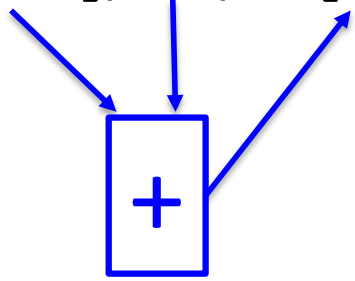
1.2 Instructions and Programs

- An instruction specifies an operation and the locations of its data operands; e.g.:
  - Concept: Total := Total + Sum
  - Assignment for variables Total and Sum
    - Need physical space
    - R6 := Sum; M[200] := Total
    - R6 is a register, Register-#6
    - M[200] is content of memory location, 200

Instruction Operation

- Operation: Total := Total + Sum
  - M[200] := M[200] + R6 (pseudo instruction)
- Assembly instruction:
  - ADD M[200],R6,M[200] ; in,in,out



; indicates comment

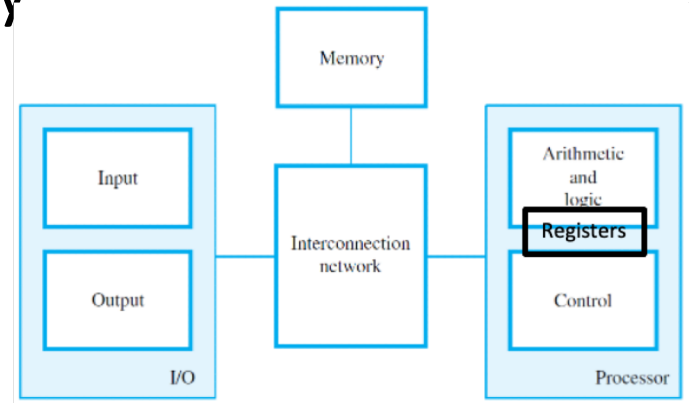
Operation Steps

- Instruction:
  - ADD M[200],R6,M[200]
    ; M[200] := M[200] + R6

'write' changes the content !

- Operation steps:
  1. Get M[200]; read from memory
  2. Get R6; read from register
  3. Add; operation in ALU
  4. Put M[200]; write to memory

How about 'read' ?



1.2 Instructions and Programs

- A sequence of instructions, executed one after another, constitutes a program: e.g.,
  - Swaps two adjacent elements in an array, **HLL**:

```
Swap(int v[], int k)
  {int temp;
     temp = v[k]
     v[k] = v[k+1]
     v[k+1] = temp;
  }
```

HLL = High Level Language
That is, at Conceptual Level

Patterson/Hennnessy Computer Organization & Design

Swap in Assembly

- Swap: (will discuss the green parts later)

```
LSL     X10, X1, 3          ; setting up pointers
ADD     X10, X0, X10        ; pointing to array
LD      X9, [X10,0]         ; Load M[X10+0] to X9
LD      X11, [X10,8]        ; Load M[X10+8] to X11
ST      X11, [X10,0]        ; Store X11 to M[X10+0]
ST      X9, [X10,8]         ; Store X9 to M[X10+8]
BR      X10                 ; branch to main
```

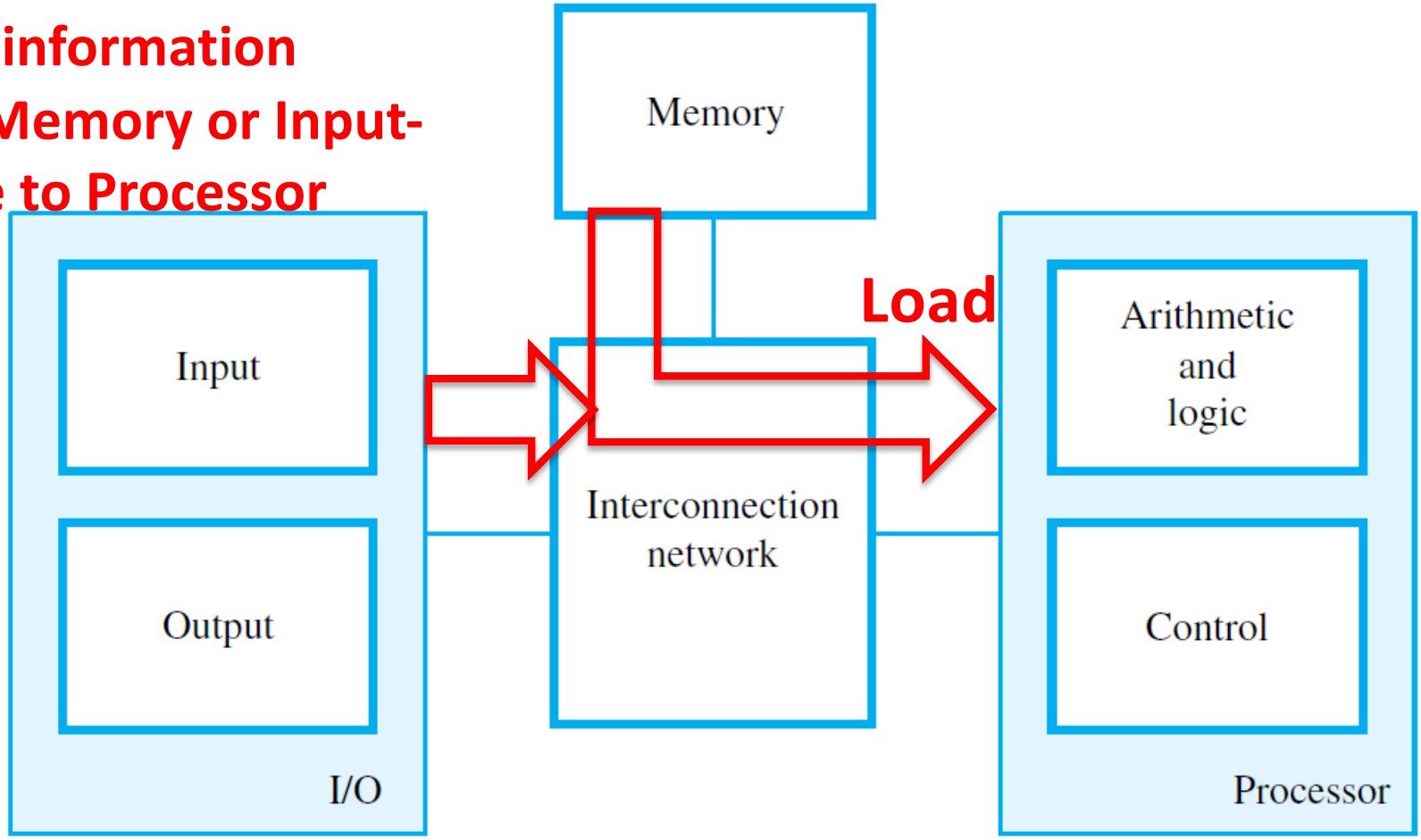
Patterson/Hennnessy Computer Organization & Design

Instruction Types

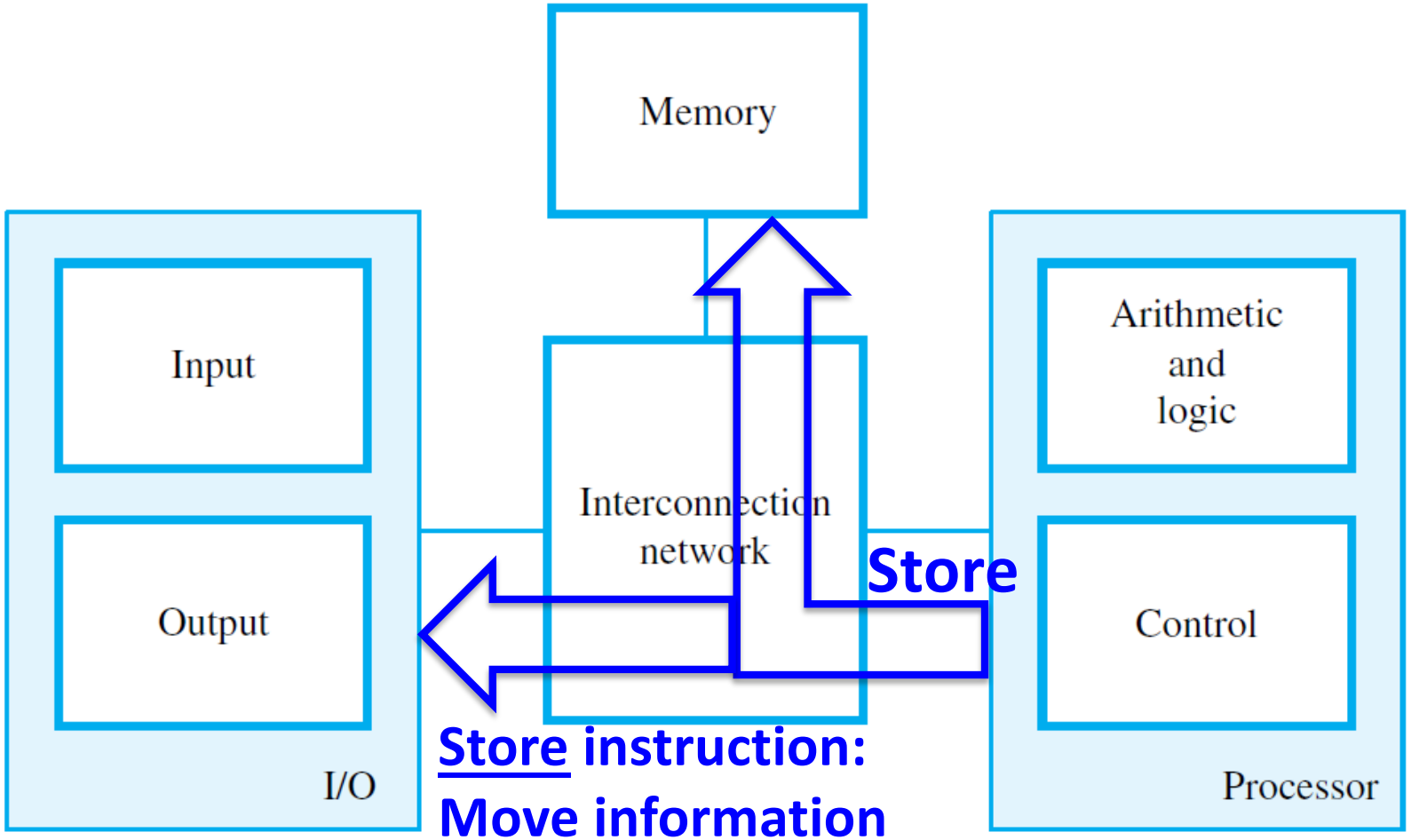
1. **Load** : Transfer
   - memory or input-device ⟶ processor register
2. **Store** : Transfer
   - processor register ⟶ memory or output-device
3. **Operate** : Perform calculation
   - arithmetic or logic operation

Instruction Types: Load; Store; Operate

Load instruction:
Move information from Memory or Input-device to Processor



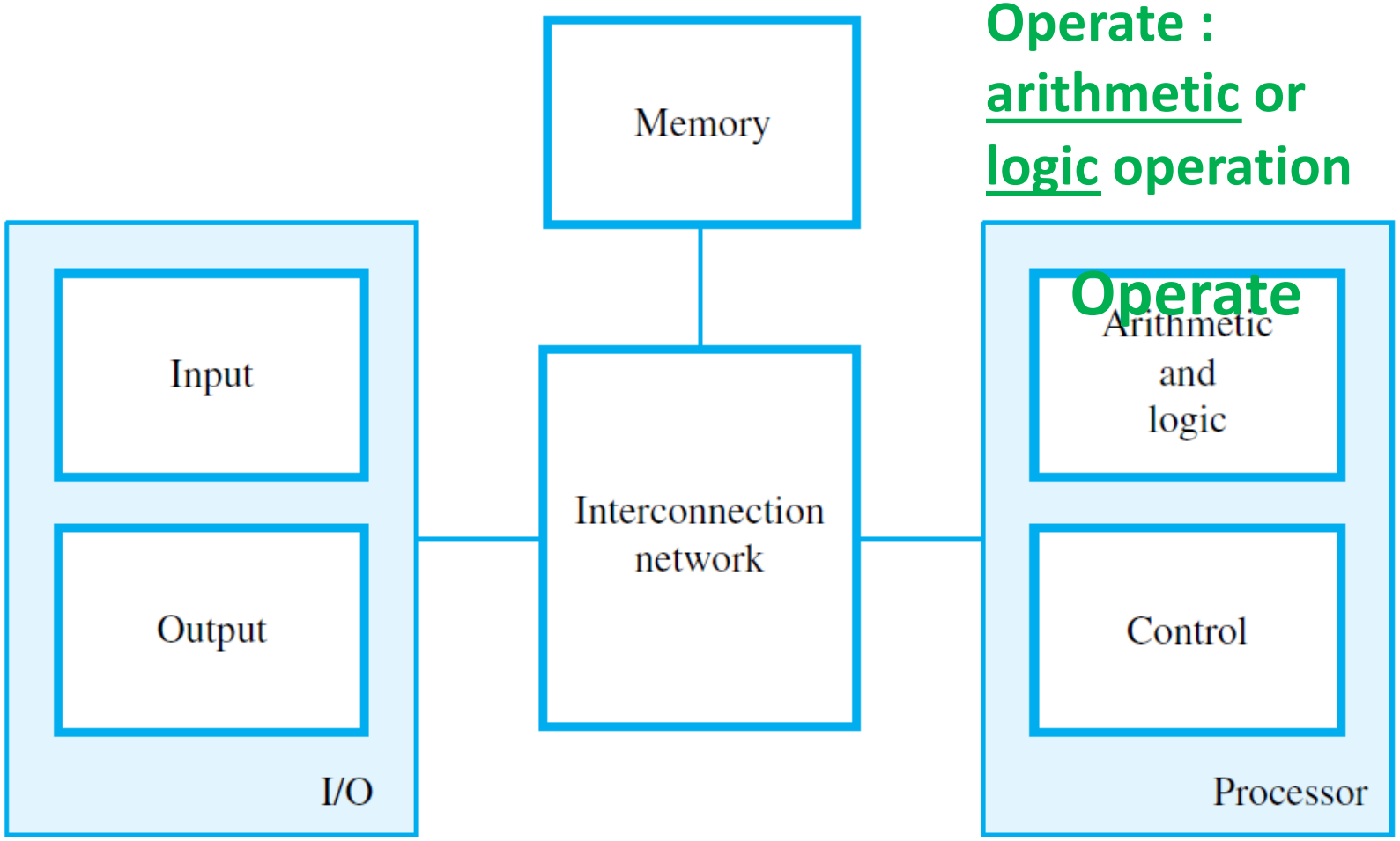
Instruction Types: Load; Store; Operate



Store instruction:
Move information
From Processor to Memory or Output-device

Instruction Types: Load; Store; Operate

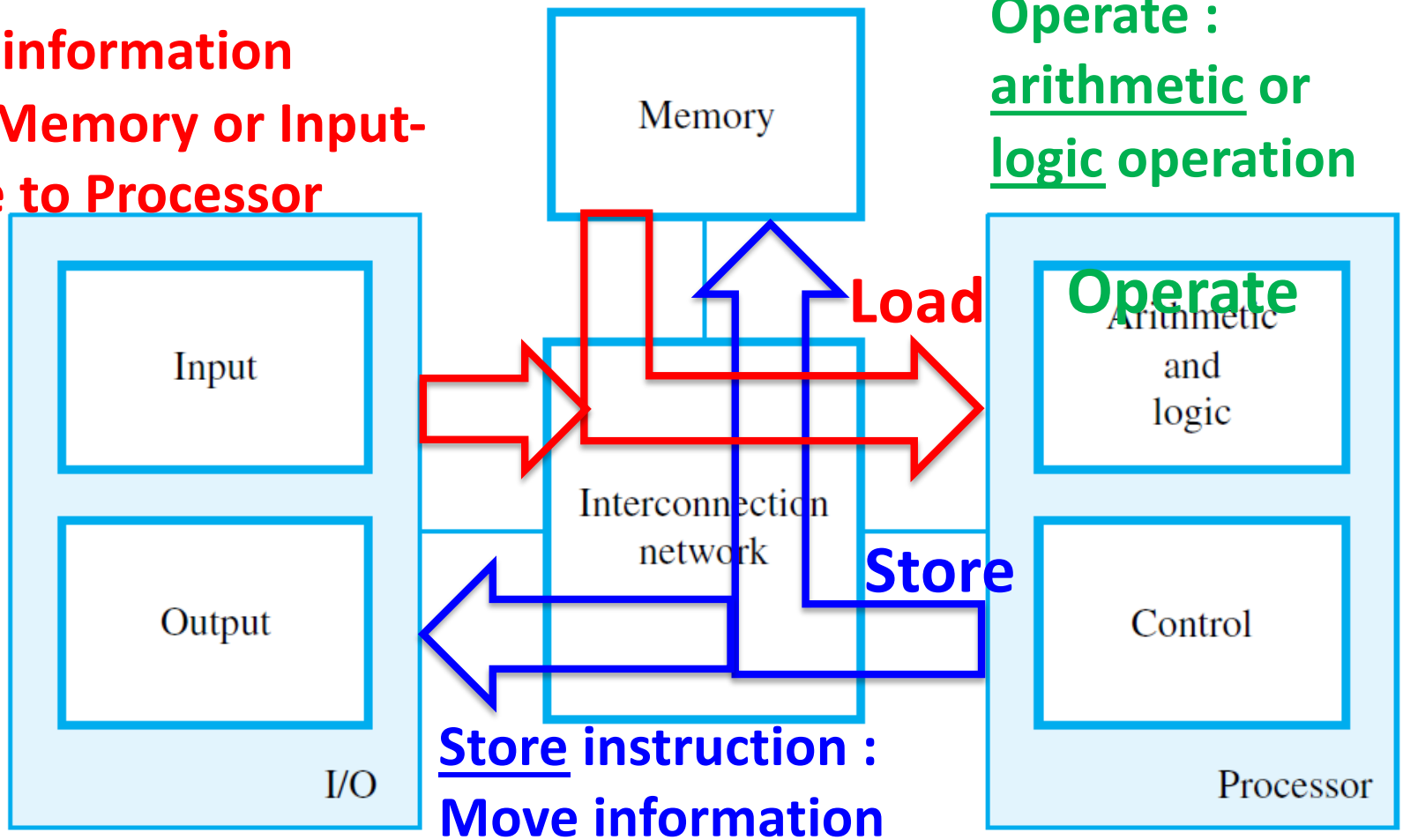
Operate : arithmetic or logic operation



Instruction Types: Load; Store; Operate

Load instruction :
Move information
From Memory or Input-device to Processor

Operate :
arithmetic or logic operation



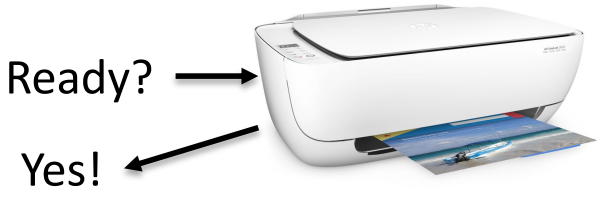
Store instruction :
Move information
From Processor to Memory or Output-device

Handling I/O devices

An application program can:

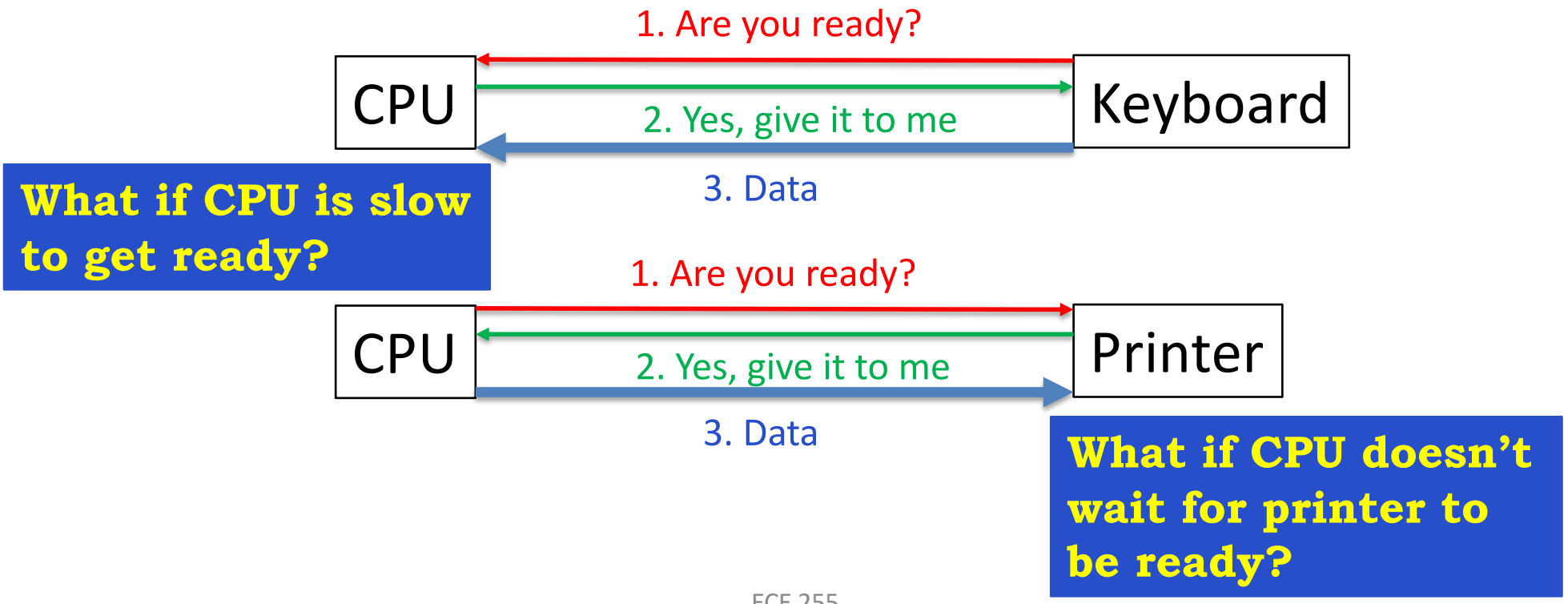
- Read data (e.g., a number)
  from an input device (keyboard)
- Write data (e.g., an alphabet)
  to an output device (monitor))

- Need to sense the readiness of an I/O device
  to perform a transfer



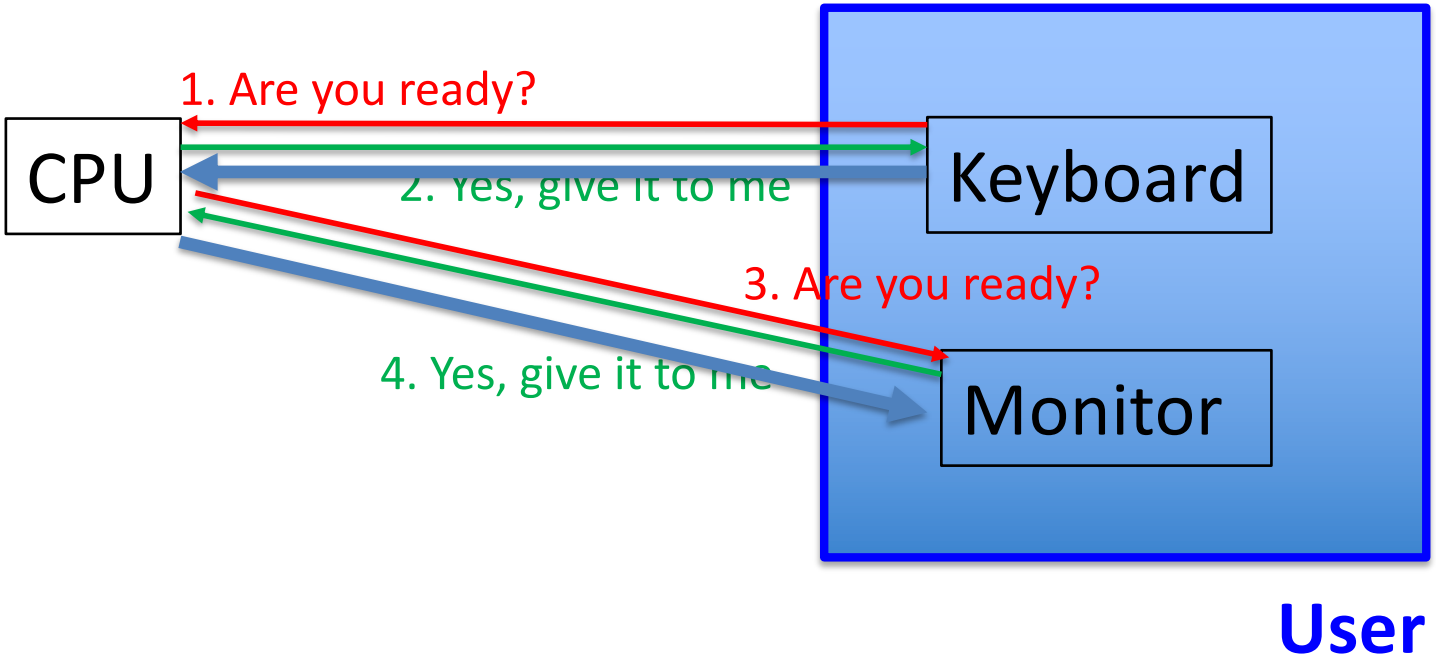
Handshaking

- **Handshaking** is the process by which two devices initiate communications to synchronize.



Echo

- **Echo** when
  1. data is sent to a computer or device
  2. that information is sent back to verify the information has been received correctly



Memory: Size k X 1 Byte

Given:

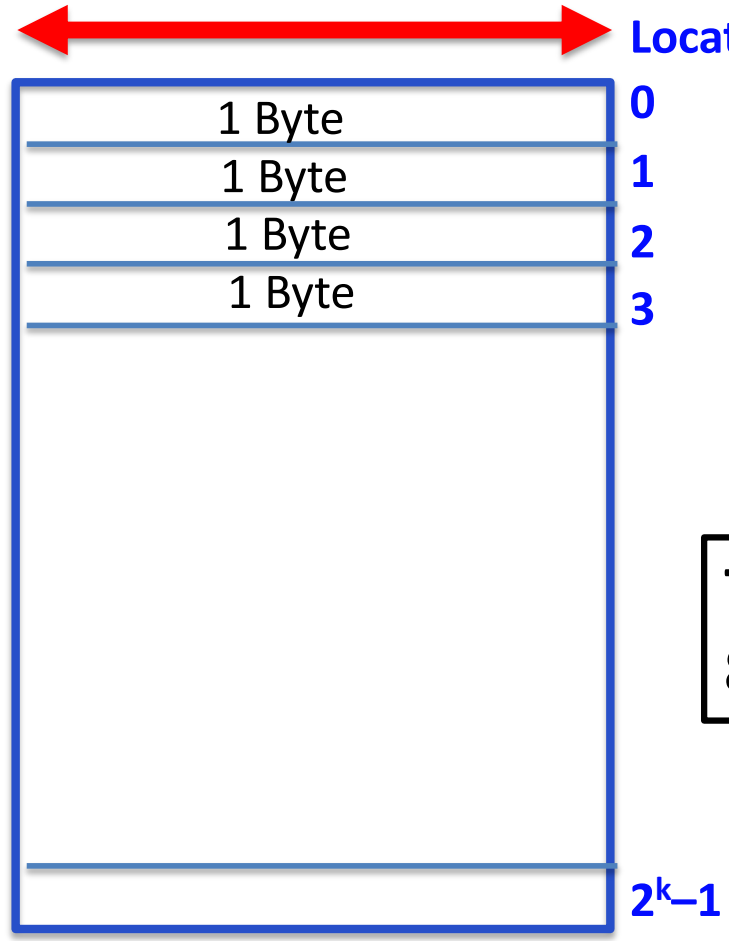
- 2^k location
- 1 Byte in each of the 2^k locations or addresses

Max amount of memory if k=10?

Amount if k=20?

Amount if k=30?

Amount if k=32?



What is k?

Number of bits on Address Bus

Typical Address has 8, 16, 32, 64 bits

Not necessarily Power of 2

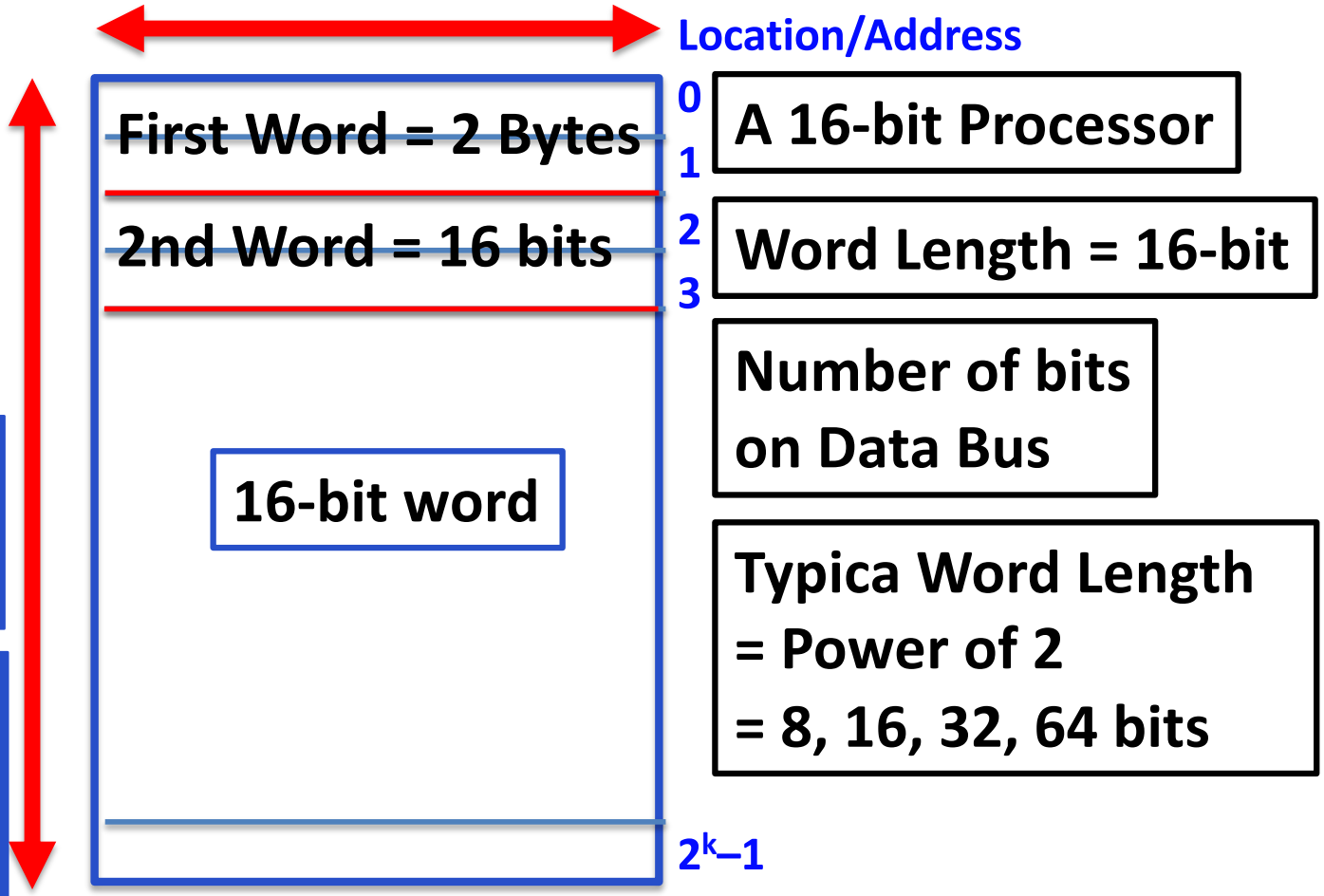
Memory: Word Length

Given:

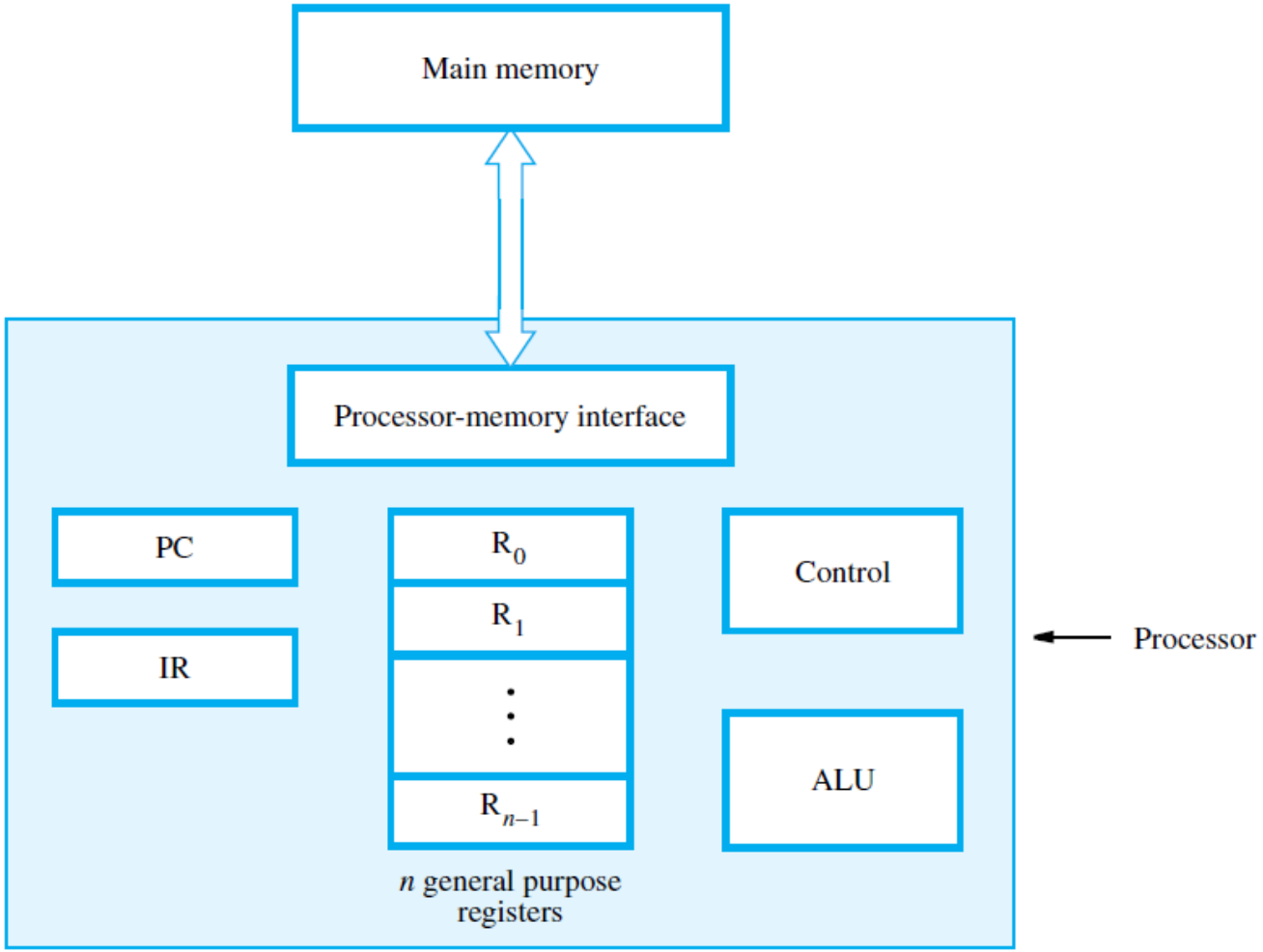
- 2^k location/byte
- 1 Byte in each of the 2^k locations or addresses

4th Word occupies which Addresses or Locations?

4th Word of a 32-bit processor occupies which Addresses or Locations?



Processor Memory Interaction

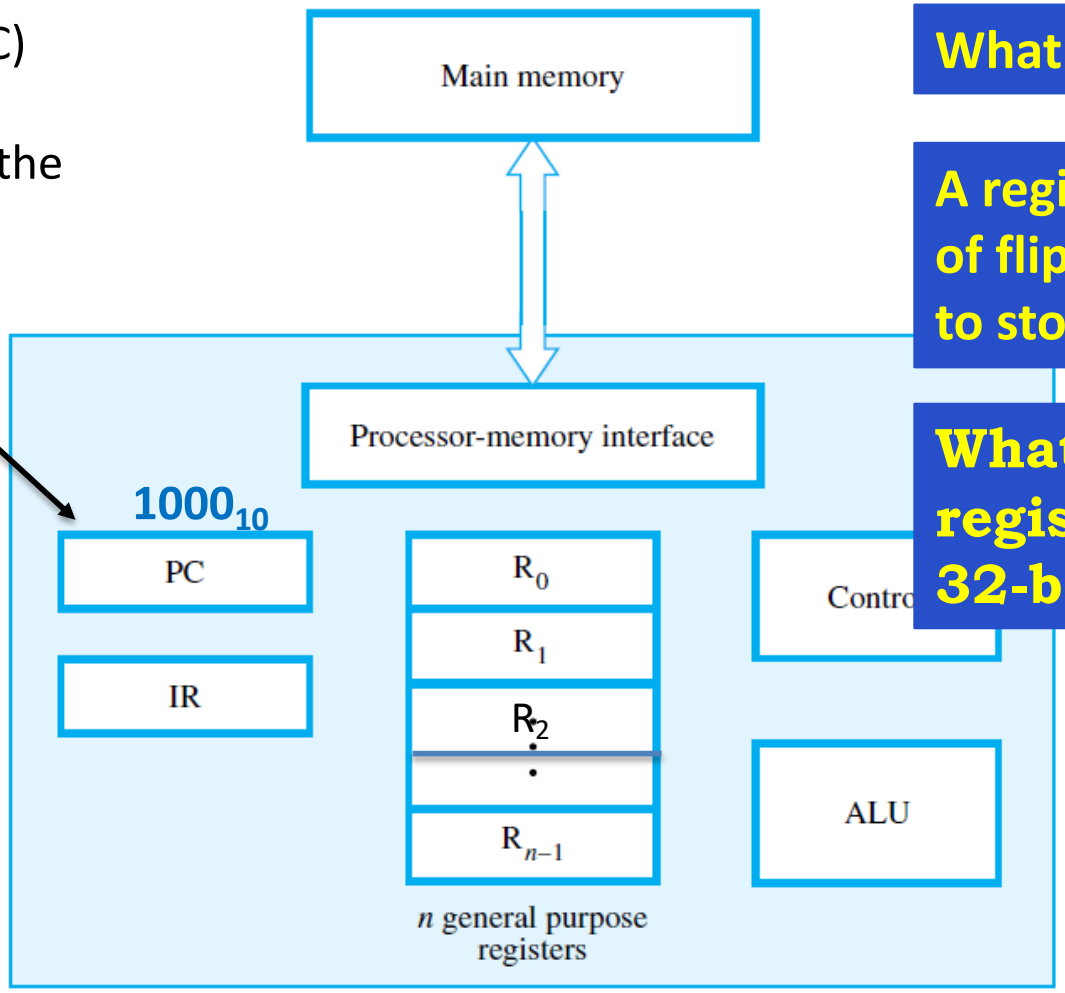


Program Counter

Program counter (PC) **register** holds the memory address of the current instruction:
e.g., **1000_{10}**

Subscript 10 or none:
- base 10

Subscript 2:
- binary base 2

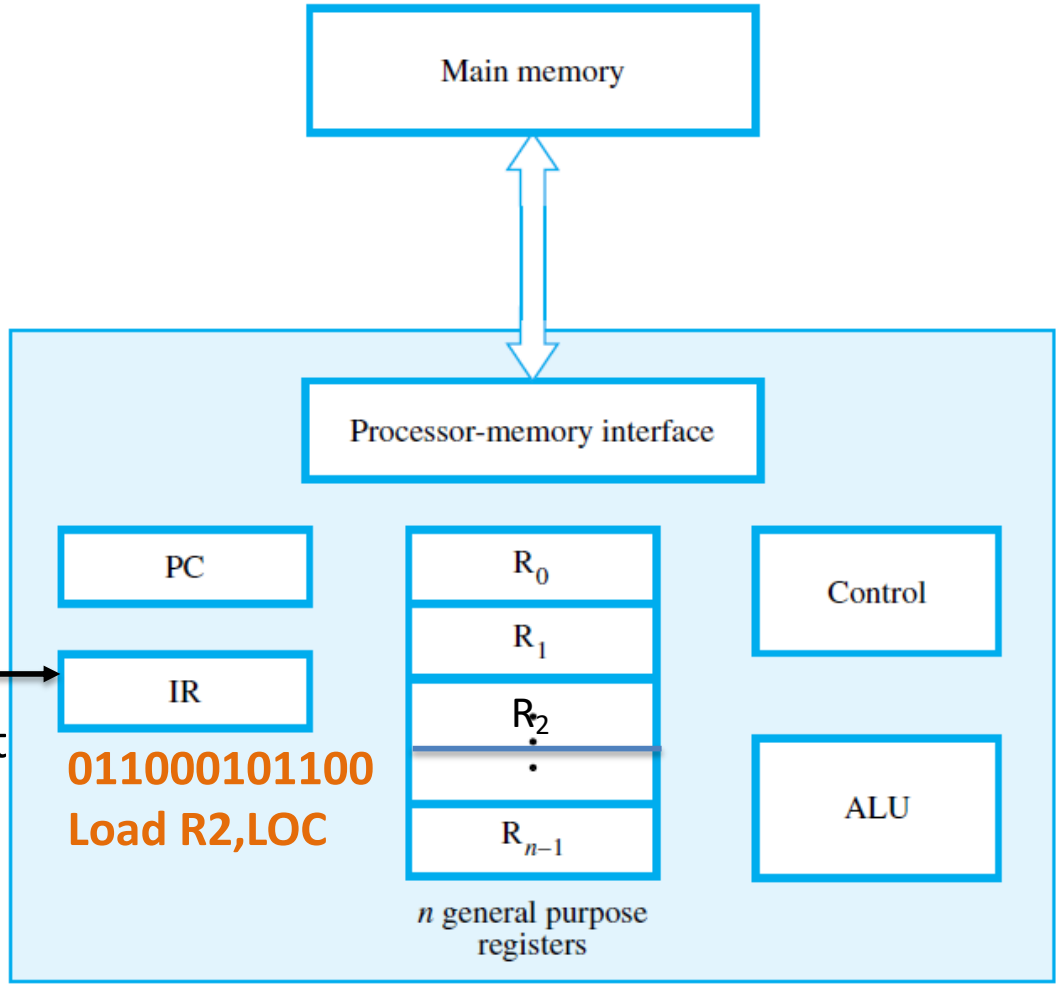


What is a register?

A register is a collection of flip flops (1-bit) used to store binary data

What is the register size of a 32-bit processor?

Instruction Register

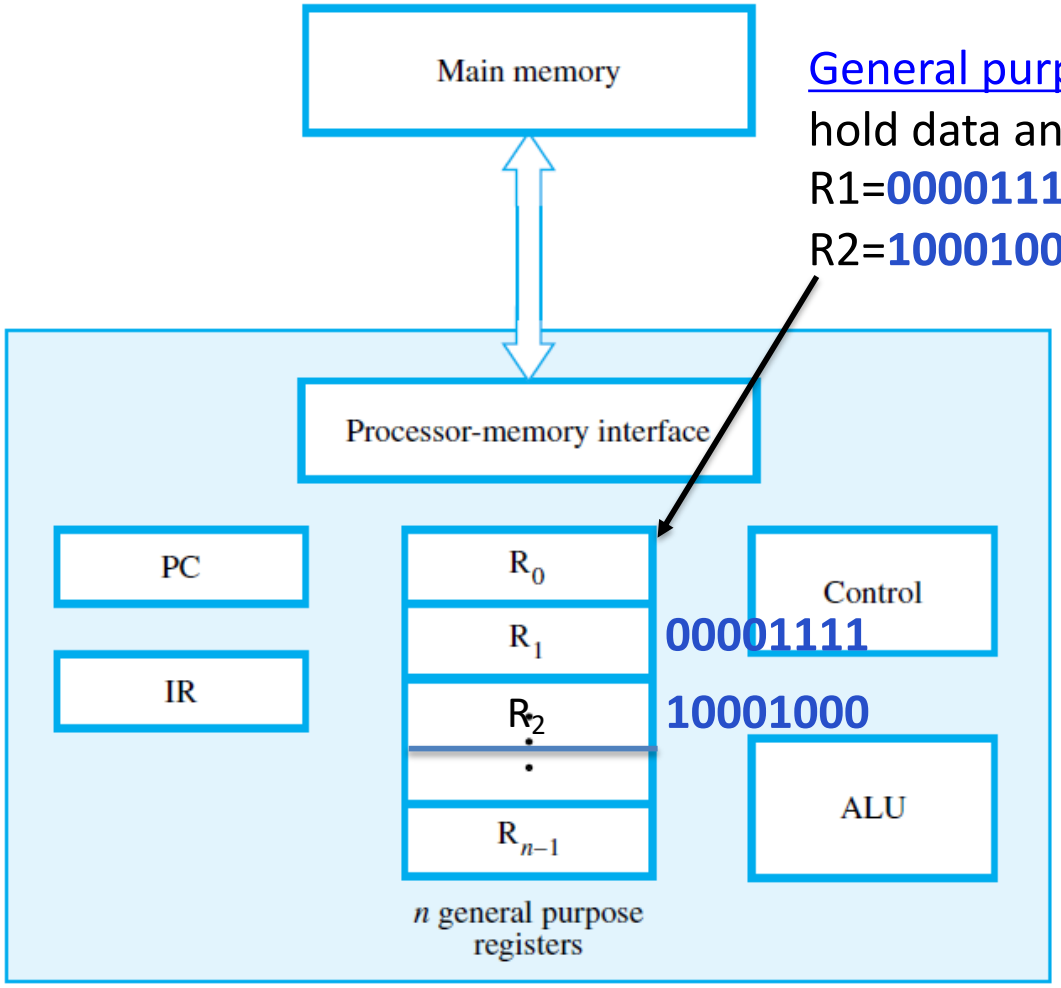


A register is a collection of flip flops (1-bit) used to store binary data

Instruction register (IR) holds the current instruction:
e.g., **011000101100 = Load R2,LOC**

011000101100
Load R2,LOC

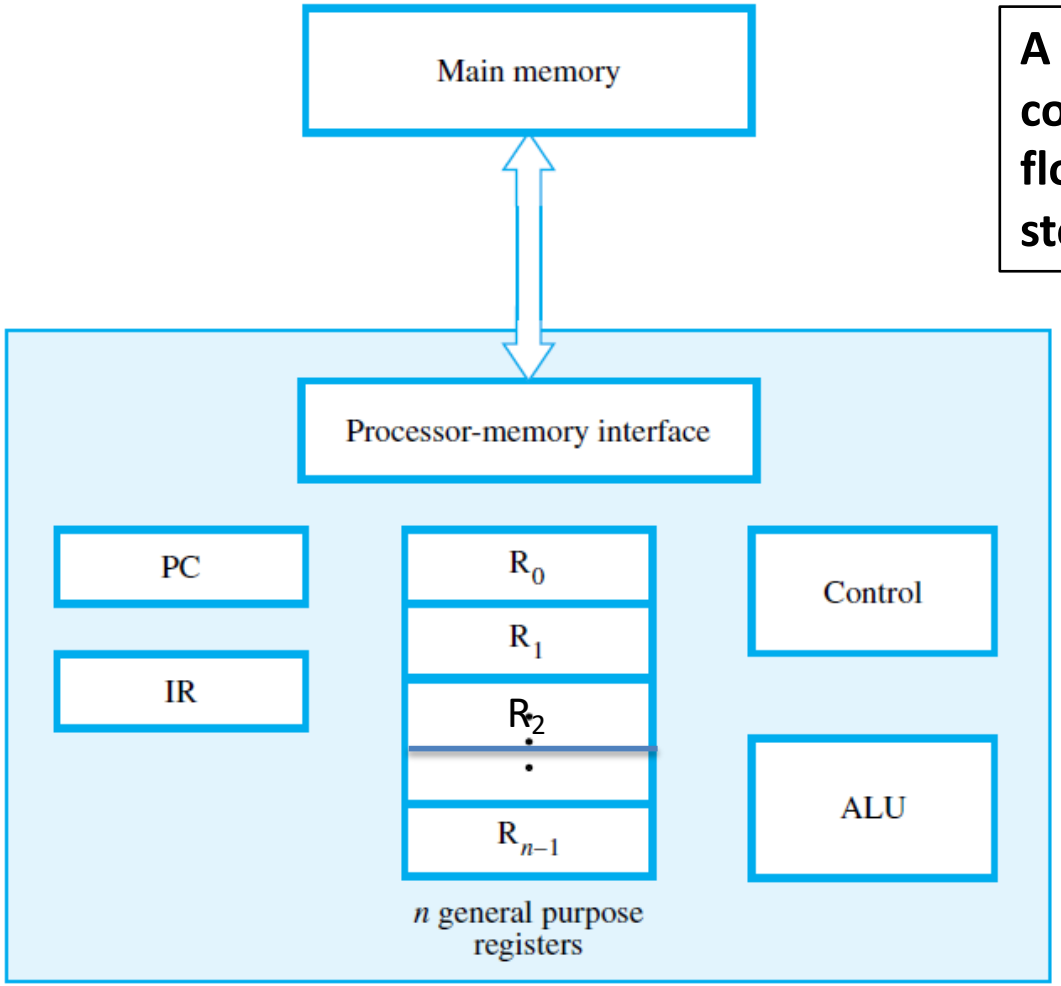
General Purpose Registers



General purpose registers R_0 to R_{n-1} hold data and addresses: e.g.,
R1=**00001111** (data or address)
R2=**10001000** (data or address)

A register is a collection of flip flops (1-bit) used to store binary data

Control Circuits



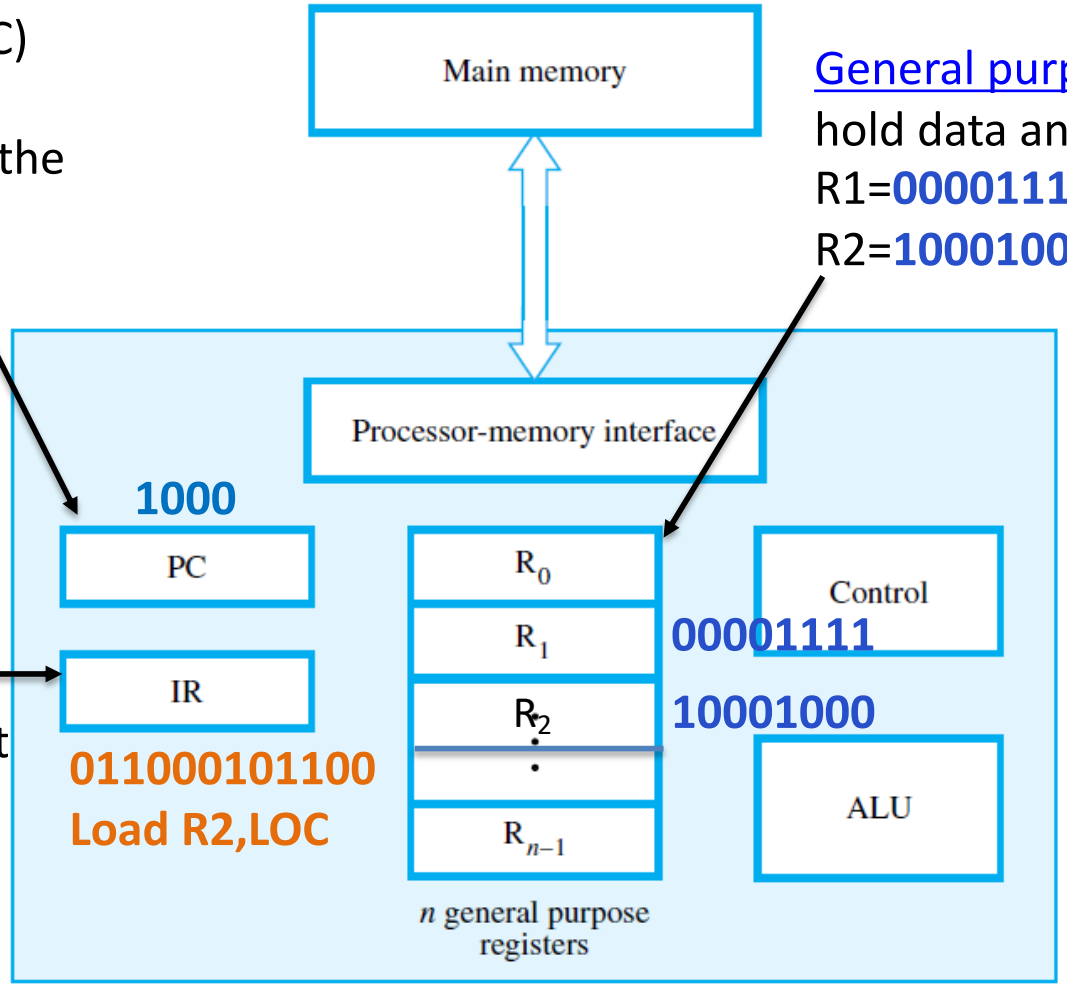
A register is a collection of flip flops (1-bit) used to store binary data

Control circuits for Instruction Cycle: fetch and execute instructions

Processor Registers

Program counter (PC) register holds the memory address of the current instruction:
e.g., **1000 $_{10}$**
Subscript 10: base 10
No subscript: binary base 2

Instruction register (IR) holds the current instruction:
e.g., **011000101100 = Load R2,LOC**



General purpose registers R_0 to R_{n-1} hold data and addresses: e.g.,
R1=**00001111** (data or address)
R2=**10001000** (data or address)

Control circuits for Instruction Cycle: fetch and execute instructions

A register is a collection of flip flops (1-bit) used to store binary data

Fetching and Executing Instructions

- Example: Inst#1@ Address 1000_{10}

  $M[1000_{10}]$ = Load R2,LOC = 0110 0010 1100

	Operation	To Register	From Memory
$M[1000_{10}]$ =	0110 = Load	0010 = 2 (reg#2)	1100 = $M[1100_{10}]$

Fetching and Executing Instructions

- Example: Inst#1@ Address 1000

  Load R2, LOC

	Operation	To Register	From Memory
M[1000]=	0110	0010	1100
	=Load	=2 (reg#2)	=M[1100]

Where is the instruction?

- The processor control circuits do the following:

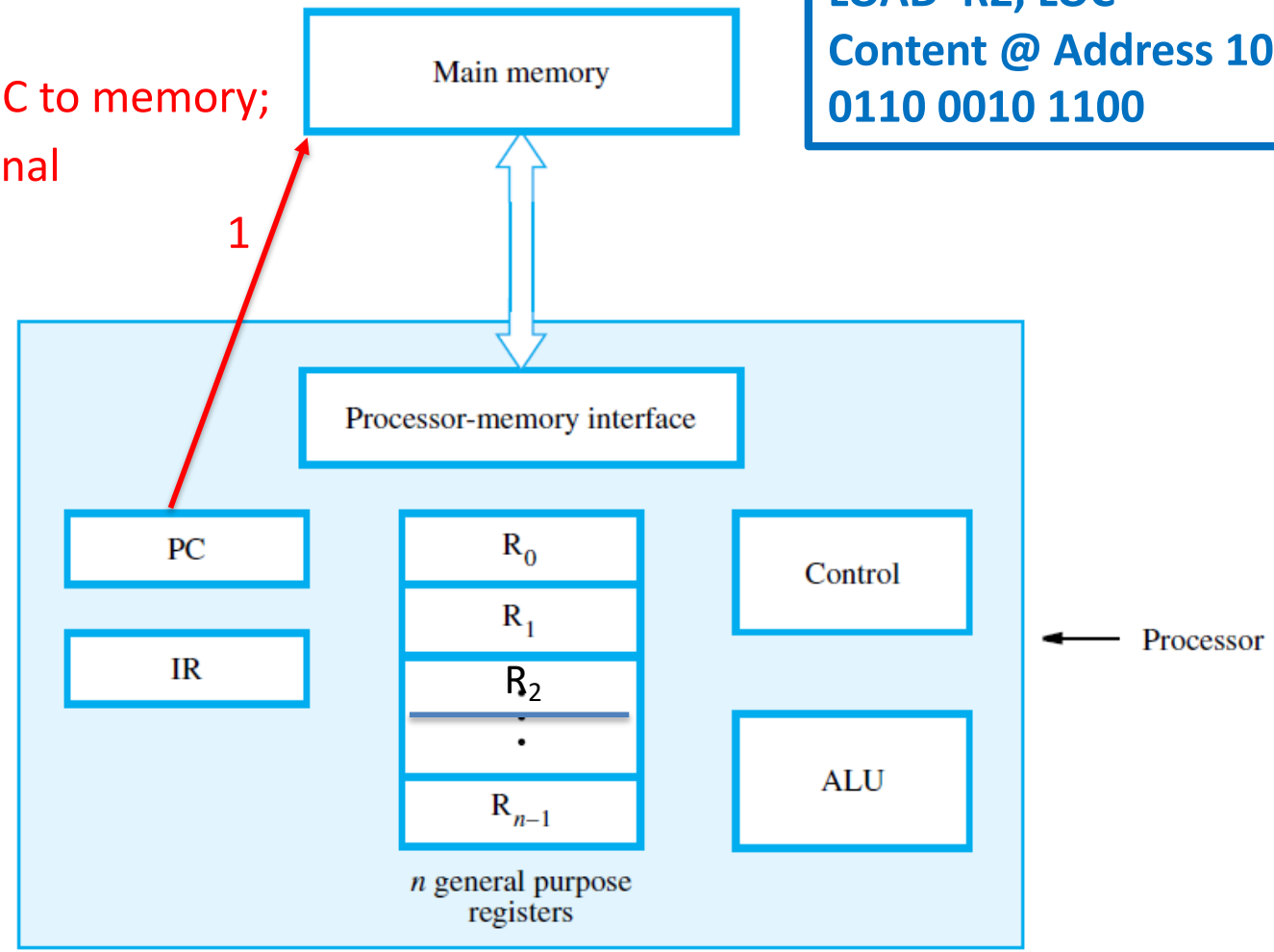
1. Send 1000_{10} (address) from PC to memory; issue Read-Mem signal
2. Load instruction (0110 0010 1100) from memory into IR
3. Increment PC to point to next instruction (now 1004_{10})
4. Send 1100_{10} (address LOC in part of IR) to memory; issue Read-Mem signal
5. Load data (XXXX) from memory address 1100_{10} into register R2
   - X means "don't care" or "not known"

Where is the data?

Instruction Cycle: step 1

LOAD R2, LOC
Content @ Address 1000_{10}
0110 0010 1100

1. Send address in PC to memory; issue Read-Mem signal

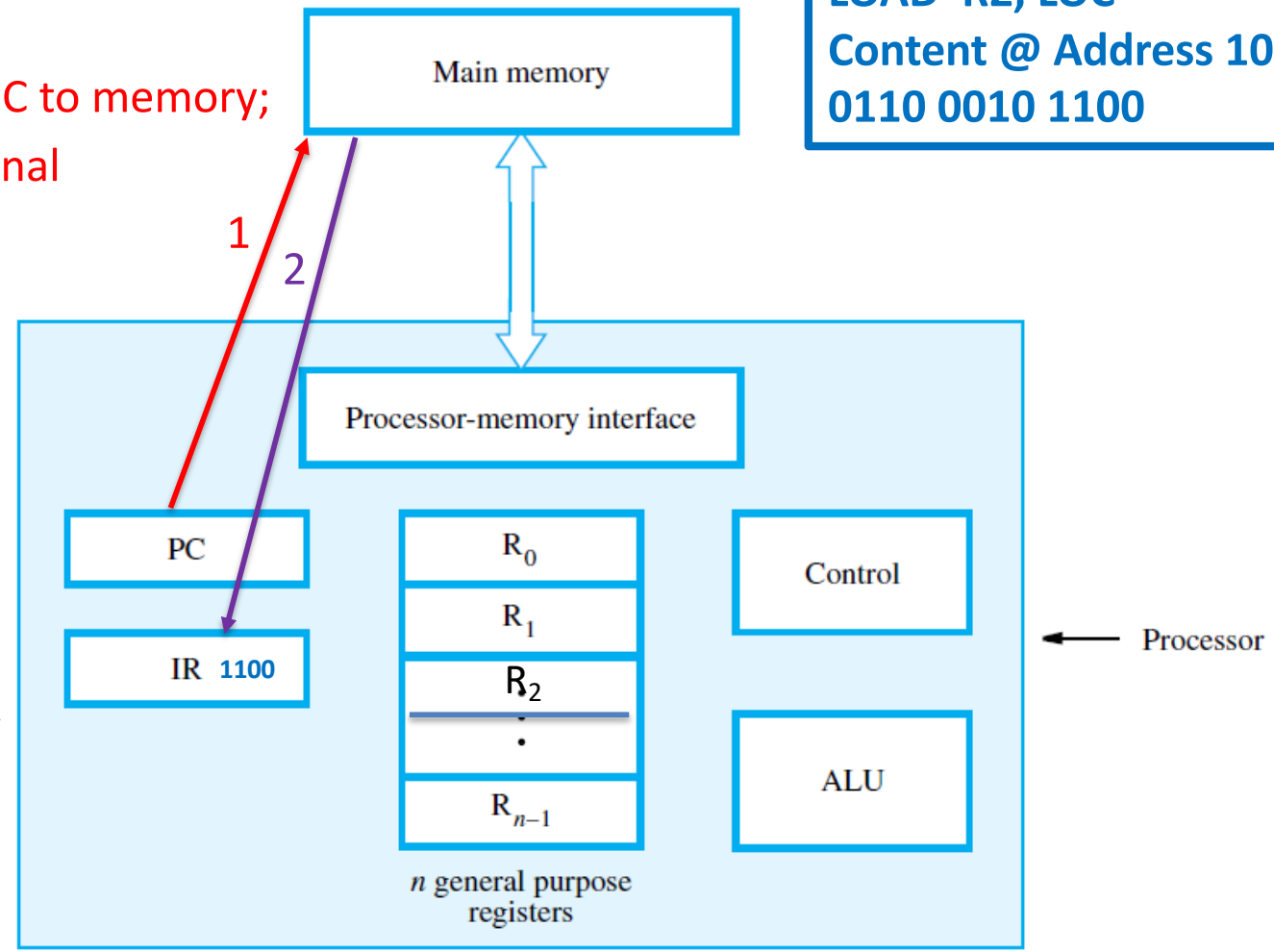


Instruction Cycle: step 2

LOAD R2, LOC
Content @ Address 1000_{10}
0110 0010 1100

1. Send address in PC to memory; issue Read-Mem signal

2. Load instruction from memory into IR



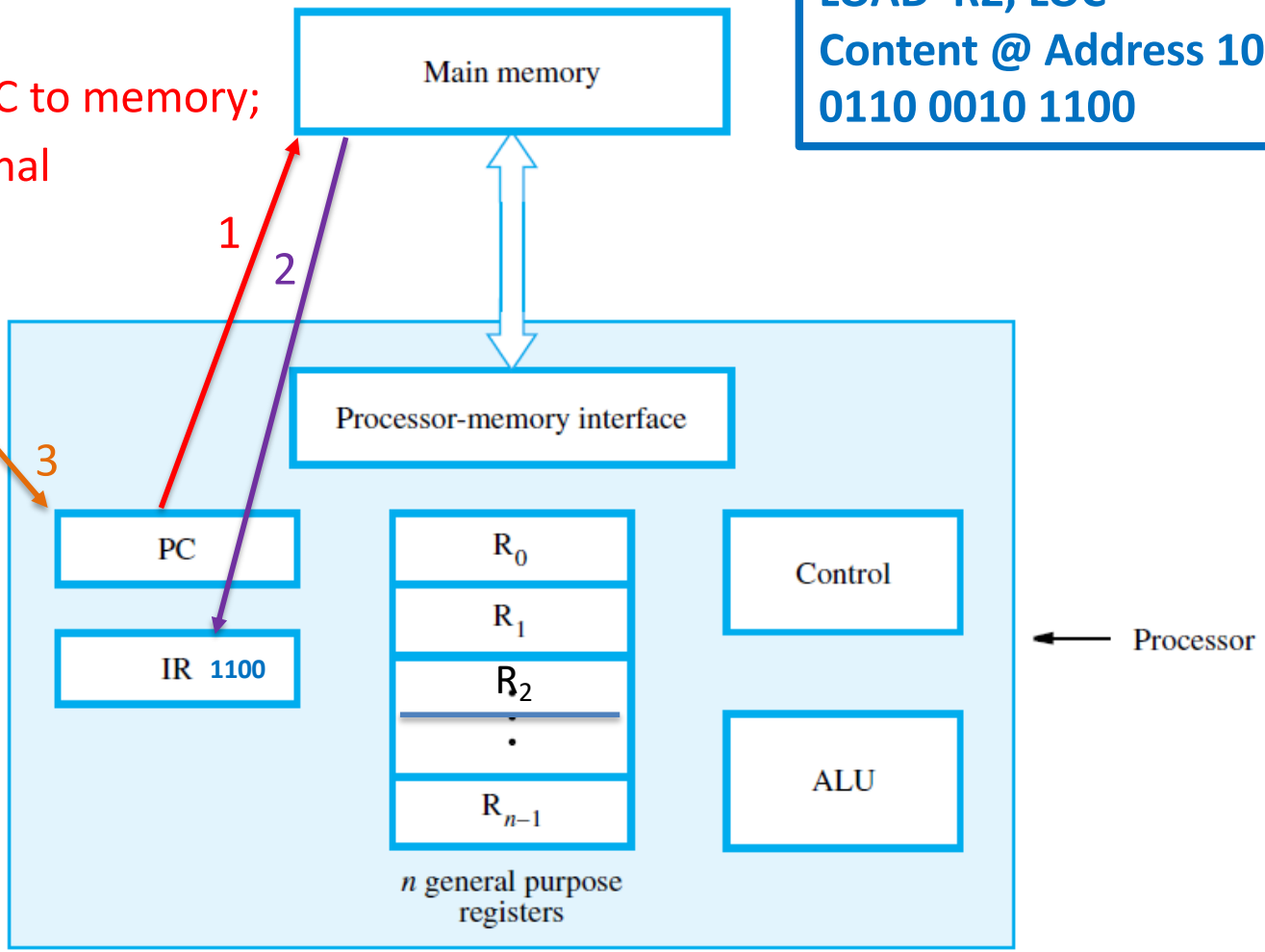
Instruction Cycle: step 3

LOAD R2, LOC
Content @ Address 1000_{10}
0110 0010 1100

1. Send address in PC to memory; issue Read-Mem signal

3. Increment PC (+X, word length) to point to next instruction

2. Load instruction from memory into IR



Instruction Cycle: step 4

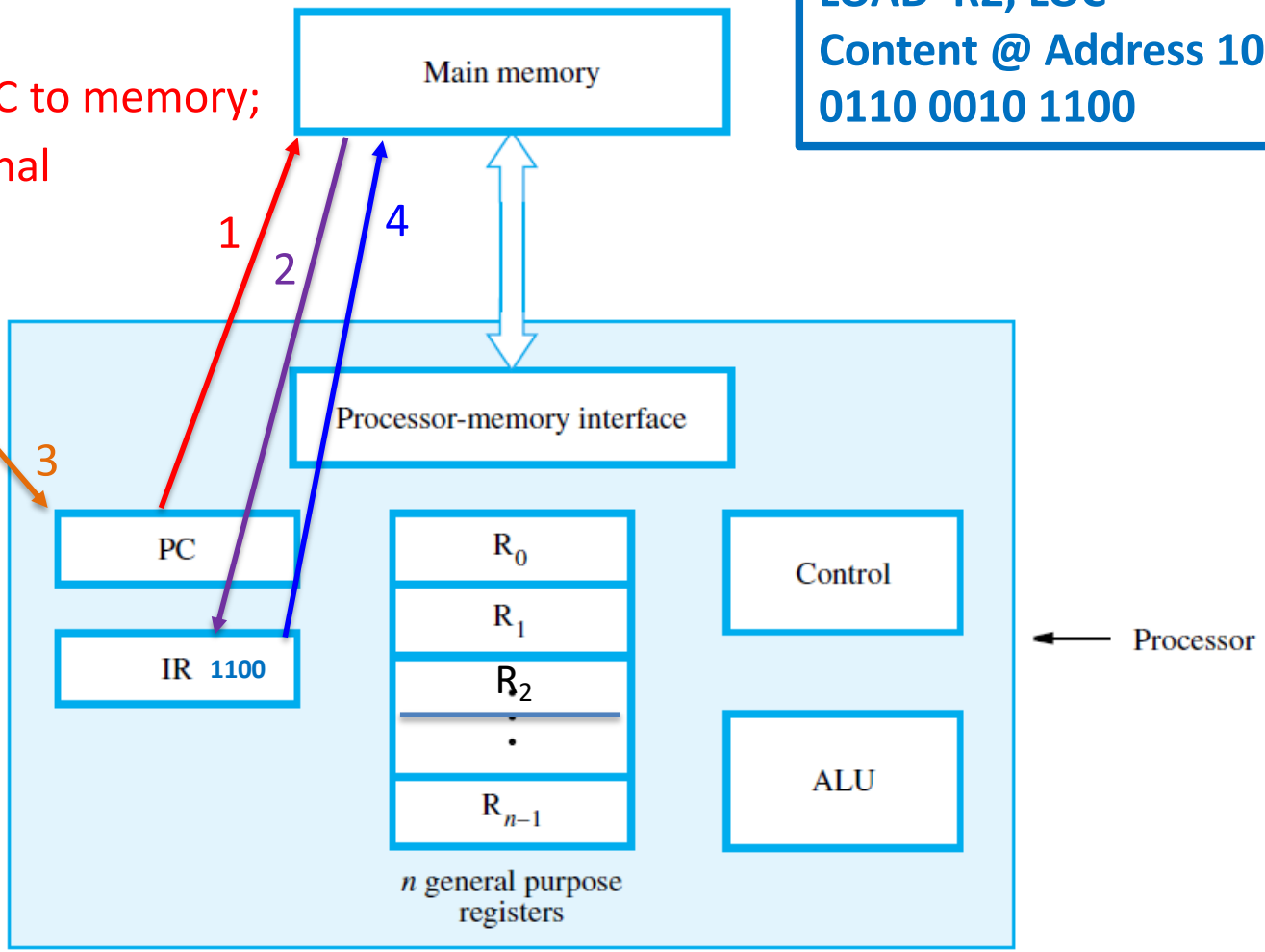
LOAD R2, LOC
Content @ Address 1000_{10}
0110 0010 1100

1. Send address in PC to memory; issue Read-Mem signal

3. Increment PC (+X, word length) to point to next instruction

2. Load instruction from memory into IR

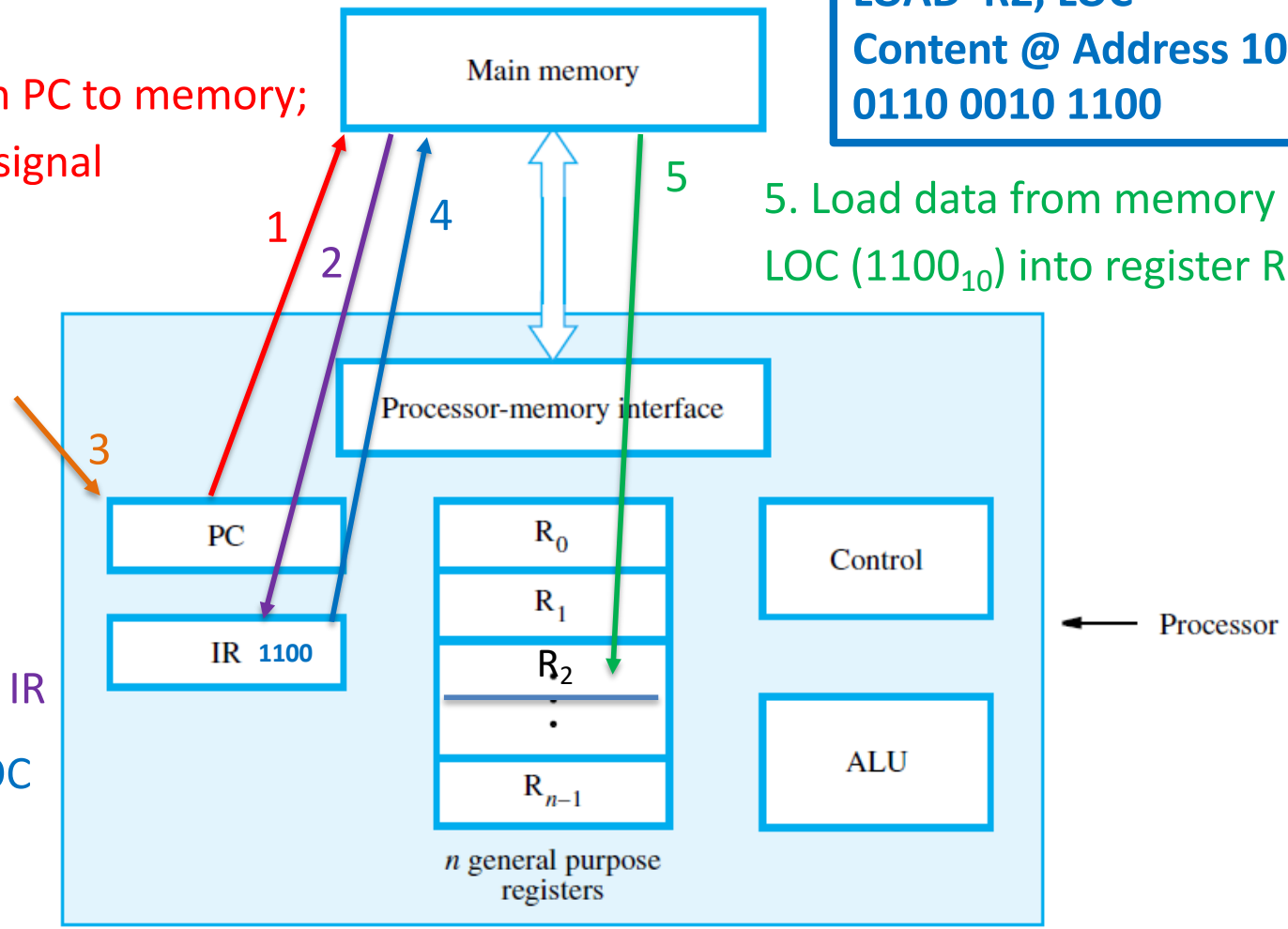
4. Send address LOC to memory; issue Read-Mem signal



Instruction Cycle: step 5

LOAD R2, LOC
Content @ Address 1000_{10}
0110 0010 1100

1. Send address in PC to memory; issue Read-Mem signal



5. Load data from memory LOC (1100_{10}) into register R2

3. Increment PC (+X, word length) to point to next instruction

2. Load instruction from memory into IR

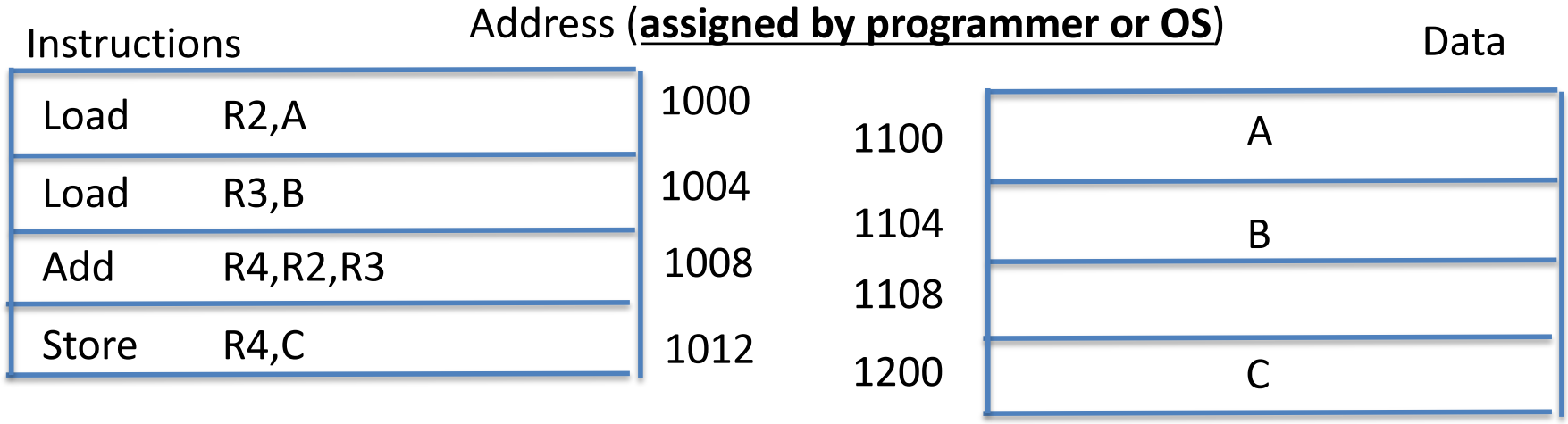
4. Send address LOC to memory; issue Read-Mem signal

Assembly Program Example

- A program for the calculation with these variables

  C = A + B

- Variables A, B, and C, are symbols assigned to memory addresses
- Symbol: human readable form of variable, constant, etc.
- R_i's are general purpose registers



Instruction Operation

Instructions		Address (assigned)
Load	R2,A	1000
Load	R3,B	1004
Add	R4,R2,R3	1008
Store	R4,C	1012

Address (assigned)	Data
1100	A
1104	B
1108	
1200	C

- R2 ← M[1100]
- R3 ← M[1104]
- R4 ← R2 + R3
- R4 → M[1200]

1.3 Operational Concept

- High-level languages: C, C++, etc.
  - Design and Implement at
    - Conceptual / Algorithmic Level
- Assembly language: specific to each processor
  - Design and Implement at
    - Machine / Hardware Level

Semantic Gap

High Level Language (HLL)

Conceptual / Algorithmic Level

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

a. Translate into Assembly

How good is the translation?

GAP

Compiler

Assembly language program (for ARMv8)

```
swap:
      LSL  X10, X1,3
      ADD  X10, X0,X10
      LDUR X9, [X10,0]
      LDUR X11,[X10,8]
      STUR X11,[X10,0]
      STUR X9, [X10,8]
      BR   X10
```

b. May then be Optimized

Machine / Hardware Level

Assembly

1. Is translated & optimized Assembly an exact match to the HLL/Concept ?
2. Matching to the intention of the user ?

https://clipartxtras.com
ps://www.britannica.com/technology/computer
Hennnessy Computer Organization & Design