Chapter 04: Instruction Sets and the Processor organizations

Lesson 03:

Basic operations and Instruction Formats

Objective

- Understand the basic Instructions
- The operations for executing instructions
- Learn Instruction Formats
- Assembly language representation of instruction

Basic Instructions

Basic Instructions

- Memory operations by the load and store instruction
- Register transfers (also called move) instruction
- Instruction execution and straight line sequence
- Branching instruction

Basic Instructions

- Subroutine and program flow control Instructions
- Condition codes
- Simple arithmetic Instructions
- Logic Instructions

Basic Instructions

- Compare and test Instructions
- Shift and rotate Instructions
- Multiplication and division Instructions
- Floating point arithmetic Instructions
- Input and output Instructions

Assembly Language and Notations

Assembly Language and Notations

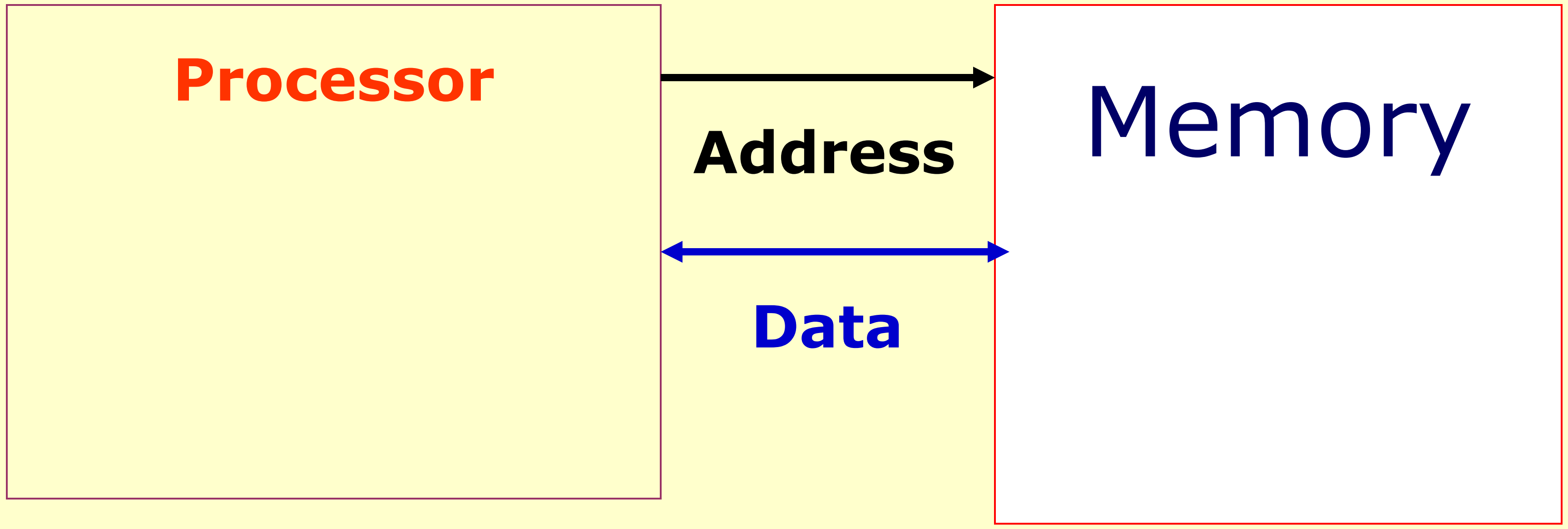
- An instruction represented in notations that can be easily understood
- For a register-related operation—
- MOV for copy of word into a register

Assembly Language and Notations

- LD for loading copy of word into a register from memory location
- ST for storing copy of word from a register to memory location
- ADD, MUL, SUB, and DIV for addition, multiplication, subtraction, and division operations, respectively

Instruction Fetching and Execution

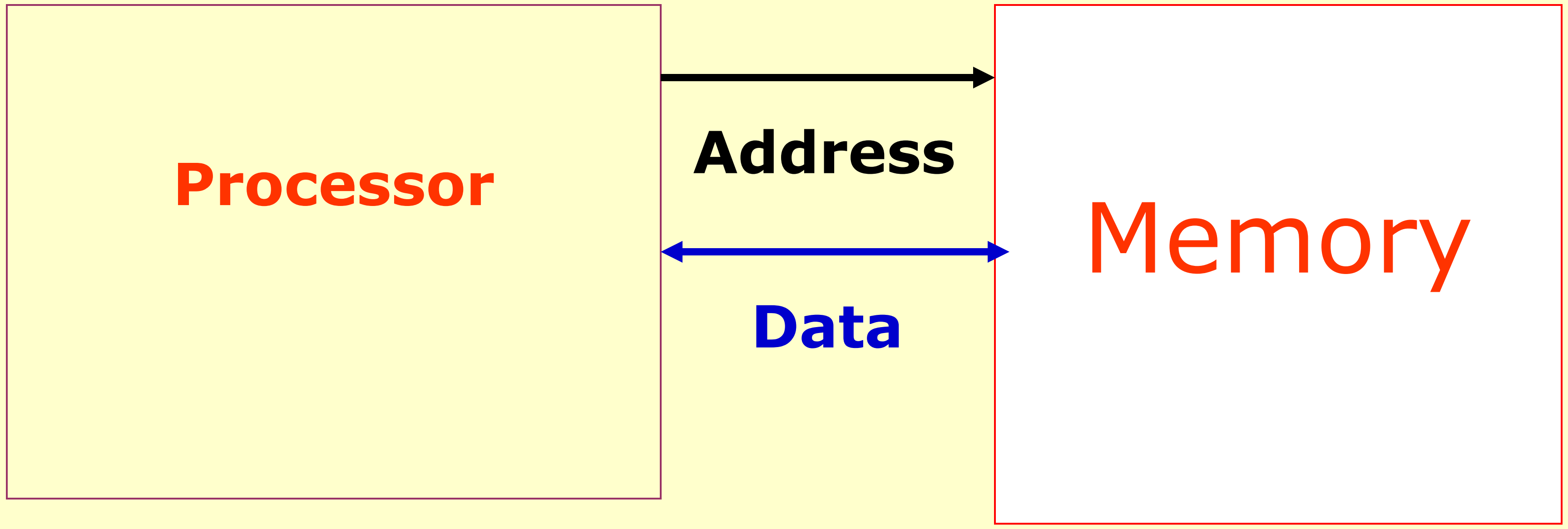
Instruction Fetch



Step 1) Processor requests instruction from memory using address in PC (or IP) register

Step 2) Memory using data bus returns the instruction to IR register

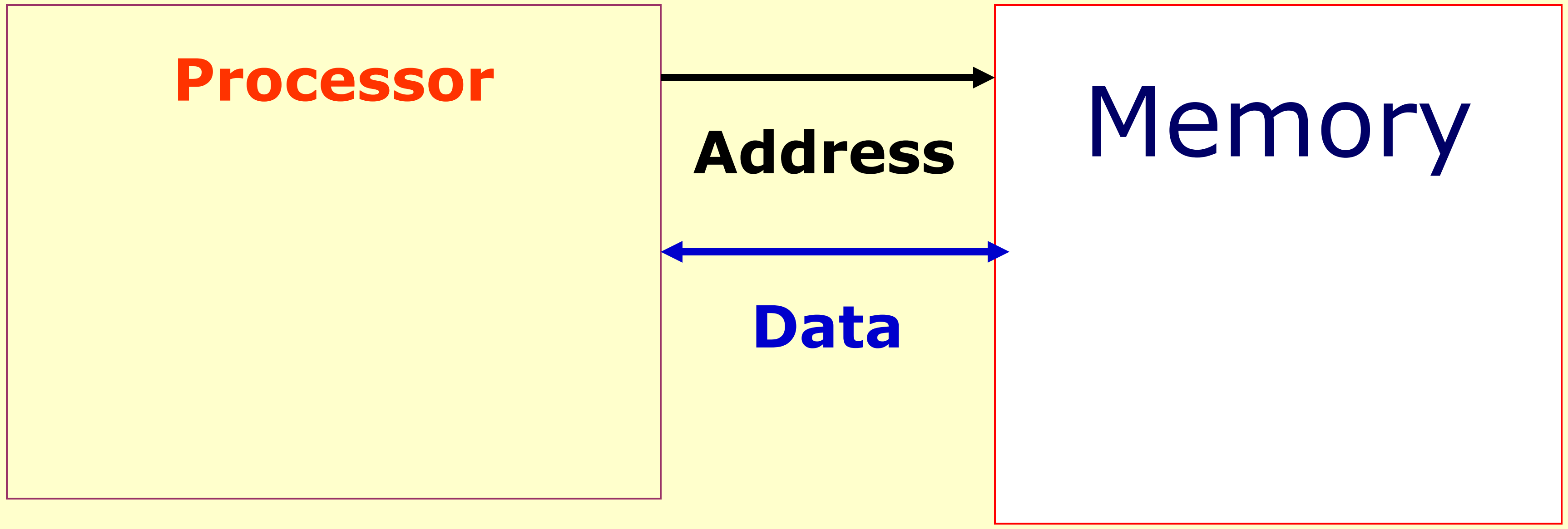
Instruction decoding and execution



Step 3) Processor decodes instruction and places at ID register

Step 4) Processor executes instruction at execution unit

Result Write back and PC update for the next



Step 5) Result of instruction written back for register or memory

Step 6) PC updates for next instruction

Instruction Format

Instruction Format

- An instruction to a processor must specify the

1. *operation* to be done at processor
2. *operands* that are to be used in the operation plus the operand which gives the result of the operation

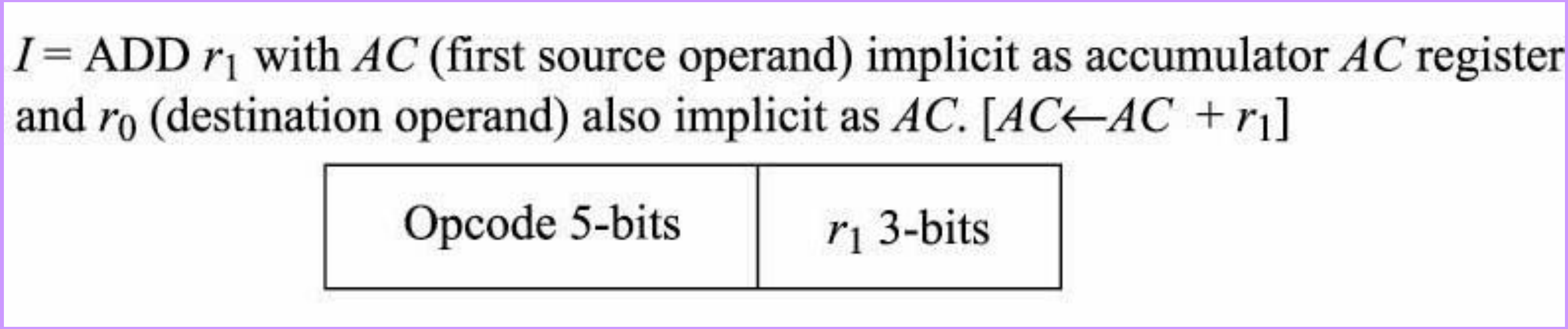
Instruction Word

- Binary *m*-bytes, where *m* = 1or 2 or 3 or 4 or….
- Operation represented by the bits called *opcode* (operation-code)
- Placed in a field, called opcode-field in the instruction, which is fetched by the processor

An Instruction I

I of m-bytes $\longleftarrow\!-\!-\!-\!-\!-\!\longrightarrow$ *n* Bits in opcode's field and (8 × m – n) bits in operands' field, where m = 1, or 2, or 3 or 4 or,…

Example Instruction ADD r1



I = ADD r_1 with AC (first source operand) implicit as accumulator AC register and r_0 (destination operand) also implicit as AC. [$AC \leftarrow AC + r_1$]

Opcode 5-bits	r_1 3-bits

Fetching Opcode

- The processor must fetch the bits placed in a field, called opcode field in the instruction, before the instruction can be executed

Fetching Operands

- The processor also fetches the bits placed in a field, called operand-field in the instruction, before the instruction can execute

Machine codes for all processor-instructions

- The instructions are expressed in a processor-specific language called machine language or assembly language
- An assembly language instruction translates into bits for that instruction, which the processor fetches (called machine codes)

Opcodes and Operands for processor-instructions

- Opcode field always present and can have variable or fixed length
- Variable number of operands

Operands for processor-instructions

- Operands different types
- Registers
- Short memory address
- Long memory address
- Register pointing to memory
- An immediate constant specified at the instruction

Implicit Operand

- An operand may be implicit in an instruction in certain machines (processors), for example, accumulator AC or AC and MQ (multiplicand-quotient register)

Prefixed implicit addresses

- In order to simply the processor circuits and control unit design— a machine (processor) design may prefix implicit addresses

Instructions lengths of *m* number of bytes

- Can depend on the instruction
- Instruction length short when using the register(s) in place of memory address(es)
- short when using the implicit operands

Register to Register-related Operation Format

$$I = \underbrace{\text{operation}}_{\text{opcode}}\ (\underbrace{r_0,\ \ldots,\ r_n}_{\text{operands}})\ \text{,where } n = 0,\ 1,\ 2,\ 3,\ldots$$

Processor instruction Formats

- INTEL-Processor— Destination operand first after the opcode and source operand next
- MOV r1, r2
- The bits from a register r2 are transferred (copied) into r1
- r1 $\leftarrow$ r2

Processor instruction formats

- Motorola -Processor— Source operand first after the opcode and destination operand next
- MOV r2, r1 is when bits from a register r2 are transferred (copied) into r1
- r2$\rightarrow$ r1

An Assumed Convention

- Let us use the convention that unless otherwise stated the destination operand is written first after the opcode and then next source operand

One Address Machine Instruction Format

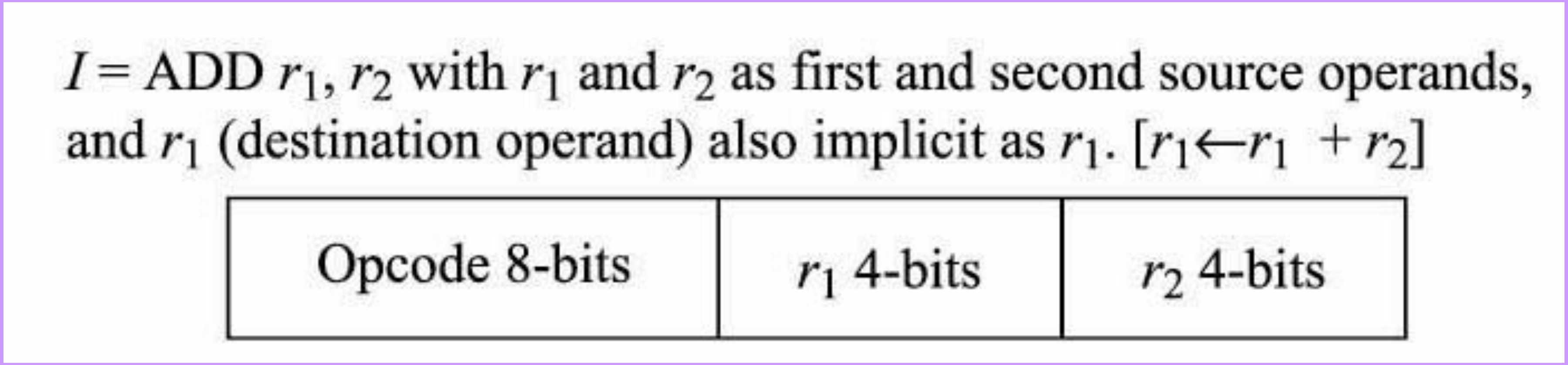
- *I* = ADD r1 with *AC* (first source operand) implicit as accumulator *AC* register and r0 (destination operand) also implicit as *AC*. [$AC \leftarrow AC + r1$]

Opcode 5-bits	r1 3-bits

ADD Instruction Example

- ADD r1, r2 (or ADD r0, r1, r2). [Two or three explicit operand-addresses at the instruction.]
- If register *r* is specified by 4-bits, then ADD r1, r2 can be two-bytes (*m* = 2) long when the opcode field can have 16 – 4 – 4 = 8 bits.

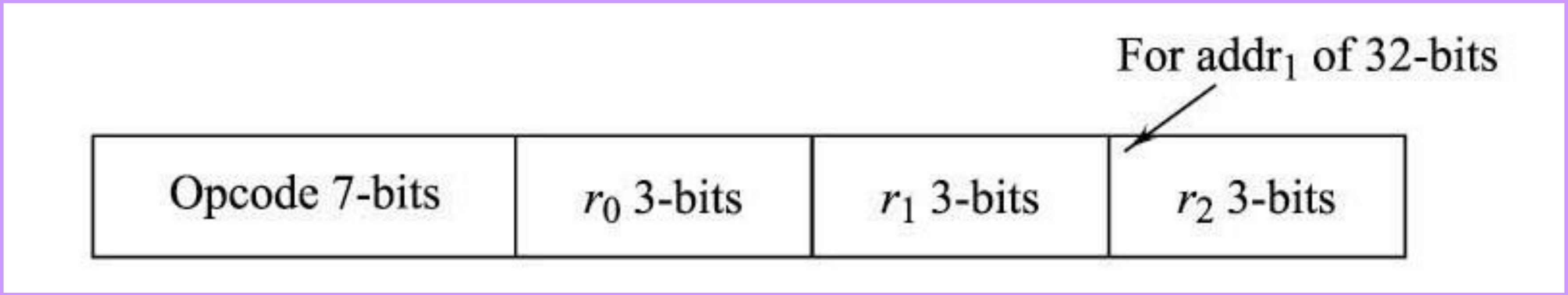
Two Address Machine Register to Register-related Operation Format ADD r1, r2



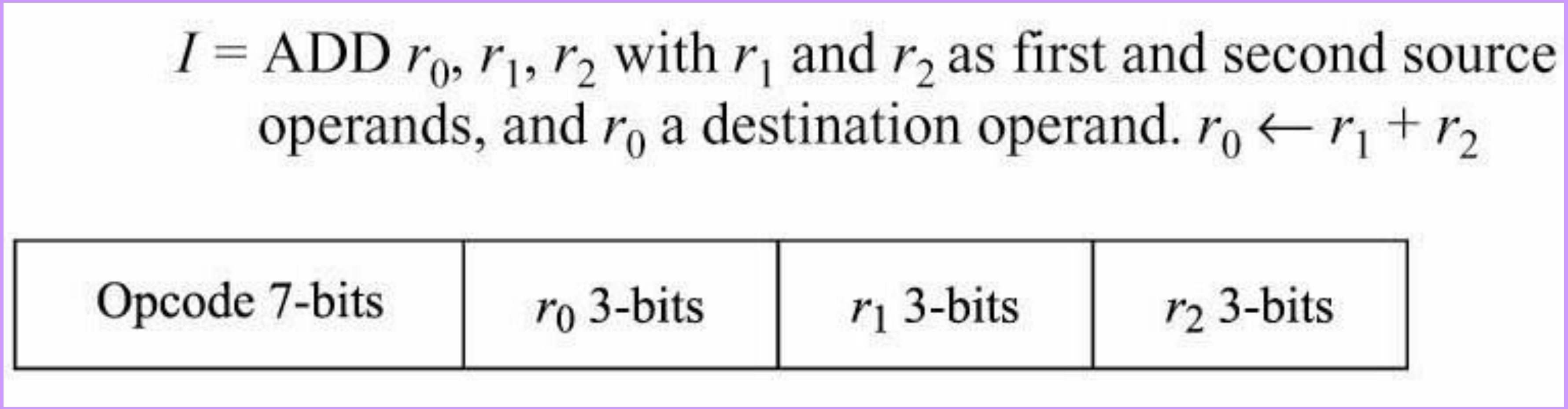
I = ADD r_1, r_2 with r_1 and r_2 as first and second source operands, and r_1 (destination operand) also implicit as r_1. [$r_1 \leftarrow r_1 + r_2$]

Opcode 8-bits	r_1 4-bits	r_2 4-bits

Three Address Machine Instruction Format *I*
ADD r1, r2, (r3)



Three address machine



I = ADD r_0, r_1, r_2 with r_1 and r_2 as first and second source operands, and r_0 a destination operand. $r_0 \leftarrow r_1 + r_2$

Opcode 7-bits	r_0 3-bits	r_1 3-bits	r_2 3-bits

Zero Address Machine Instruction Format

- ADD [Both source operands popped from on stack and result back to stack]

Opcode 8-bits

Summary

We learnt

- Basic Instructions
- Opcode and operand fields
- Instruction Formats of Instructions in 0, 1, 2 and 3 address machines

End of Lesson 03

Basic operations and Instruction Formats