

# Chapter 15

## Control Unit Operation

---

**2020.6**  
**Howon Kim**

정보보호 및 지능형 IoT 연구실 - <http://infosec.pusan.ac.kr>  
부산대 지능형융합보안대학원 - <http://aisec.pusan.ac.kr>

# Topics

---

- Micro-Operations
- Control of Processor
- Hardwired Implementation

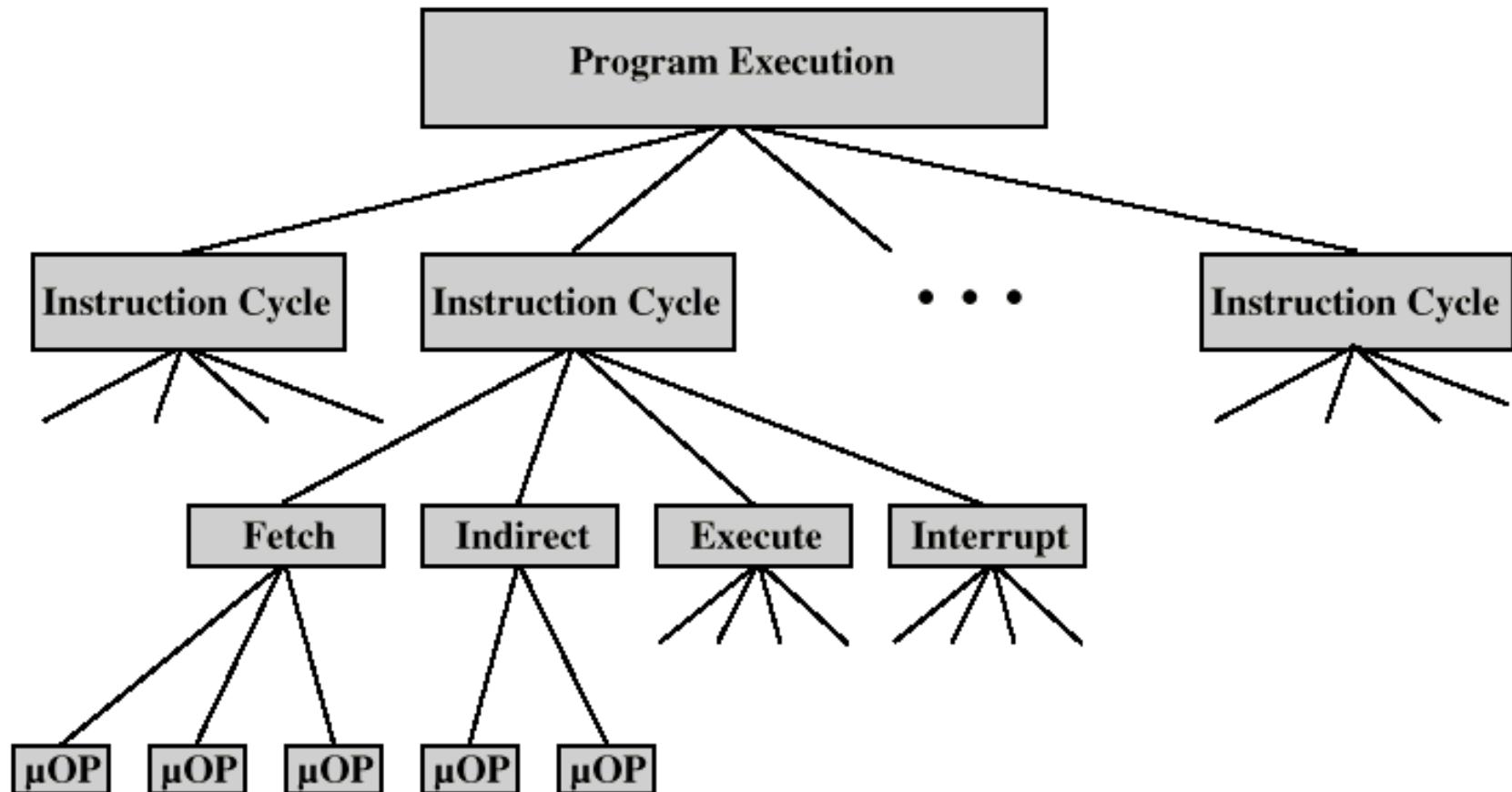
# Micro-Operations

---

- A computer executes a program
- Fetch/execute cycle
  - indirect, interrupt cycles, etc
- Each cycle has a number of steps
  - see pipelining
  - Called micro-operations
- Each step does very little
  - Atomic operation of CPU

# Constituent Elements of Program Execution

---



# Fetch - 4 Registers

---

- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched

# Fetch Sequence

---

- Address of next instruction is in PC
- Content of PC is copied to MAR
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches

# Fetch Sequence (symbolic)

---

t1: MAR <- (PC)

t2: MBR <- (memory)

    PC <- (PC) + I

t3: IR <- (MBR)

**or**

t1: MAR <- (PC)

t2: MBR <- (memory)

    PC <- (PC) + I

    IR <- (MBR)

- (R) means “content of R”
- We use () in the right side of a Micro-operation
- tx = time unit/clock cycle

MAR	
MBR	
PC	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
IR	
AC	

(a) Beginning (before  $t_1$ )

MAR	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
MBR	
PC	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
IR	
AC	

(b) After first step

MAR	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
MBR	0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
PC	0 0 0 0 0 0 0 0 1 1 0 0 1 0 1
IR	
AC	

(c) After second step

MAR	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
MBR	0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
PC	0 0 0 0 0 0 0 0 1 1 0 0 1 0 1
IR	0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
AC	

(d) After third step

Figure 15.2 Sequence of Events, Fetch Cycle

# Rules for Clock Cycle Grouping

---

- Proper sequence must be followed
  - $\text{MAR} \leftarrow (\text{PC})$  must precede  $\text{MBR} \leftarrow \text{Memory}$
- Conflicts must be avoided
  - Must not read & write same register at same time
  - $\text{MBR} \leftarrow \text{Memory}$  &  $\text{IR} \leftarrow (\text{MBR})$  must not be in same cycle
- Also:  $\text{PC} \leftarrow (\text{PC}) + /$  involves addition
  - Use ALU: May need additional micro-operations
  - Use increment circuit exclusively for PC

# Indirect Cycle

---

- Assume one-address instruction format
- t1:  $\text{MAR} \leftarrow (\text{IR}(\text{Address}))$  // address field of IR  
t2:  $\text{MBR} \leftarrow \text{Memory}$   
t3:  $\text{IR}(\text{Address}) \leftarrow (\text{MBR}(\text{Address}))$
- MBR contains an address
- IR is now in same state as if direct addressing had been used

# Interrupt Cycle

- t1:  $MBR \leftarrow (PC)$
  - t2:  $MAR \leftarrow Save\_Address$
  - $PC \leftarrow Routine\_Address$
  - t3:  $Memory \leftarrow (MBR)$
  - This is a minimum
    - May be additional micro-ops to get addresses
    - N.B. saving context is done by interrupt handler routine, not micro-ops in interrupt cycle
- Interrupt가 발생하여, 현재의 실행주소를 save\_address 영역에 저장함

# Execute Cycle (ADD)

---

- Different for each instruction
  - N different opcodes  $\Rightarrow$  N different sequences of  $\mu$ OPs
- e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1

t1: MAR  $\leftarrow$  (IR(Address))

t2: MBR  $\leftarrow$  Memory

t3: R1  $\leftarrow$  (R1) + (MBR)

- Note that no overlap of micro-operations

# Execute Cycle (ISZ)

---

- ISZ X - increment and skip if zero

t1: MAR <- (IR<sub>address</sub>)

t2: MBR <- (memory)

t3: MBR <- (MBR) + 1

t4: memory <- (MBR)

    if (MBR) == 0 then PC <- (PC) + 1

- Notes:

— if is a single micro-operation

— Micro-operations done during t4

Example:

L1:     INC R1

ISZ X

JMP L1

ADD R1,R2

# Execute Cycle (BSA)

- BSA X - Branch and save address

- Address of instruction following BSA is saved in X
  - X is a memory location
  - Execution continues from X + I

t1: MAR  $\leftarrow$  (IR(Address)) // Addr. of X to MAR  
MBR  $\leftarrow$  (PC) // Addr. of instr. To MBR  
t2: PC  $\leftarrow$  (IR(Address)) // PC  $\leftarrow$  X  
Memory  $\leftarrow$  (MBR) // Save addr. of instr. in X  
t3: PC  $\leftarrow$  (PC) + I // PC  $\leftarrow$  X + I

*Example of BSA*

PC = 10	0	BSA 135
PC = 21	next instruction	
135	21(return address)	
PC = 136	Subroutine	
1	BUN 135	

서브 프로그램으로 분기/실행 후, 다시 복귀함  
- BSA 135 // 135번지에 BSA의 다음 명령어 번지(21)  
를 저장함. 그 후, 135+1을 실행함.. 마치면 BUN 135에 의해  
135번지로 unconditional branch (BUN 135)가 실행되어  
21번지로 되돌아옴

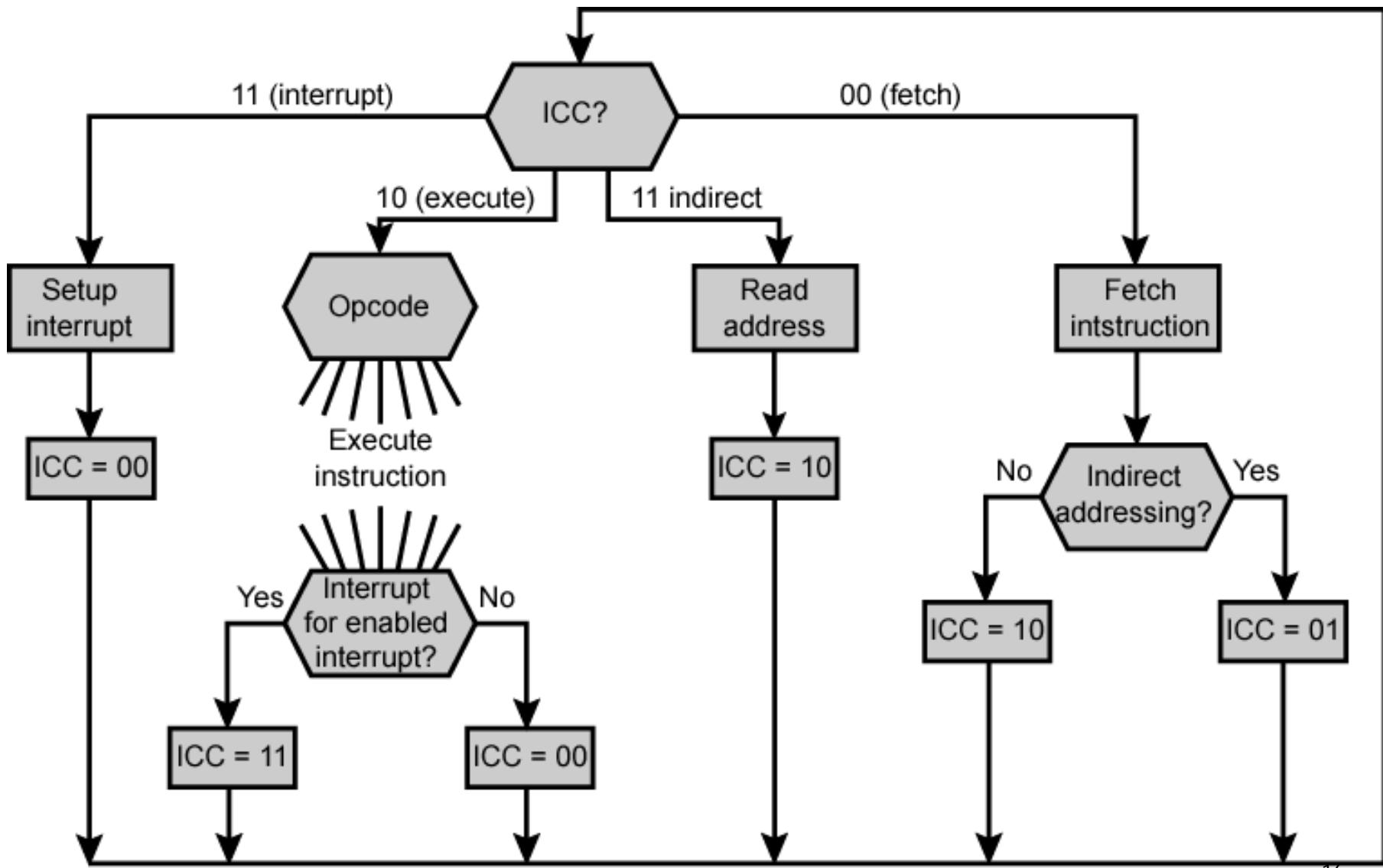
[www.qazviniau.ac.ir/Services/elearning/farokhi](http://www.qazviniau.ac.ir/Services/elearning/farokhi)

# Instruction Cycle

---

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

# Flowchart for Instruction Cycle



# Control of Processor

---

- Decomposition CPU functioning into  $\mu$ Ops:  
Defines functional requirements for the control unit
  - Functional requirements: those functions that the control unit must perform
- Definition of functional requirements:  
Basis for design and implementation of CPU

# Functional Requirements

---

- Characterization of control unit: 3 steps
  1. Define basic elements of processor
  2. Describe micro-operations processor performs
  3. Determine functions that the control unit must perform

# Basic Elements of Processor

---

- ALU
- Registers
- Internal data paths
- External data paths
- Control Unit

# Types of Micro-operation

---

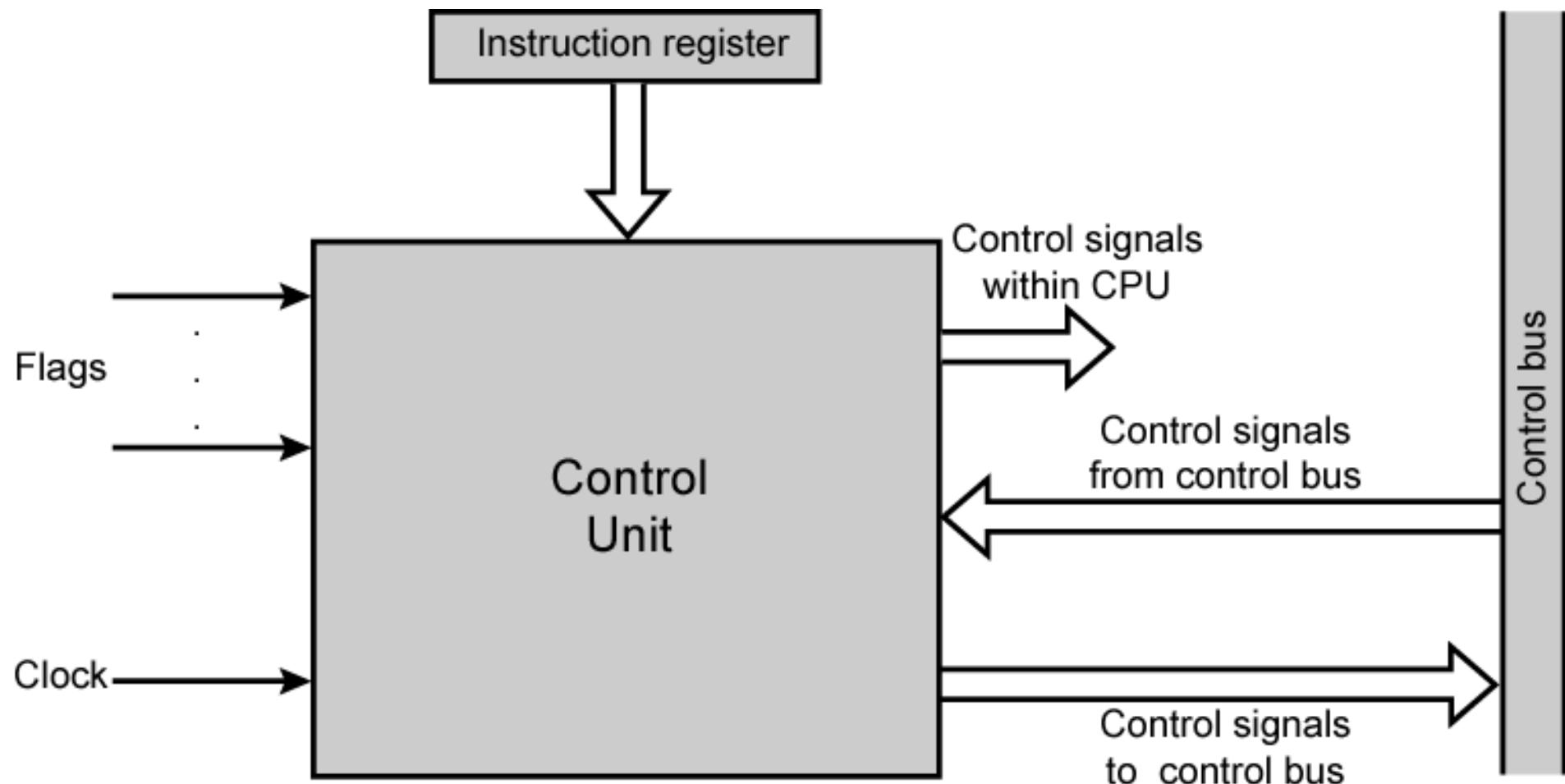
- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical ops

# Functions of Control Unit

---

- Two basic tasks:
  1. **Sequencing**
    - The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.
  2. **Execution**
    - The control unit causes each micro-operation to be performed.
- This is done using *Control Signals*

# Model of Control Unit



# Control Signals - Inputs (1)

---

- Clock
  - One micro-instruction (or set of parallel micro-operations) per clock cycle
- Instruction register
  - Op-code for current instruction
  - Determines which micro-operations are performed

## Control Signals - Inputs (2)

---

- Flags
  - State of CPU
  - Results of previous operations
- From control bus
  - Interrupts
  - Acknowledgements

# Control Signals - Outputs

---

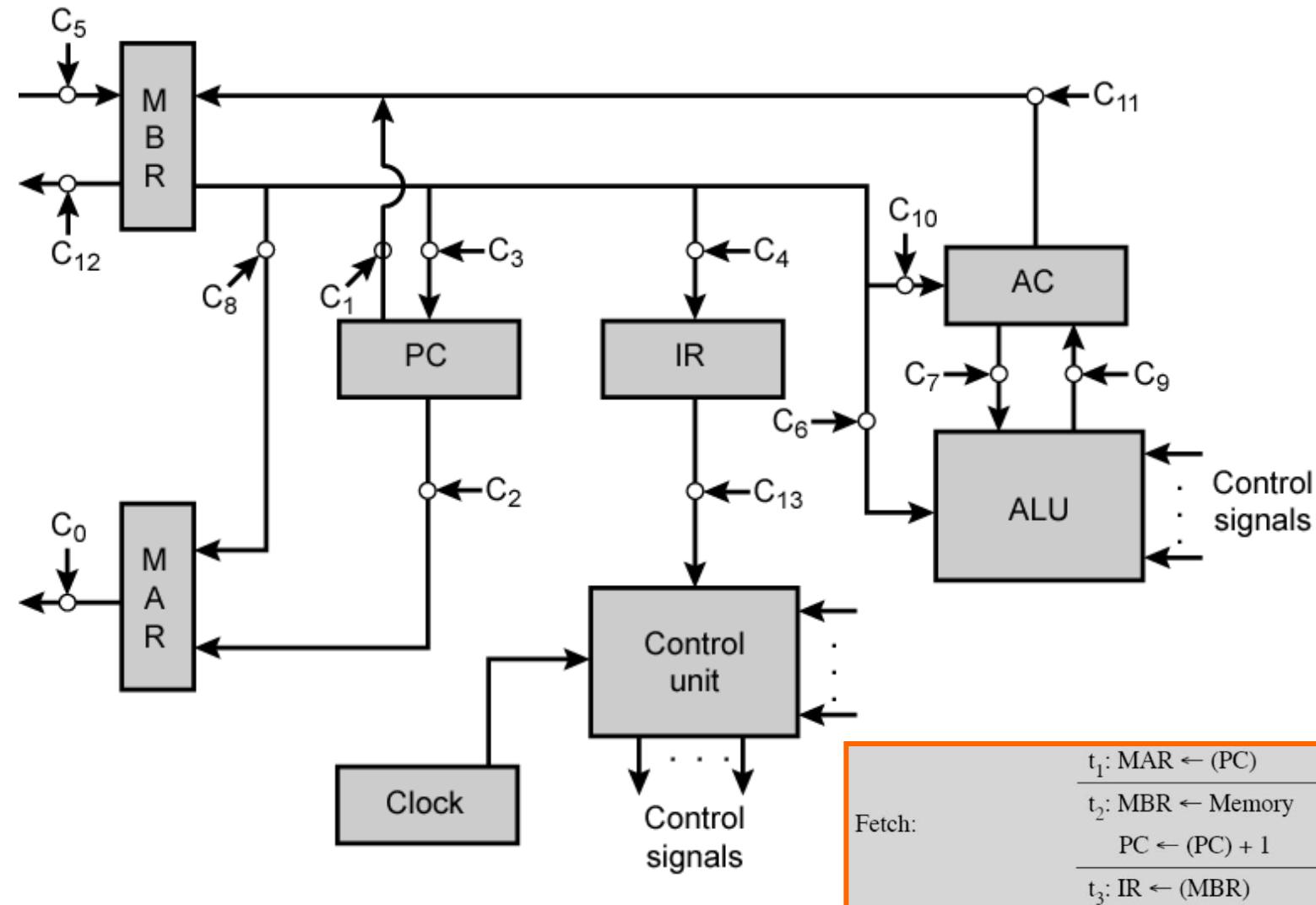
- Within CPU
  - Cause data movement
  - Activate specific functions
- Via control bus
  - To memory
  - To I/O modules

# Example Control Signal Sequence - Fetch

---

- $\text{MAR} \leftarrow (\text{PC})$ 
  - Control unit activates signal to open gates between PC and MAR
- $\text{MBR} \leftarrow (\text{memory})$ 
  - Open gates between MAR and address bus
  - Memory read control signal
  - Open gates between data bus and MBR

# Data Paths and Control Signals



Micro-operations	Timing	Active Control Signals
Fetch:	$t_1: MAR \leftarrow (PC)$	$C_2$
	$t_2: MBR \leftarrow \text{Memory}$	$C_5, C_R$
	$PC \leftarrow (PC) + 1$	
Indirect:	$t_3: IR \leftarrow (MBR)$	$C_4$
	$t_1: MAR \leftarrow (IR(\text{Address}))$	$C_8$
	$t_2: MBR \leftarrow \text{Memory}$	$C_5, C_R$
Interrupt:	$t_3: IR(\text{Address}) \leftarrow (MBR(\text{Address}))$	$C_4$
	$t_1: MBR \leftarrow (PC)$	$C_1$
	$t_2: MAR \leftarrow \text{Save-address}$ $PC \leftarrow \text{Routine-address}$	
	$t_3: \text{Memory} \leftarrow (MBR)$	$C_{12}, C_W$

$C_R$  = Read control signal to system bus.  
 $C_W$  = Write control signal to system bus.

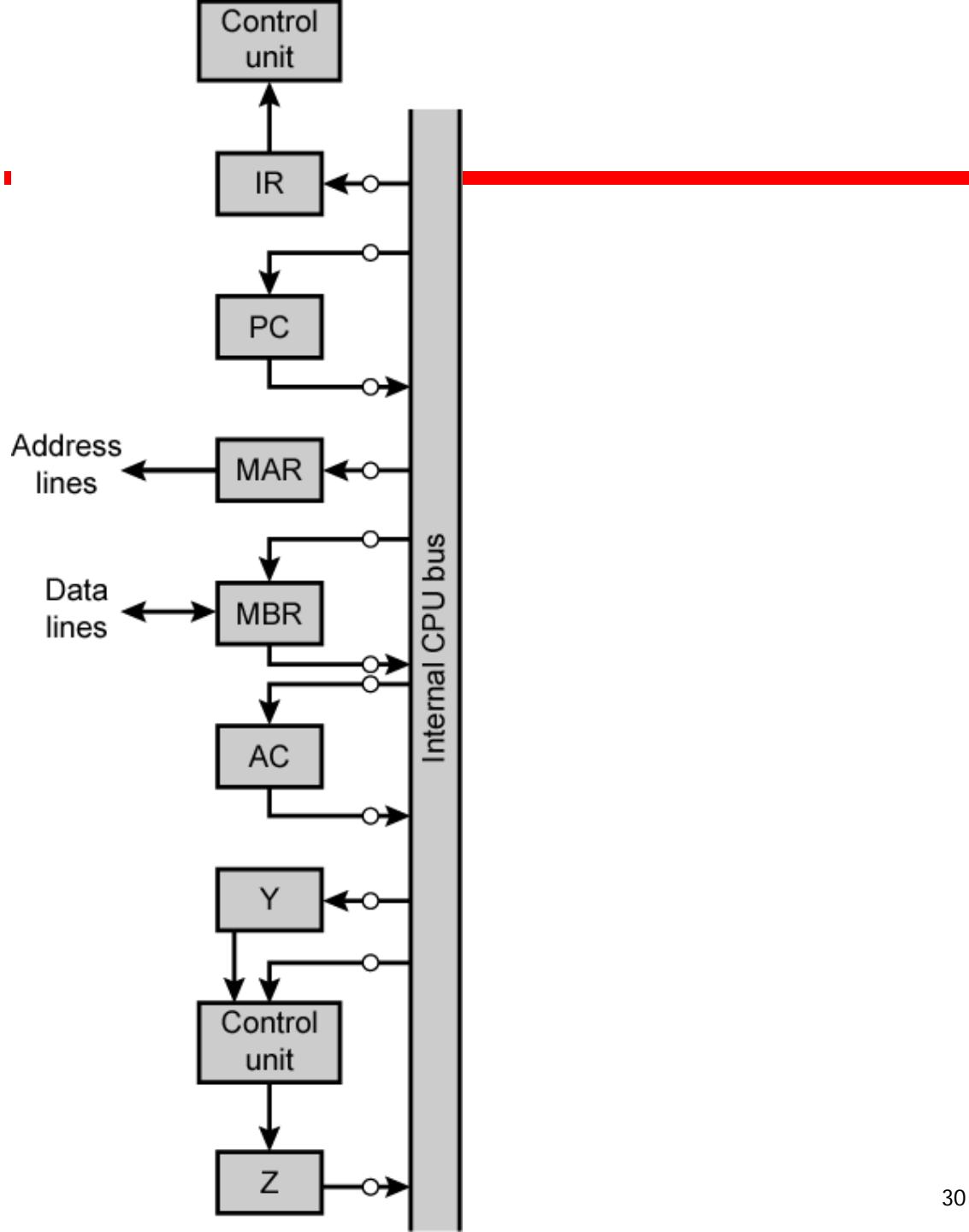
# Internal Processor Organization

---

- Usually a single internal bus
- Gates control movement of data onto and off the bus
- Control signals control data transfer to and from external systems bus
- Temporary registers needed for proper operation of ALU

# CPU with Internal Bus

- Y: temporary storage of an input
- Z: temporary storage of output
- OP:  $Z \leftarrow (AC) \text{ OP } (Y)$



# Control Unit Implementation

---

- Hardwired
- Microprogrammed

# Hardwired Implementation (1)

---

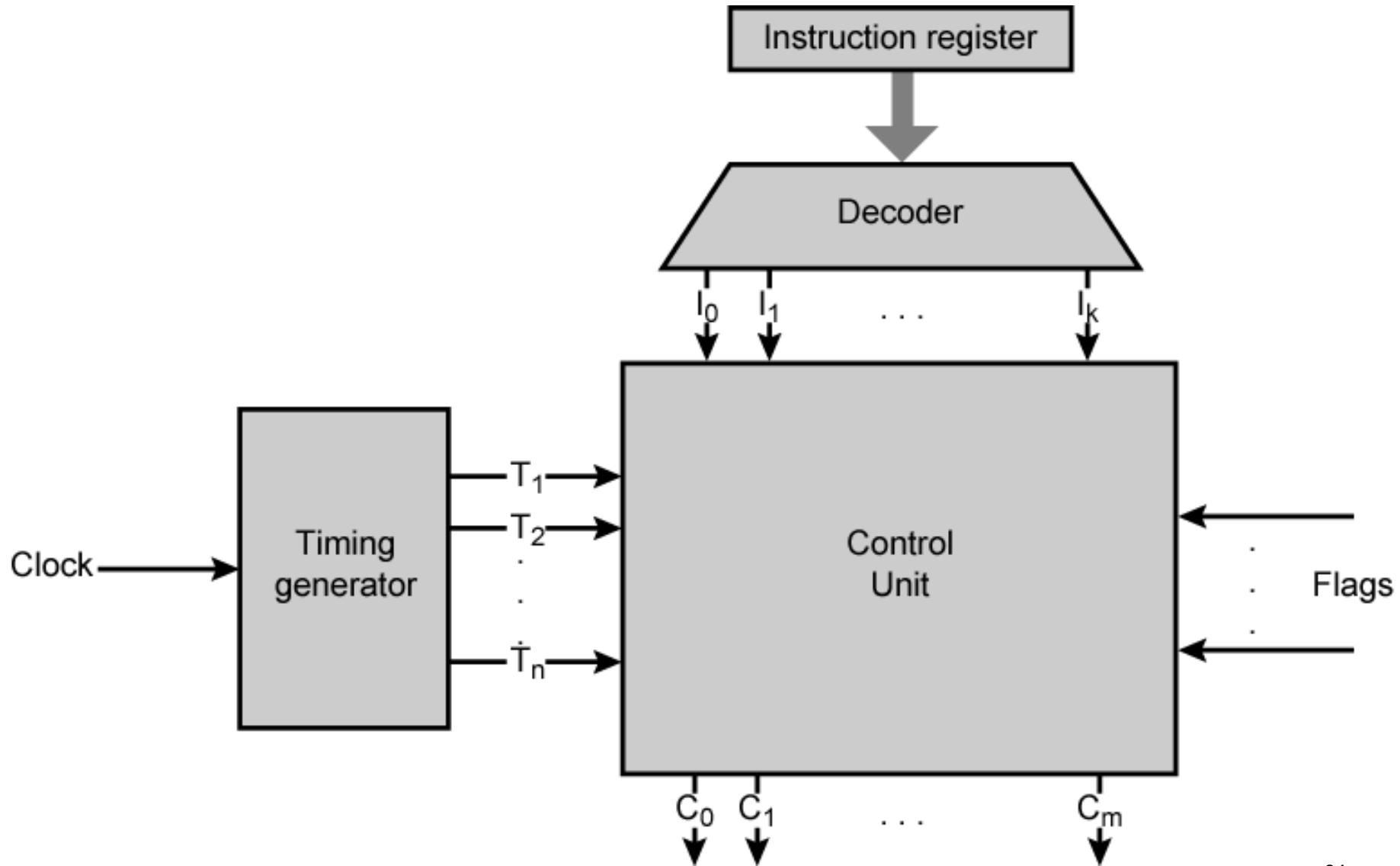
- Control unit inputs
- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - $n$  binary inputs and  $2^n$  outputs

# Hardwired Implementation (2)

---

- Clock
  - Repetitive sequence of pulses
  - Useful for measuring duration of micro-ops
  - Must be long enough to allow signal propagation
  - Different control signals at different times within instruction cycle
  - Need a counter with different control signals for  $t_1$ ,  $t_2$  etc. ( $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1 \rightarrow \dots$ )

# Control Unit with Decoded Inputs



# Control Signal Example

- Assume
    - $C_5$  : signal for MBR  $\leftarrow$  Memory
    - P, Q: control signals for the four cycles
      - 00: fetch cycle
      - 01: indirect cycle
      - 10: execute cycle
      - 11: interrupt cycle
    - LDA, ADD, AND: the only instructions read from memory
  - $C_5 = \overline{P} \bullet \overline{Q} \bullet T_2 + \overline{P} \bullet Q \bullet T_2 + P \bullet \overline{Q} \bullet (LDA + ADD + AND) \bullet T_2$ 
    - (fetch)
    - (indirect)
    - (execute)

# Problems With Hard Wired Designs

---

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions



*Thank you !*