PES UNIVERSITY ONLINE

DIGITAL DESIGN & COMPUTER ORGANISATION

Dr. Reetinder Sidhu and Dr. Kiran D C
Department of Computer Science and Engineering

DIGITAL DESIGN & COMPUTER ORGANISATION

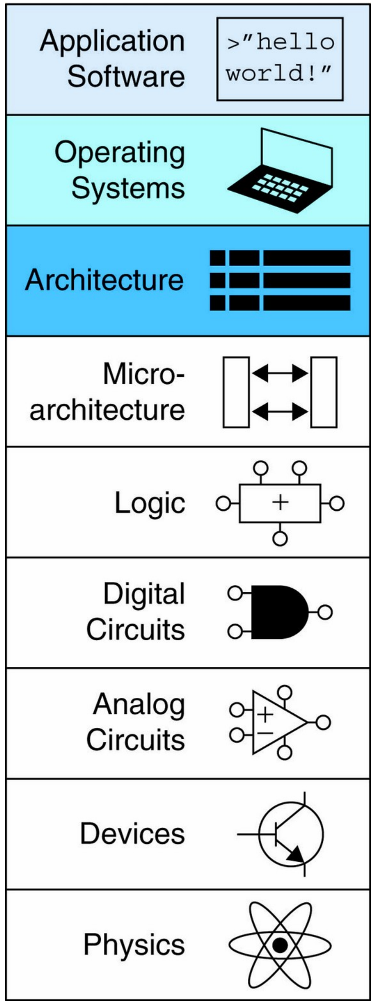
Computer Organization Introduction

Dr. Reetinder Sidhu and Dr. Kiran D C

Department of Computer Science and Engineering

Topics

- Introduction
- Assembly Language
- Machine Language
- Programming
- Addressing Modes
- Lights, Camera, Action:
  Compiling, Assembling, & Loading
- Odds and Ends



Assembly Language

PES UNIVERSITY ONLINE

- **Instructions:** commands in a computer’s language
  - **Assembly language:** human-readable format of instructions
  - **Machine language:** computer-readable format (1’s and 0’s)

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

Underlying design principles:

1. Simplicity favors regularity
2. Make the common case fast
3. Smaller is faster
4. Good design demands good compromises

Instructions: Addition

Instructions: Addition

C Code

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

Instructions: Addition

C Code

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

MIPS assembly code

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

Instructions: Addition

C Code

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

MIPS assembly code

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

- **add:** mnemonic indicates operation to perform

Instructions: 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)

Instructions: 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)

Instructions: Subtraction

C Code

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

Instructions: Subtraction

- Similar to addition - only mnemonic changes

C Code

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

Instructions: Subtraction

- Similar to addition - only mnemonic changes

C Code

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

MIPS assembly code

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

Instructions: Subtraction

- Similar to addition - only mnemonic changes

C Code

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

MIPS assembly code

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

- **sub:** mnemonic

Instructions: Subtraction

- Similar to addition - only mnemonic changes

C Code

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

MIPS assembly code

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

- **sub:** mnemonic
- **b, c:** source operands

Instructions: Subtraction

- Similar to addition - only mnemonic changes

C Code

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

MIPS assembly code

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

- **sub:** mnemonic
- **b, c:** source operands
- **a:** destination operand

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.

Multiple Instructions

- More complex code is handled by multiple MIPS instructions.

C Code

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

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

PES UNIVERSITY ONLINE

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
- 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)**

Operands

- Operand location: physical location in computer
  - Registers
  - Memory
  - Constants (also called *immediates*)

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

Design Principle 3

PES UNIVERSITY ONLINE

Smaller is Faster

- MIPS includes only a small number of registers

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

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

- Revisit add instruction

C Code

```
a = b + c
```

Instructions with Registers

- Revisit add instruction

C Code

```
a = b + c
```

MIPS assembly code

```
$s0 = a, $s1 = b, $s2 = c
```

Instructions with Registers

- Revisit add instruction

C Code

```
a = b + c
```

MIPS assembly code

```
$s0 = a, $s1 = b, $s2 = c

add $s0, $s1, $s2
```

Operands: Memory

- Too much data to fit in only 32 registers
- Store more data in memory
- Memory is large, but slow
- Commonly used variables kept in registers

Think About It

- Using the instructions learnt so far (add and sub) can you write a program (sequence of instructions) that would left shift a number (stored in some register) by three bit positions? How many registers do you need?