub>

# Issues for Binary Representation

- **Complexity of arithmetic operations**
- **Negative numbers**
- **Maximum representable number**

# Binary Integers

## \*Unsigned Integers:

♦  $i = 100101_2$ ;  $i = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

\*4 bits  $\Rightarrow$  max number is 15

\*What about representing negative numbers?

# Sign-Magnitude Representation for Integers

- Add a sign bit

  - ◆ Example:  $010110_2 = 22_{10}$  ;  $110110_2 = -22_{10}$

- Advantages:

  - ◆ Simple extension of unsigned numbers.
  - ◆ Same number of positive and negative numbers.

- Disadvantages:

  - ◆ Two representations for 0:  $0=000000$ ;  $-0=100000$ .
  - ◆ Algorithm (circuit) for addition depends on the arguments' signs.

# 2's Complement Representation for Integers

- Key idea is to use largest positive binary numbers to represent negative numbers
- Obtain negative number by subtracting large constant
- $i = -a_{n-1} * 2^{n-1} + a_{n-2} * 2^{n-2} + \dots + a_0 * 2^0$

## 6-bit examples:

$$010110_2 = 22_{10} ; 101010_2 = -22_{10}$$

$$0_{10} = 000000_2 ; 1_{10} = 000001_2 ; -1_{10} = 111111_2$$

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

# 2's Complement

- **Advantages:**

- ◆ Only one representation for 0:  $0 = 000000$
- ◆ Addition algorithm independent of sign bits.

- **Disadvantage:**

- ◆ One more negative number than positive :  
**Example:** 6-bit 2's complement number.

$100000_2 = -32_{10}$ ; but  $32_{10}$  could not be represented

# 2's Complement Negation and Addition

- ✱ To negate a number do:
  - ◆ Step 1. complement the digits
  - ◆ Step 2. add 1

## Examples

$$\begin{array}{r} 14_{10} = 001110_2 \\ -14_{10} = 110001_2 \\ \quad \quad \quad + 1 \\ \hline 110010_2 \end{array}$$

$$\begin{array}{r} 010010_2 \\ +110010_2 \\ \hline 000100_2 \end{array}$$

- ✱ To add signed numbers use regular addition but disregard carry out
  - ◆ Example  $18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}$

## 2's Complement (cont.)

\* Example:  $A = 0x0ABC$ ;  $B = 0x0FEB$ .

\* Compute:  $A + B$  and  $A - B$  in 16-bit 2's complement arithmetic.

# 2's Complement Precision Extension

- ✱ Most computers today support 32-bit (int) or 64-bit integers
  - ◆ 64-bit using gcc is **long long**
  - ◆ 64-bit using Digital/Compaq compiler is **long**
- ✱ To extend precision do **sign bit extension**
  - ◆ precision is number of bits used to represent a number

## Example

$14_{10} = 001110_2$  in 6-bit representation.

$14_{10} = 000000001110_2$  in 12-bit representation

$-14_{10} = 110010_2$  in 6-bit representation

$-14_{10} = 111111110010_2$  in 12-bit representation.

# Reading Assignment

## Reading:

- Chapter 2.4 pages 80-94,
- Chapter 3.1-3.2, 3.5 pages 224-229, 242-253
- Read Chapter 1,