جامعة المستقبل
AL MUSTAQBAL UNIVERSITY
كلية الهندسة والتقنيات الهندسية

Computer Organization and Application

Lecture 4
Instruction set design in von Neuman

Dr Mohammed Fadhil
Email: mohammed.fadhil1@uomus.edu.iq

Learning Objectives

- Understand the Von Neumann model
- Understand the Instruction: Opcode & Operands
- Understand of Reading/Loading Operands from Memory
- Understand the Reading/Loading Word-Addressable Memory
- Overview of Instruction Format With Immediate

von Neumann Model: Two Key Properties

- **Von Neumann model** is also called stored program computer (instructions in memory). It has two key properties:
- **Stored program**
  - Instructions stored in a linear memory array
  - Memory is unified between instructions and data
    - **The interpretation of a stored value depends on the control signals**
- **Sequential instruction processing**
  - **One instruction processed** (fetched, executed, completed) at a time
  - **Program counter** (instruction pointer) identifies the current instruction
  - **Program counter is advanced sequentially** except for control transfer instructions

Stored Program & Sequential Execution

- Instructions and data are **stored in memory**
  - Typically the **instruction length is the word length**
- The processor fetches instructions from memory **sequentially**
  - **Fetches one instruction**
  - **Decodes and executes the instruction**
  - **Continues with the next instruction**
- The address of the current instruction is stored in the **program counter (PC)**
  - If **word-addressable** memory, the processor **increments the PC by 1 (in LC-3)**
  - If **byte-addressable** memory, the processor increments the PC by the instruction length in bytes (4 in MIPS)
    - In MIPS the OS typically sets the PC to **0x00400000** (start of a program)

A Sample Program Stored in Memory

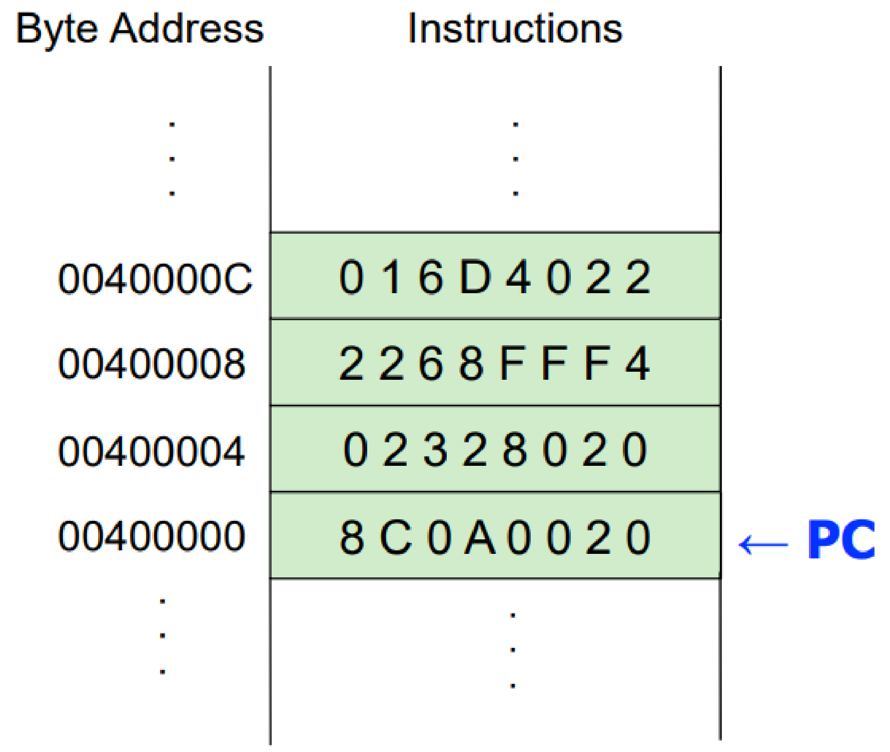
- A sample MIPS program
  - 4 instructions stored in consecutive words in memory
    - No need to understand the program now. We will get back to it

MIPS assembly

```
lw    $t2, 32($0)
add   $s0, $s1, $s2
addi  $t0, $s3, -12
sub   $t0, $t3, $t5
```

Machine code (encoded instructions)

```
0x8C0A0020
0x02328020
0x2268FFF4
0x016D4022
```

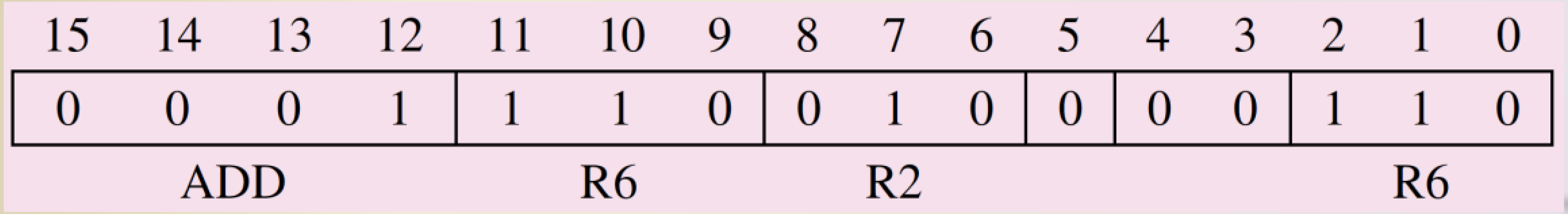


The Instruction

- An **instruction** is the most basic unit of computer processing
  - **Instructions** are words in the language of a computer
  - **Instruction Set Architecture (ISA)** is the vocabulary
- The language of the computer can be written as
  - **Machine language**: Computer-readable representation (that is, 0's and 1's)
  - **Assembly language**: Human-readable representation
- We will study LC-3 instructions and MIPS instructions
  - Principles are similar in all ISAs (x86, ARM, RISC-V, …)

The Instruction: Opcode & Operands

- An instruction is made up of two parts
  - **Opcode** and **Operands**
- **Opcode** specifies **what** the instruction does
- **Operands** specify **who** the instruction is to do it to
- Both are specified in **instruction format** (or instr. encoding)
  - An LC-3 instruction consists of 16 bits (bits [15:0])
  - Bits [15:12] specify the opcode → 16 distinct opcodes in LC-3
  - Bits [11:0] are used to figure out where the operands are



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0
ADD				R6			R2						R6		

Instruction Types

- There are **three main types of instructions**
- Operate instructions
  - Execute operations in the ALU
- Data movement instructions
  - Read from or write to memory
- Control flow instructions
  - Change the sequence of execution

Let us start with some example instructions

An Example Operate Instruction

- **Addition**

High-level code

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

Assembly

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

- **add**: mnemonic to indicate the operation to perform
- **b, c**: source operands
- **a**: destination operand
- **$a \leftarrow b + c$**

Registers

- We map variables to registers

LC-3 registers

b = R1
c = R2
a = R0

Assembly

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

MIPS registers

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

From Assembly to Machine Code in LC-3

- **Addition**

LC-3 assembly

ADD R0, R1, R2

Field Values

OP	DR	SR1			SR2
1	0	1	0	00	2

Machine Code (Instruction Encoding)

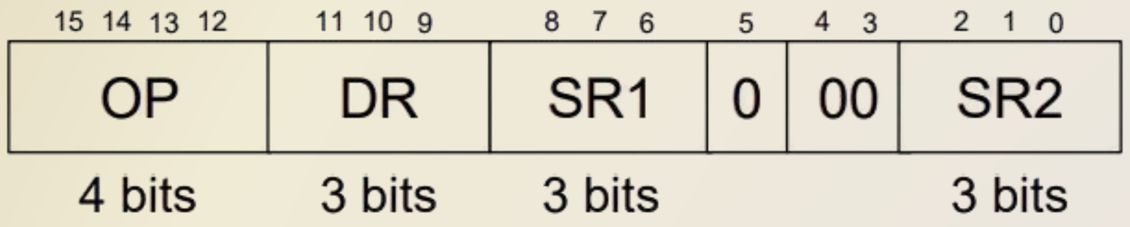
OP	DR	SR1			SR2
0 0 0 1	0 0 0	0 0 1	0	0 0	0 1 0
15 14 13 12	11 10 9	8 7 6	5	4 3	2 1 0

0x1042

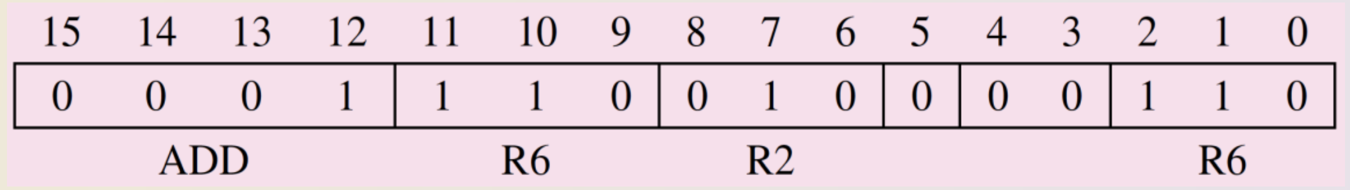
Machine Code, in short (hexadecimal)

Instruction Format (or Encoding)

- **LC-3 Operate Instruction Format**



  - **OP** = **opcode** (what the instruction does)
    - E.g., **ADD** = 0001
      - **Semantics: DR ← SR1 + SR2**
    - E.g., **AND** = 0101
      - **Semantics: DR ← SR1 AND SR2**
  - **SR1, SR2** = source registers
  - **DR** = destination register

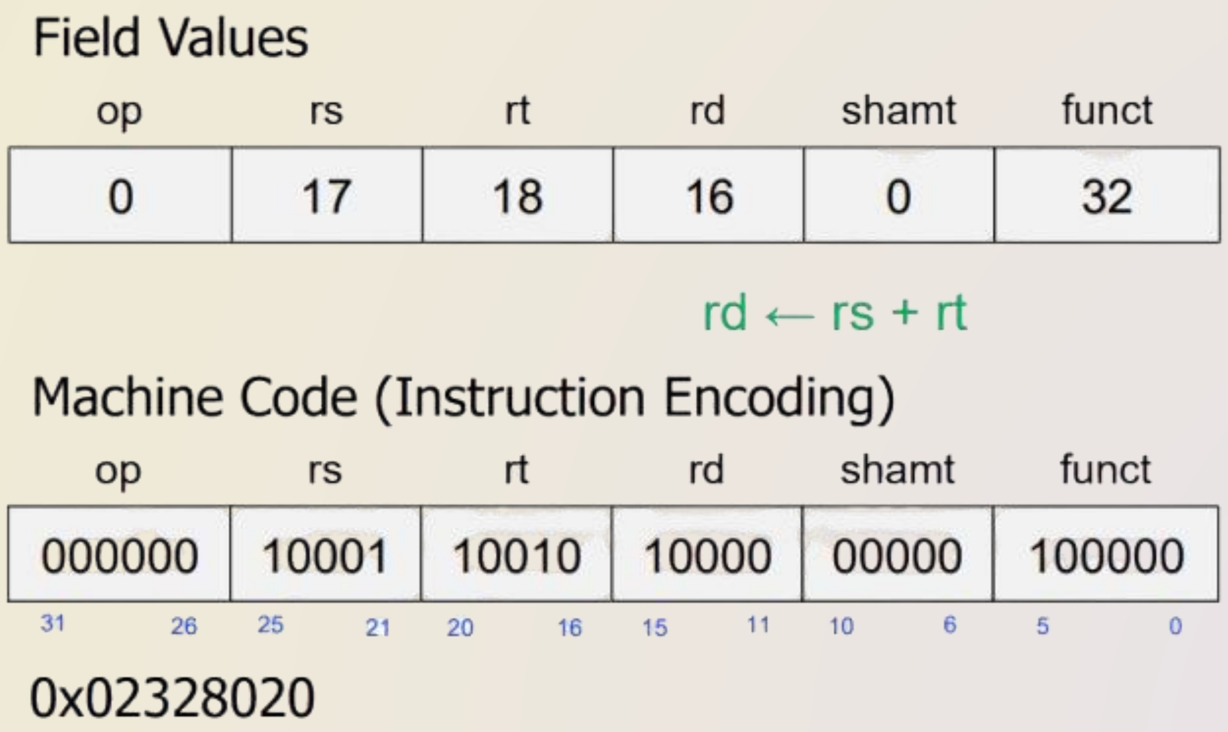


From Assembly to Machine Code in MIPS

- **Addition**

MIPS assembly

add $s0, $s1, $s2



Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32

rd ← rs + rt

Machine Code (Instruction Encoding)

op	rs	rt	rd	shamt	funct
000000	10001	10010	10000	00000	100000
31 26	25 21	20 16	15 11	10 6	5 0

0x02328020

Instruction Format: R-Type in MIPS

- MIPS R-type Instruction Format
  - **3 register operands**

0	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - **0** = opcode
  - **rs, rt** = source registers
  - **rd** = destination register
  - **shamt** = shift amount (only shift operations)
  - **funct** = operation in R-type instructions

Reading Operands from Memory

- With **operate instructions**, such as addition, we tell the computer to **execute arithmetic (or logic) computations in the ALU**
- We also need instructions to **access the operands from memory**
  - Load them from memory to registers
  - Store them from registers to memory
- Next, we see how to **read (or load) from memory**
- **Writing (or storing)** is performed in a similar way, but we will talk about that later

An Example Operate Instruction

- **Load word**

High-level code

a = A[i];

Assembly

load a, A, i

- **load**: mnemonic to indicate the load word operation
- **A**: base address
- **i**: offset
  - E.g., **immediate or literal** (a constant)
- **a**: destination operand
- **Semantics**: a ← Memory[A + i]

Load Word in LC-3 and MIPS

- **LC-3 assembly**

R3 ← Memory[R0 + 2]

```
a = A[2];
```

LC-3 assembly

```
LDR R3, R0, #2
```

R3 ← Memory[R0 + 2]

- **MIPS assembly** (assuming word-addressable)

High-level code

```
a = A[2];
```

MIPS assembly

```
lw $s3, 2($s0)
```

$s3 ← Memory[$s0 + 2]

These instructions use a particular addressing mode (i.e., the way the address is calculated), called base+offset

Load Word in Byte-Addressable MIPS

- **MIPS assembly**

High-level code

```
a = A[2];
```

MIPS assembly

```
lw $s3, 8($s0)
```

$s3 ← Memory[$s0 + 8]

- Byte address is calculated as: **word_address * bytes/word**
  - 4 bytes/word in MIPS
  - If LC-3 were byte-addressable (i.e., LC-3b), 2 bytes/word

Instruction Format With Immediate

- **LC-3**

LC-3 assembly

LDR R3, R0, #2

Field Values

OP	DR	BaseR	offset6
6	3	0	2
15 12	11 9	8 6	5 0

- **MIPS**

MIPS assembly

lw $s3, 8($s0)

Field Values

op	rs	rt	imm
35	16	19	8
31 26	25 21	20 16	15 0

I-Type

Quiz – Group C

1. **In a hierarchical computer system design, what does ‘structure’ refer to? (Multichoice).**
   - a) The way components are connected b) The operation of individual components c) The type of memory used d) The control flow of data
2. **Can you name and explain the four basic functions that every computer performs? (List).**
3. **All models in a computer family, like the Intel x86 family, have the same organization but different architectures (True/False– correct if false).**

Quiz – Group B

1. **What does the sequencing logic in the Control Unit do? (Multichoice).**
   – **a)** Stores temporary data **b)** Controls the order of instruction execution **c)** Connects the CPU to peripherals **d)** Manages data movement
2. **Think about the CPU—what are its main parts, and what does each one do? (List).**
3. **The Control Unit (CU) in the CPU is responsible for performing arithmetic and logic operations. (True/False– correct if false).**

THANK YOU ☺