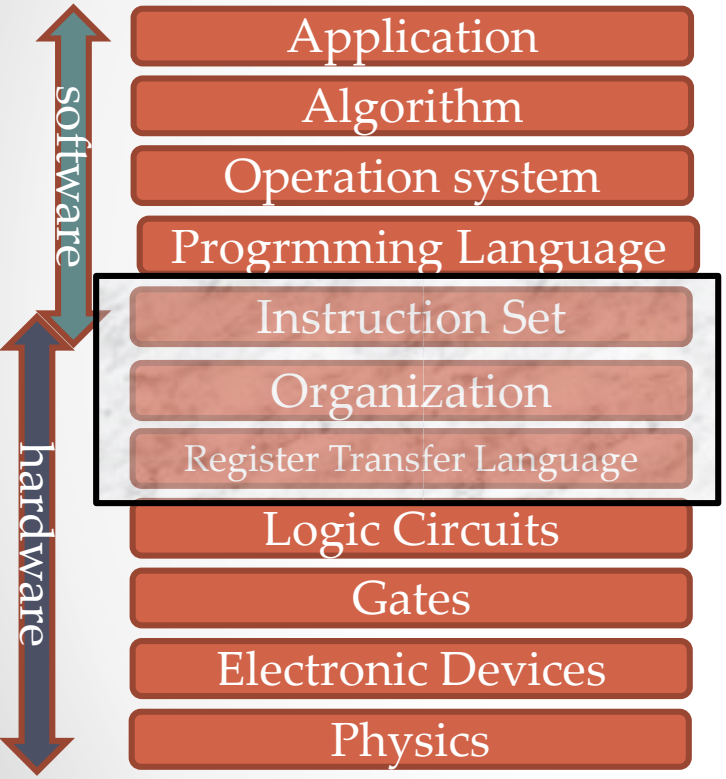
# Computer Architecture Lec 5a

Dr. Esti Stein
(Partly taken from Dr. Alon Schclar slides)

Based on slides by:
**Prof. Myung-Eui Lee**
Korea University of Technology & Education
Department of Information & Communication

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

# General Purpose Digital Computer




- Capable of executing various microoperations.
- Can be instructed as to what specific sequence of operations to perform.

# A Program

- The user of a computer can control the process by means of a ***program.***
- A program is a set of ***instructions*** that specify the operations, operand, and the sequence *(control)*
- A instruction is a binary code that specifies a sequence of microoperations

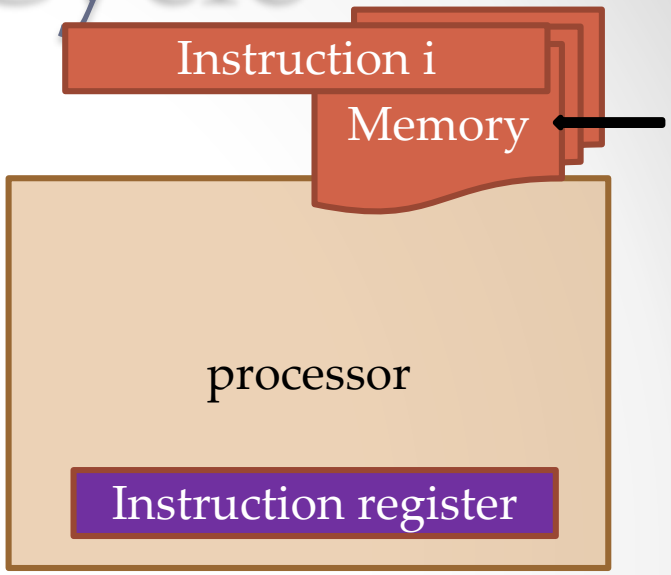
- Instruction codes together with data are stored in memory *(=Stored Program Concept)*
- The computer reads each instruction from memory and ***places it in*** IR ***control register***. The control then ***interprets the binary code*** of the instruction and proceeds to ***execute it*** by issuing a sequence of microoperations.

# Execution Cycle

*Instruction Fetch*

Program counter(addr)

Obtain an instruction from program storage in memory



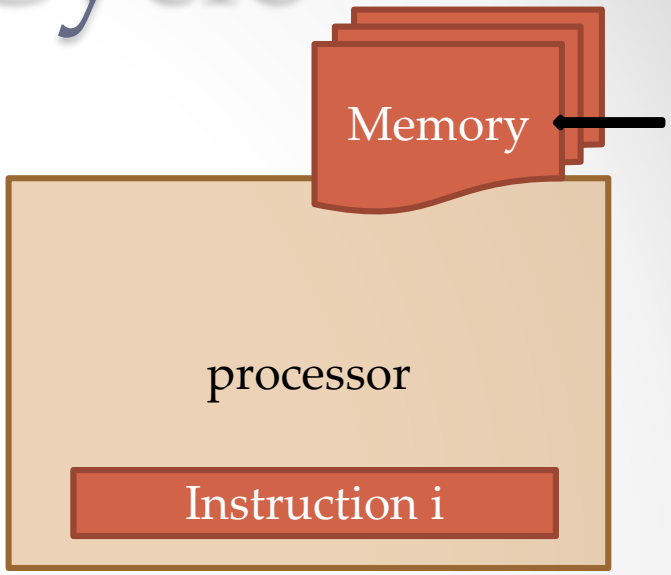
# Execution Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory

*Instruction Decode*

Determine required actions and instruction size



# Execution Cycle

*Instruction Fetch*

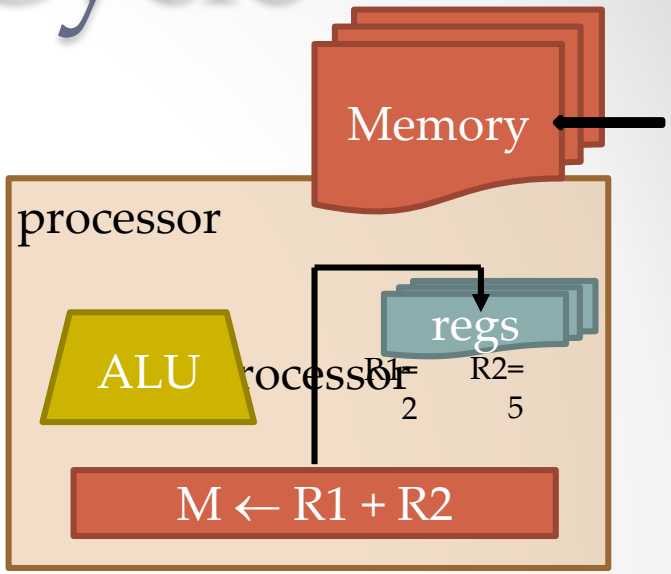
Obtain instruction from program storage in memory

*Instruction Decode*

Determine required actions and instruction size

*Operand Fetch*

Locate and obtain operand data



# Execution Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory

*Instruction Decode*

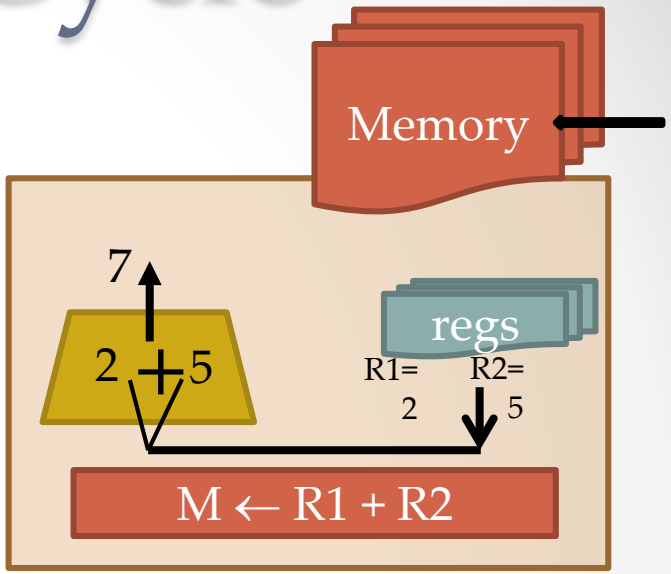
Determine required actions and instruction size

*Operand Fetch*

Locate and obtain operand data

*Execute*

Compute result value or status



# Execution Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory

*Instruction Decode*

Determine required actions and instruction size

*Operand Fetch*

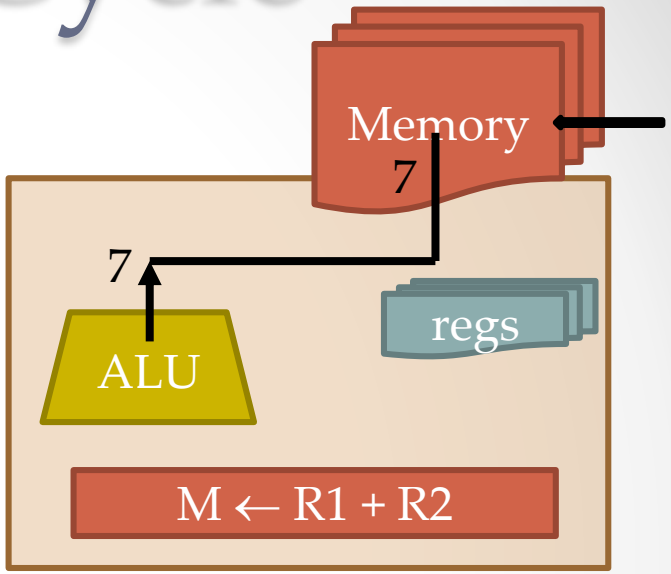
Locate and obtain operand data

*Execute*

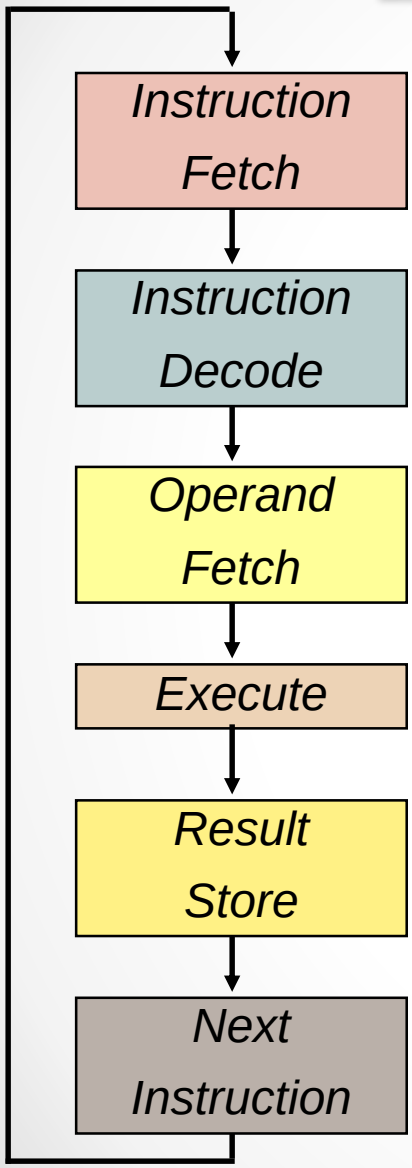
Compute result value or status

*Result Store*

Deposit results in storage



# Execution Cycle




Obtain instruction from program storage in memory

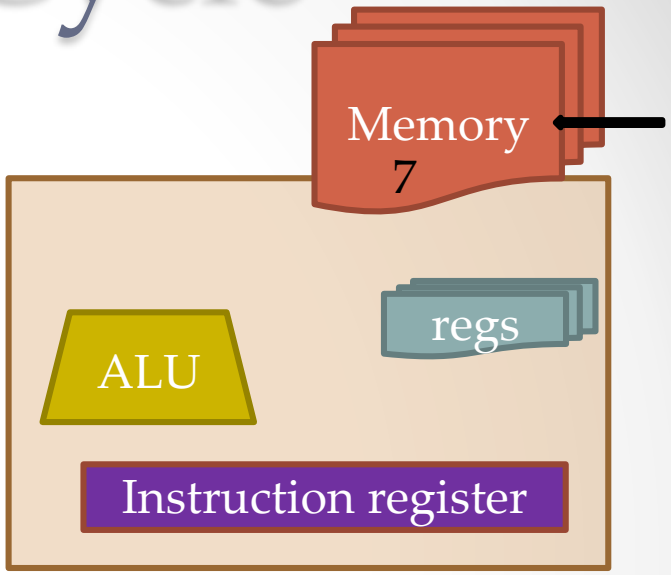
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage

Determine next instruction (not the next in case of **branch**)



# An Instruction

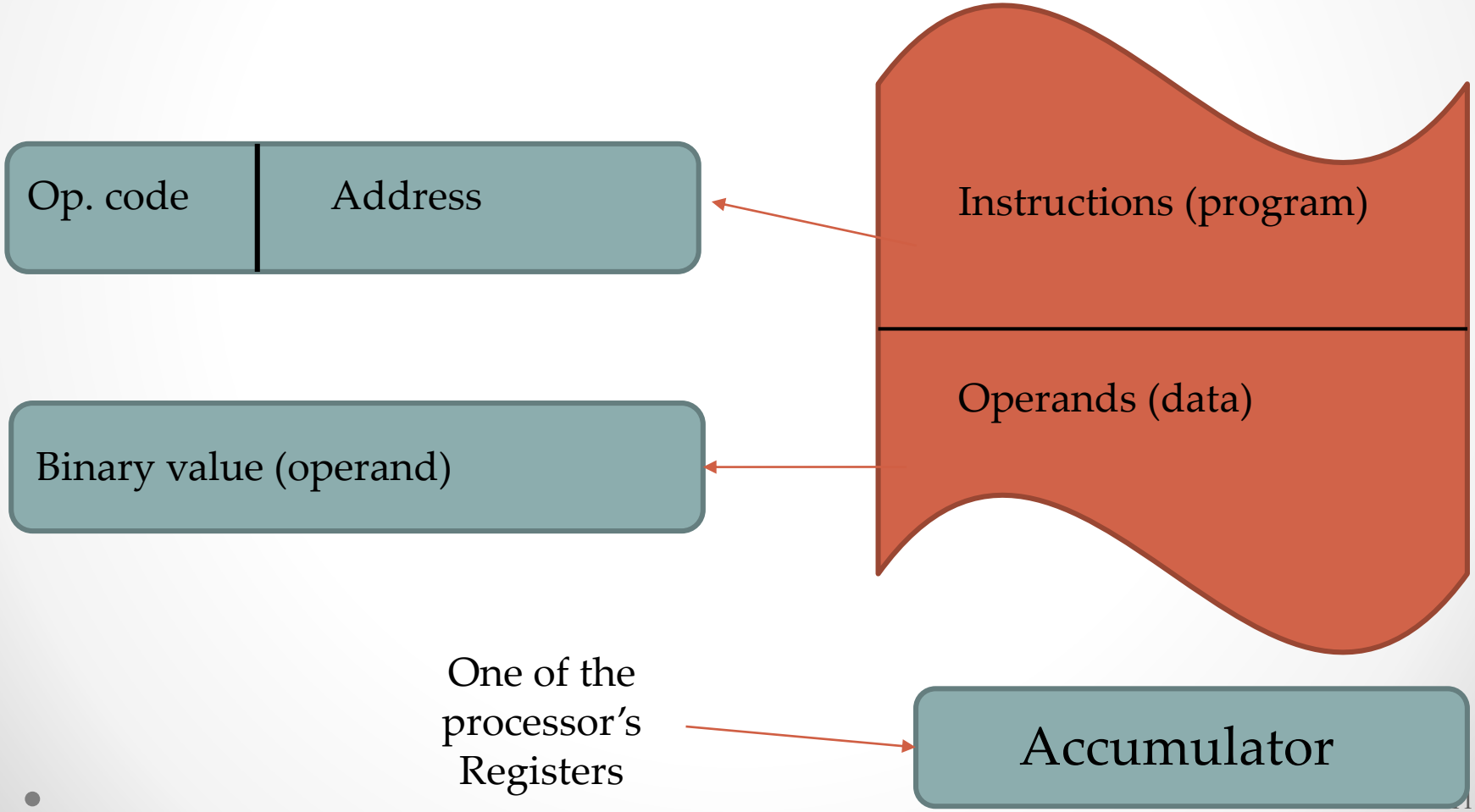
- A group of bits that instructs the computer to perform a specific operation.
- An instruction is usually divided into parts.


A group of bits that defines an operation:
add, subtract, shift, and etc.

An address in the memory

n bits define $2^n$ operations

# A Stored Program Organization



# The Instruction

All instructions and operands are 16 bits


4 bits ⇒ 16 possible operation codes

12 bits ⇒ *length* of memory is $2^{12} = 4096$

**Size of memory is 4096 × 16**

If an operation in the instruction does not need a memory operand, the rest of the bits can be used to expand the operation.
<u>Examples</u>:
clear accumulator, complement accumulator, read a character from the keyboard, etc.

# Addressing Modes

| Op. code | Value or opcode extension |
|---|---|

**Immediate addressing mode**

| 0 | Op. code | address in memory |
|---|---|---|

**Direct addressing mode**

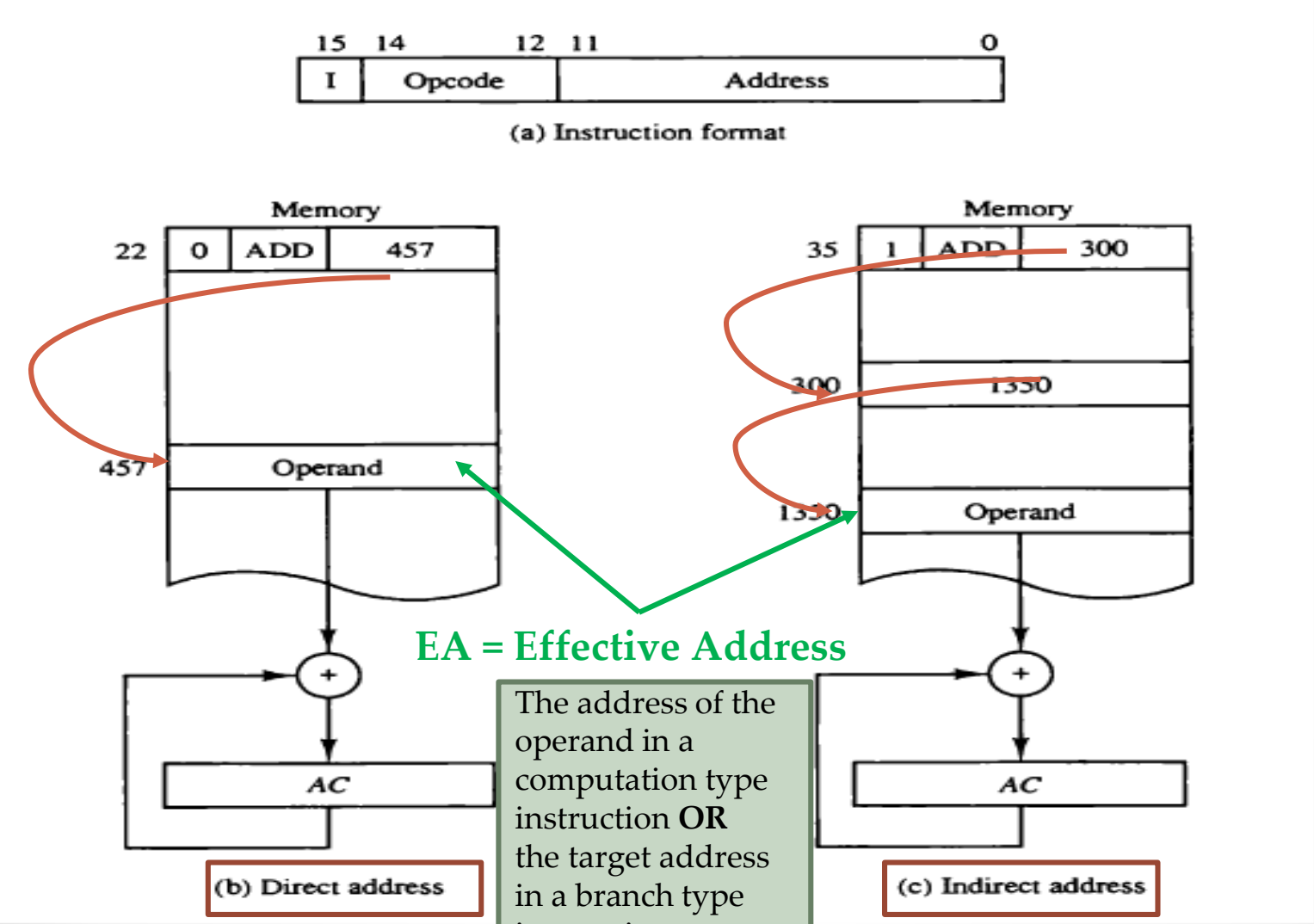
Indirect bit

3 bits remains ⇒ 8 possible operations

| 1 | Op. code | address of address in memory (pointer) |
|---|---|---|

**Indirect addressing mode**

# Addressing Modes




**EA = Effective Address**

The address of the operand in a computation type instruction **OR** the target address in a branch type instruction

# Computer Registers

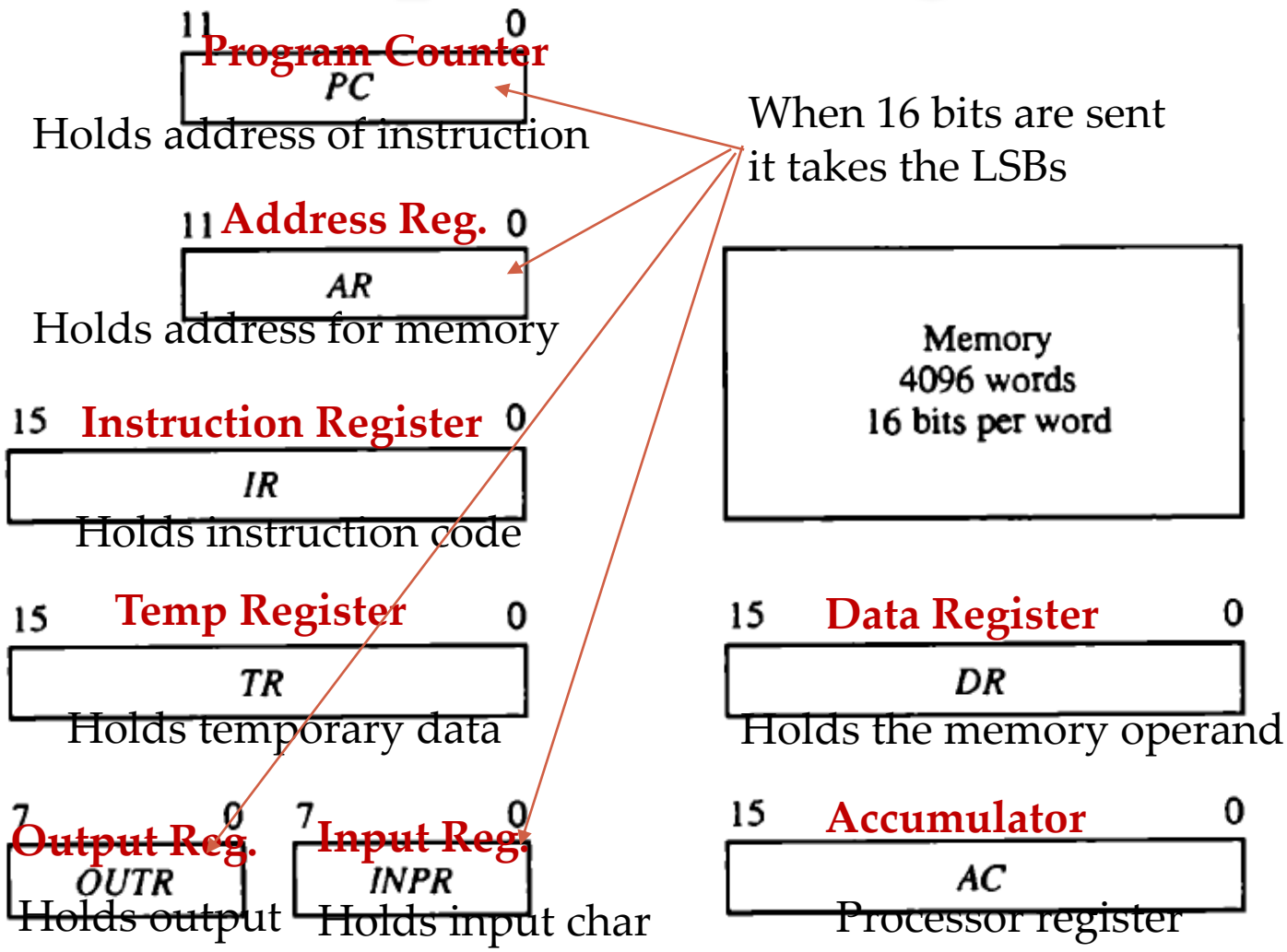



**Figure 5-3** Basic computer registers and memory.

# Register with INC, LD, CLR

AC
AR
DR
PC
TR

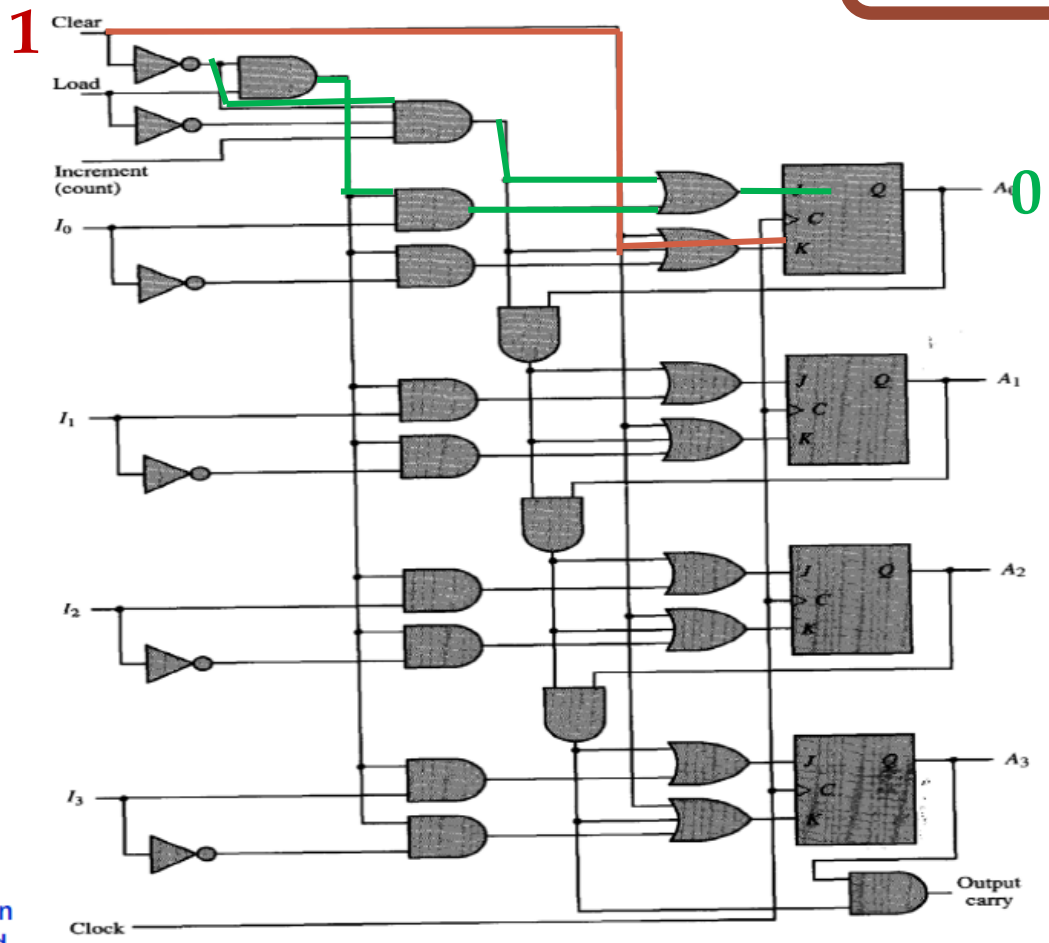



Figure 2-11 4-bit binary counter with parallel load and synchronous clear.

Taken from: M. Mano/Computer Design and Architecture 3rd Ed.

# Register with INC, LD, CLR

AC
AR
DR
PC
TR

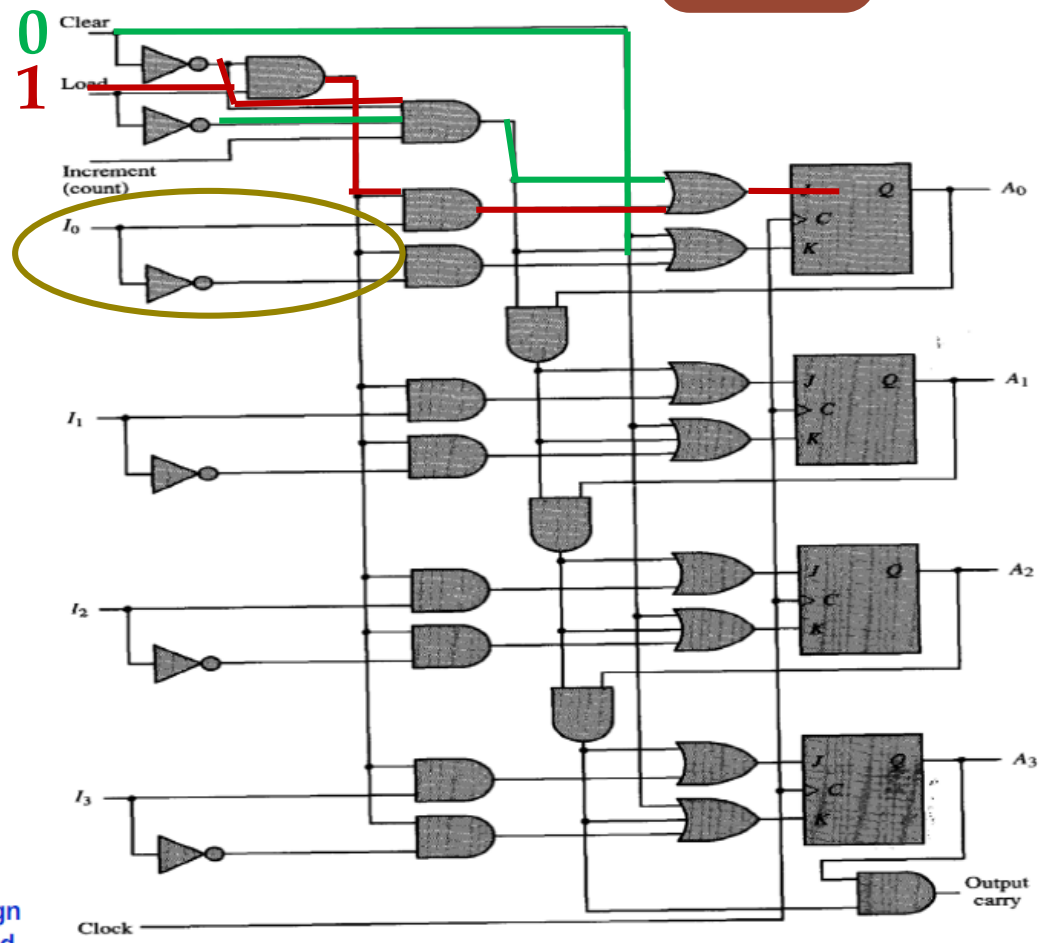



Figure 2-11 4-bit binary counter with parallel load and synchronous clear.

Taken from: M. Mano/Computer Design and Architecture 3rd Ed.

# Register with INC, LD, CLR

AC
AR
DR
PC
TR

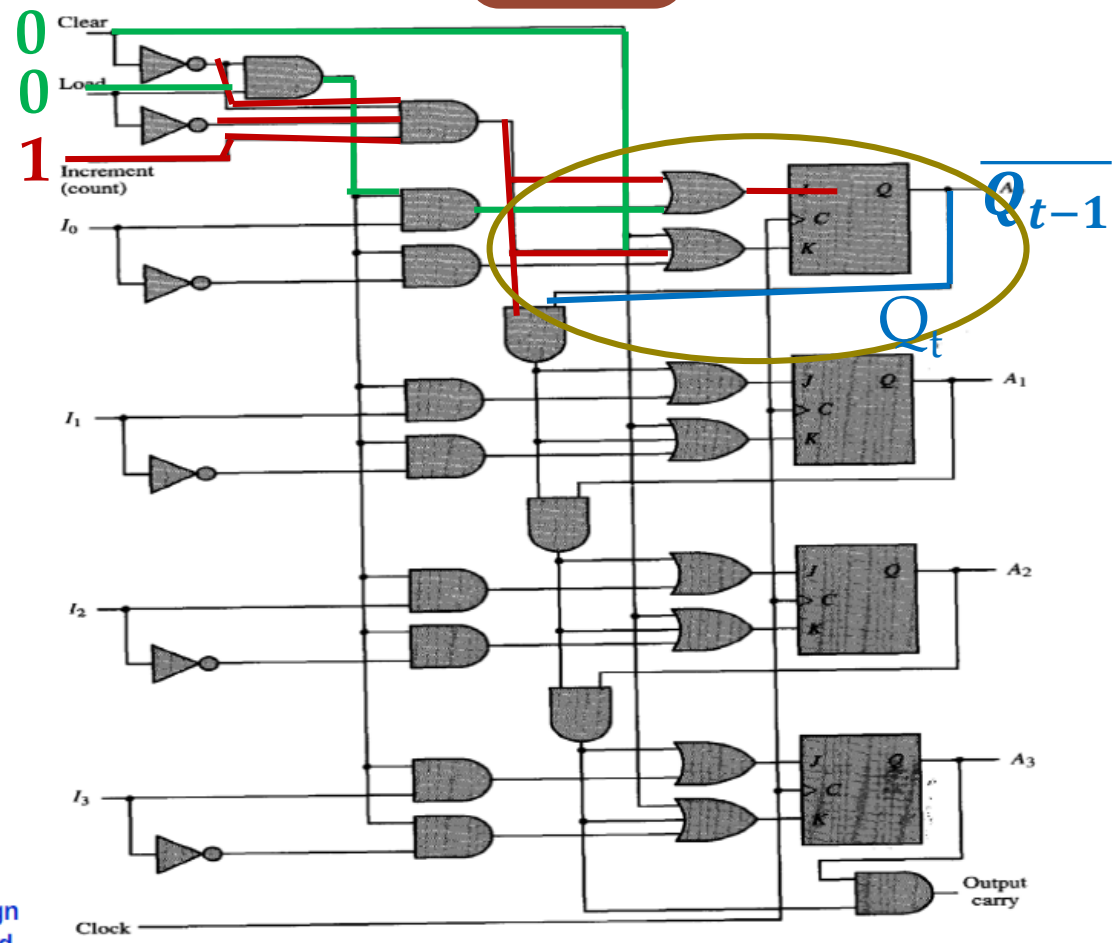



Figure 2-11 4-bit binary counter with parallel load and synchronous clear.

Taken from: M. Mano/Computer Design and Architecture 3rd Ed.

# The Bus

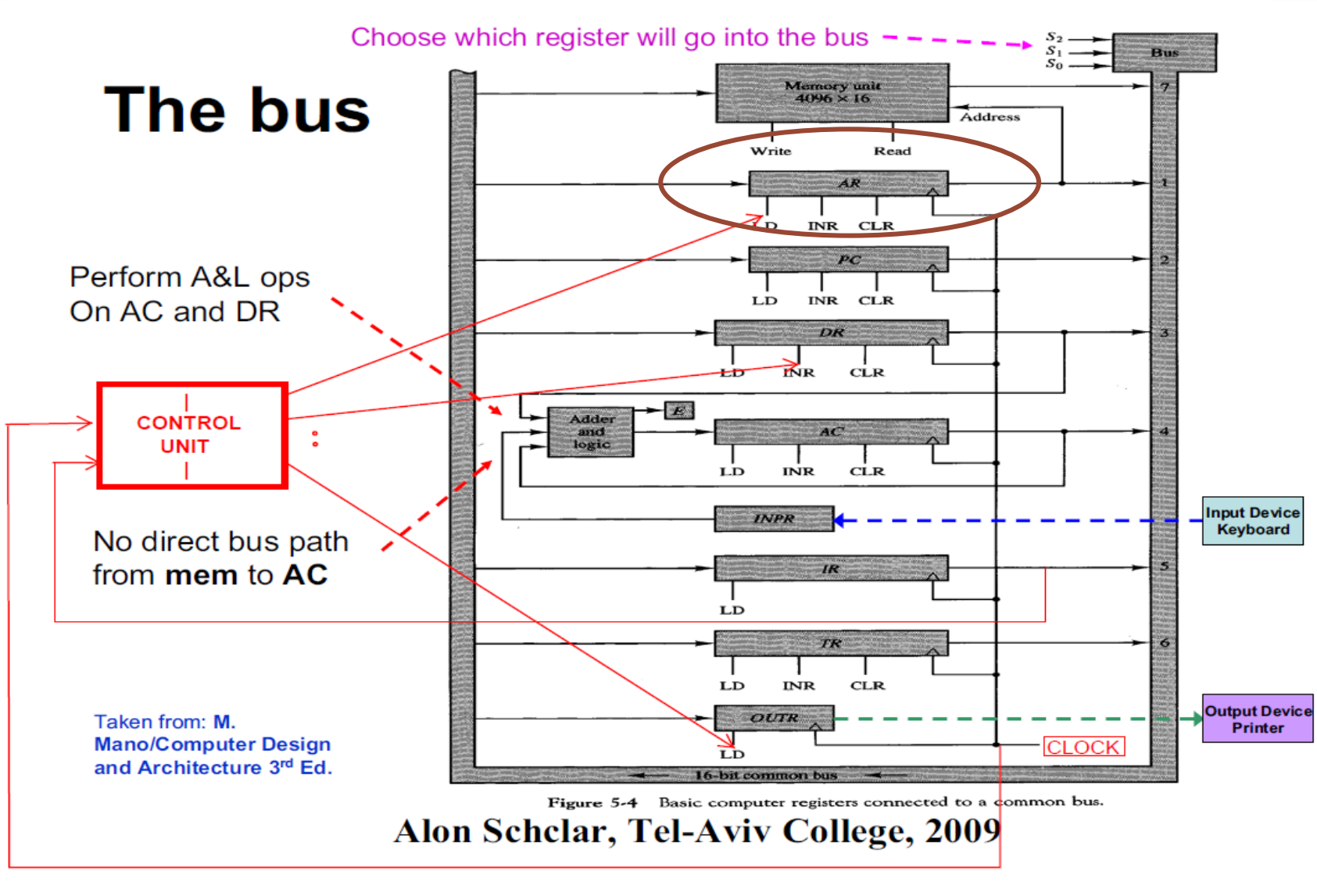



Figure 5-4 Basic computer registers connected to a common bus.

# Reminder – register with parallel load

IR
OUTR

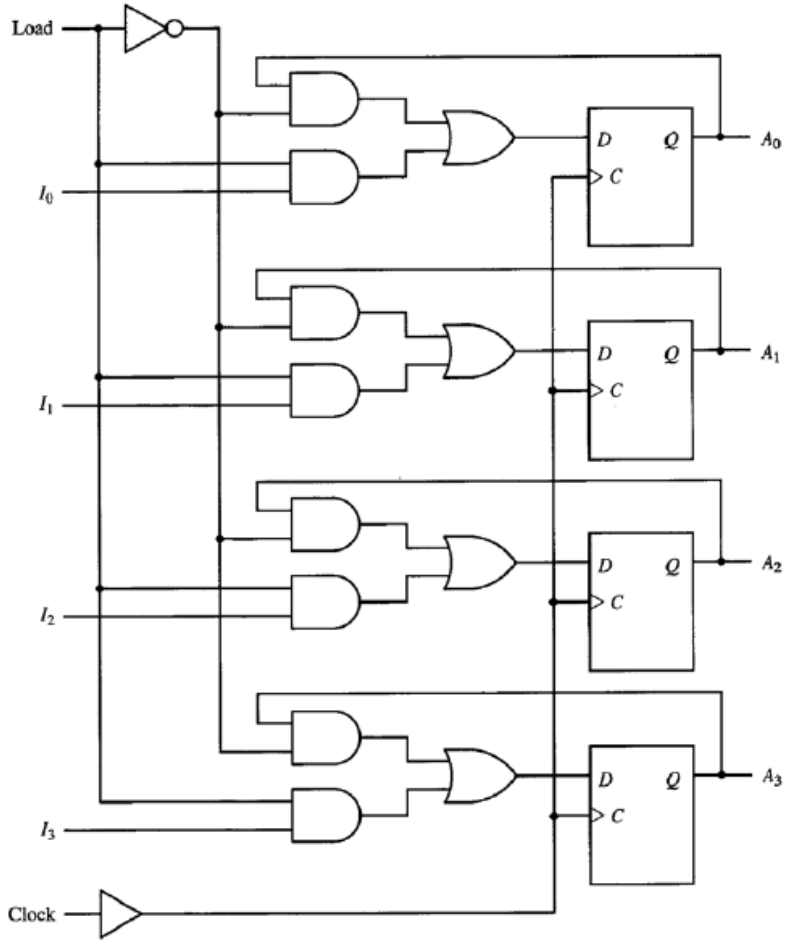
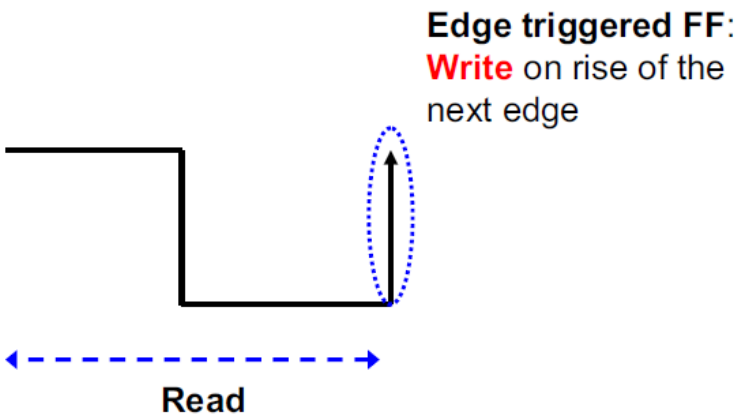



Figure 2-7 4-bit register with parallel load.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

**Alon Schclar, Tel-Aviv College, 2009**

# The clock cycle



# The Bus

» Accumulator(***AC***) : 3 Path

- 1) Register Microoperation : clear AC, shfift AC,…
- 2) Data Register : **add DR to AC, and DR to AC**
  *End carry bit set/reset),* memory **READ**

$$D_2T_4 : DR \leftarrow M[AR]$$
$$D_2T_5 : AC \leftarrow DR, SC \leftarrow 0$$

- 3) INPR : Device
  *Adder & Logic*

» ***Note***) Two microoperations can be executed at the same time

$$DR \leftarrow AC : s_2s_1s_0 = 100(4), DR(load)$$
$$AC \leftarrow DR : DR \rightarrow Adder\ \&\ Logic \rightarrow AC(load)$$

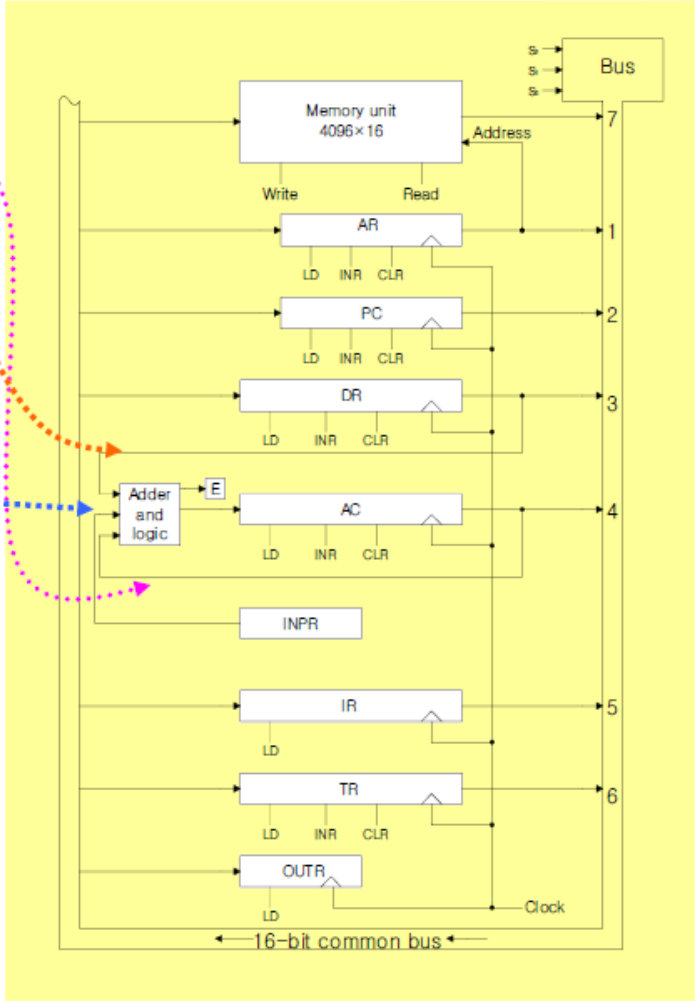
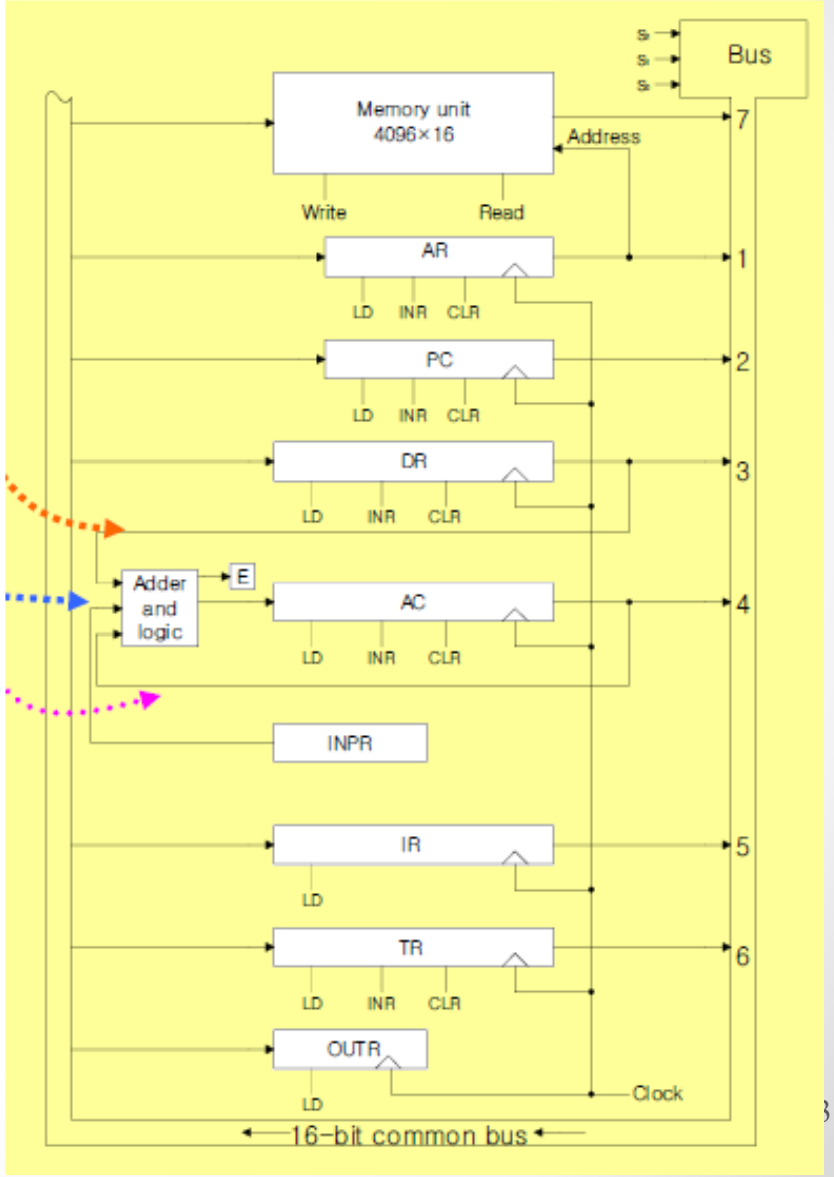



*Fig. 5-4 Basic computer registers connected to a common bus*

# The Bus - examples

- **TR ←DR**
  - Place DR on BUS
    - $S_2S_1S_0$=011 (DR num is 3)
  - Insert BUS content to TR
    - Enable LD (load) input of TR

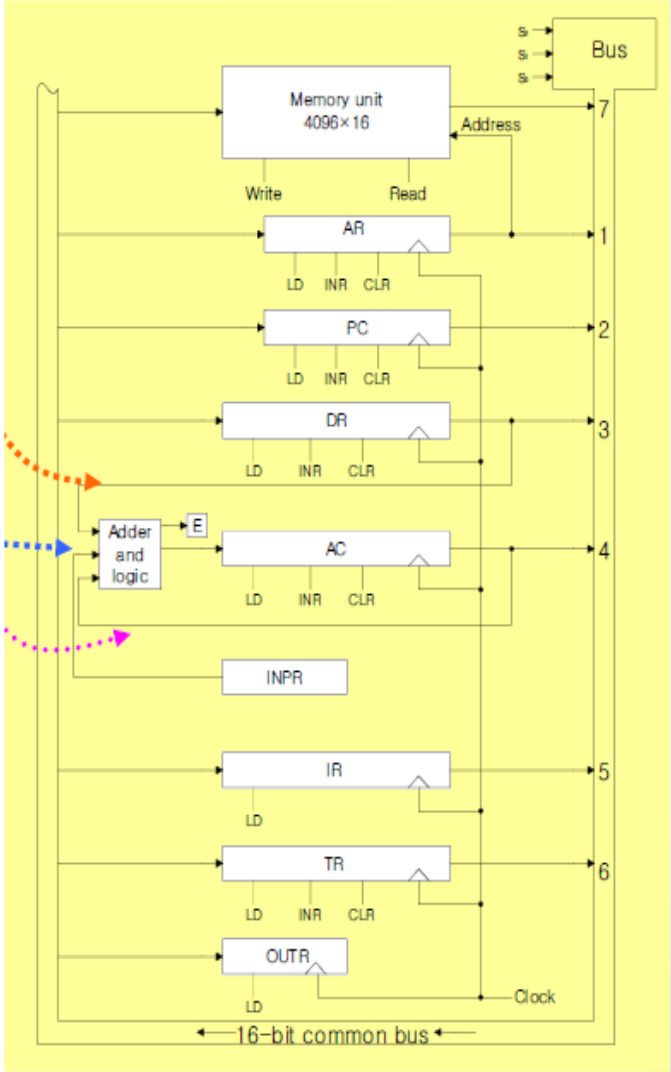
- **DR ←MEM[AR]**
  - AR is connected to address input of MEM
  - Enable READ input of MEMORY
  - Place MEMORY content (MEM[AR]) on BUS
    - $S_2S_1S_0$=111 (MEM num is 7
  - Insert BUS content to DR
    - Enable LD (load) input of DR



# The Bus – concurrent data transfer

- During the same clock cycle
  - The content of any register can be applied onto the bus and
  - an operation can be performed in the adder and logic circuit

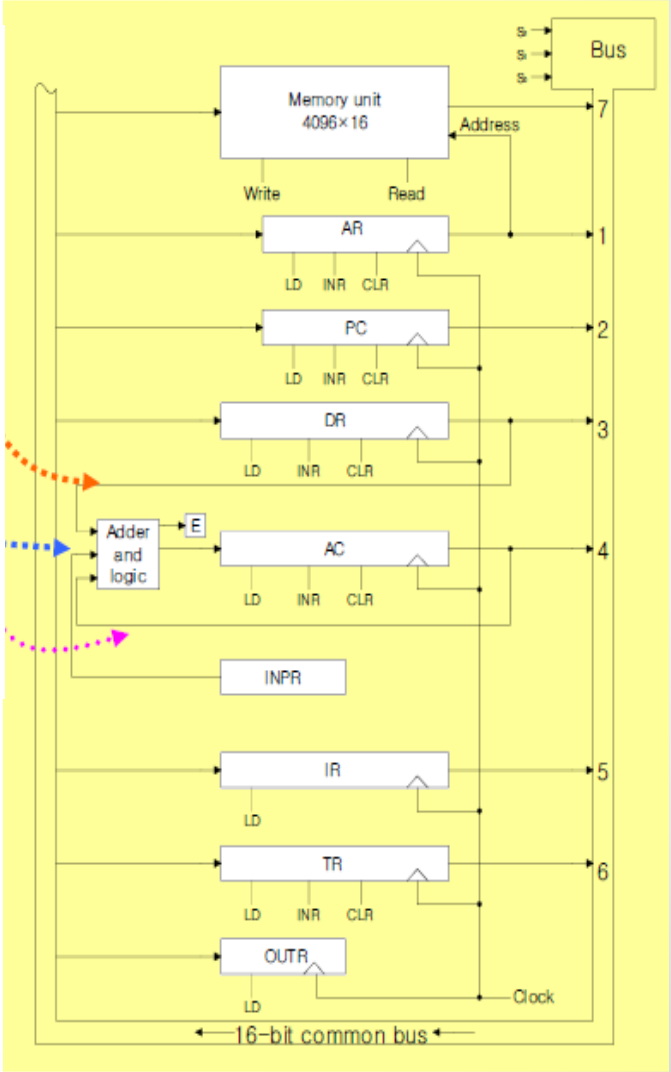
- The clock transition at the end of the cycle
  - transfers the content of the bus into the target register and
  - the output of the adder and logic circuit into AC.



# The Bus – concurrent data transfer

- **DR ←AC**
  - Place AC on BUS
    - $S_2S_1S_0$=100 (AC num is 4)
  - Insert BUS content to DR
    - Enable LD (load) input of DR
- **AC ←DR**
  - DR connected to AC via the **Adder & Logic** (aka A&L or ALU)
  - Instruct ALU to let DR pass through
  - Enable LD (load) input of AC

- **Can be executed in the same clock cycle**



# Computer Instruction

- 5-3 Computer Instruction
  - 3 Instruction Code Formats : ***Fig. 5-5***
    - Memory-reference instruction
      - » Opcode = 000 ~ 110
        - I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect

| 15 | 14 – 12 | 11 – 0 |
|---|---|---|
| I | Opcode | Address |

- Register-reference instruction
  - » 7xxx (7800 ~ 7001) : CLA, CMA, ….

| 15 | 14 | 13 | 12 | 11 – 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | Register Operation |

- Input-Output instruction
  - » Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ….

| 15 | 14 | 13 | 12 | 11 – 0 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | I/O Operation |

| Symbol | Hex Code I = 0 | Hex Code I = 1 | Description |
|---|---|---|---|
| AND | 0xxx | 8xxx | And memory word to AC |
| ADD | 1xxx | 9xxx | Add memory word to AC |
| LDA | 2xxx | Axxx | Load memory word to AC |
| STA | 3xxx | Bxxx | Store content of AC in memory |
| BUN | 4xxx | Cxxx | Branch unconditionally |
| BSA | 5xxx | Dxxx | Branch and Save return address |
| ISZ | 6xxx | Exxx | Increment and skip if zero |
| CLA | 7800 | | Clear AC |
| CLE | 7400 | | Clear E |
| CMA | 7200 | | Complement AC |
| CME | 7100 | | Comp m e |
| CIR | 7080 | | Circulate right AC and E |
| CIL | 7040 | | Circulate left AC and E |
| INC | 7020 | | Increment AC |
| SPA | 7010 | | Skip next instruction if AC positive |
| SNA | 7008 | | Skip next instruction if AC negative |
| SZA | 7004 | | Skip next instruction if AC zero |
| SZE | 7002 | | Skip next instruction if E is 0 |
| HLT | 7001 | | Halt computer |
| INP | F800 | | Input character to AC |
| OUT | F400 | | Output character from AC |
| SKI | F200 | | Skip on input flag |
| SKO | F100 | | Skip on output flag |
| ION | F080 | | Interrup |
| IOF | F040 | | Inter |

# Instruction Set Completeness

A computer should have a set of instructions so that the user can evaluate any function that is known to be computable.

A complete set must include sufficient number of instructions from the following categories:

1. Arithmetic, logical, and shift - CMA, INC, CIR, AND..
2. Moving information between the registers, and between the registers and memory – LDA, STA
3. Program control, and status check – BUN, ISZ, BSA..
4. Input and output – INP, OUT..

A complete set of instructions, but not efficient.

# Timing and Control

- Clock pulses
  - A master clock generator controls the timing for all registers in the basic computer
  - The clock pulses are applied to all F/Fs and registers in system
  - The clock pulses do not change the state of a register unless the register is enabled by a control signal
  - The control signals are generated in the control unit
    - » The control signals provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator

# Two (major) Types of Control Organization

## Hardwired Control (This chapter)

- The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
- + Fast operation, - Wiring change(if the design has to be modified)

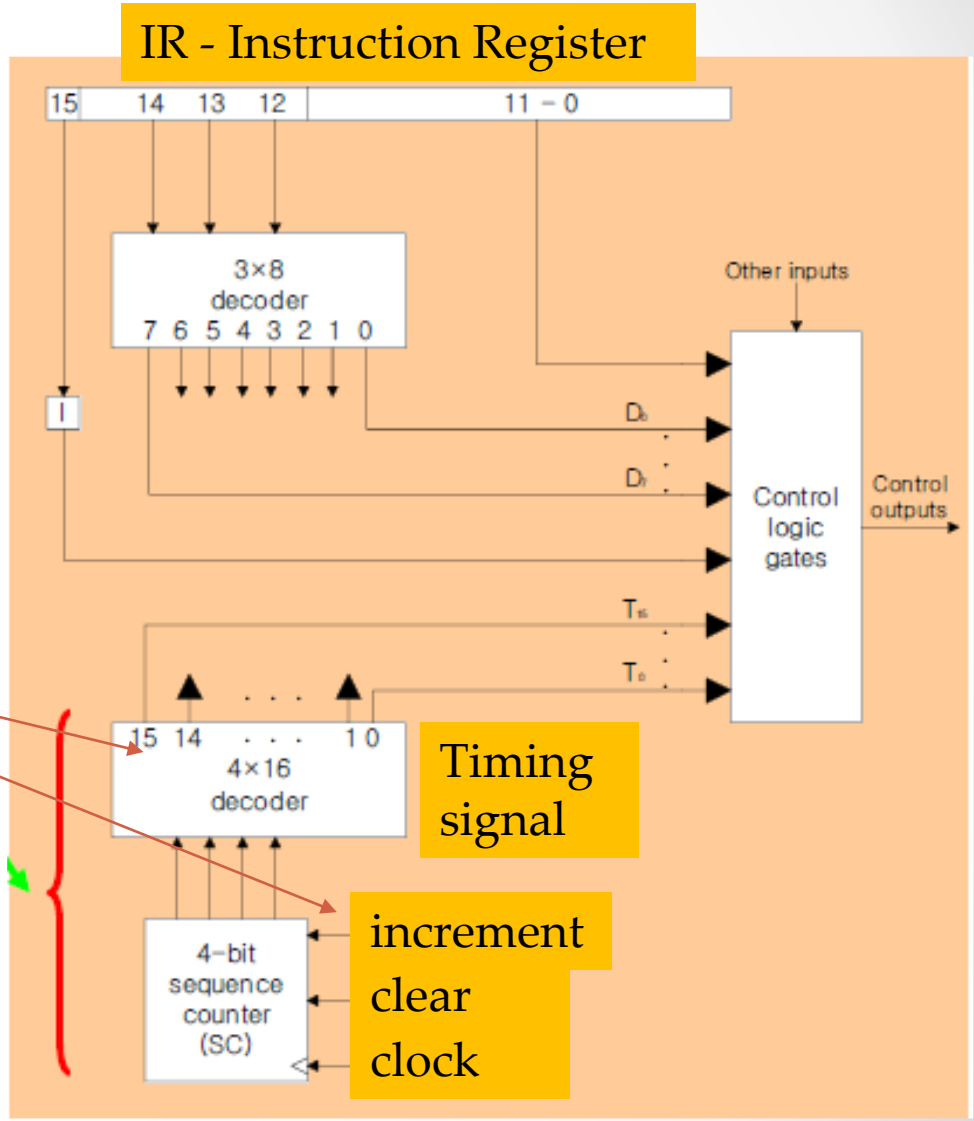
## Microprogrammed Control (chapter 7)

- The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
- + Any required change can be done by updating the microprogram in control memory, - Slow operation

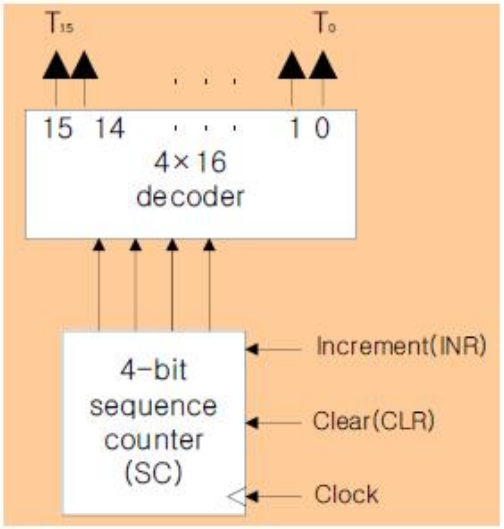
# The Control Unit

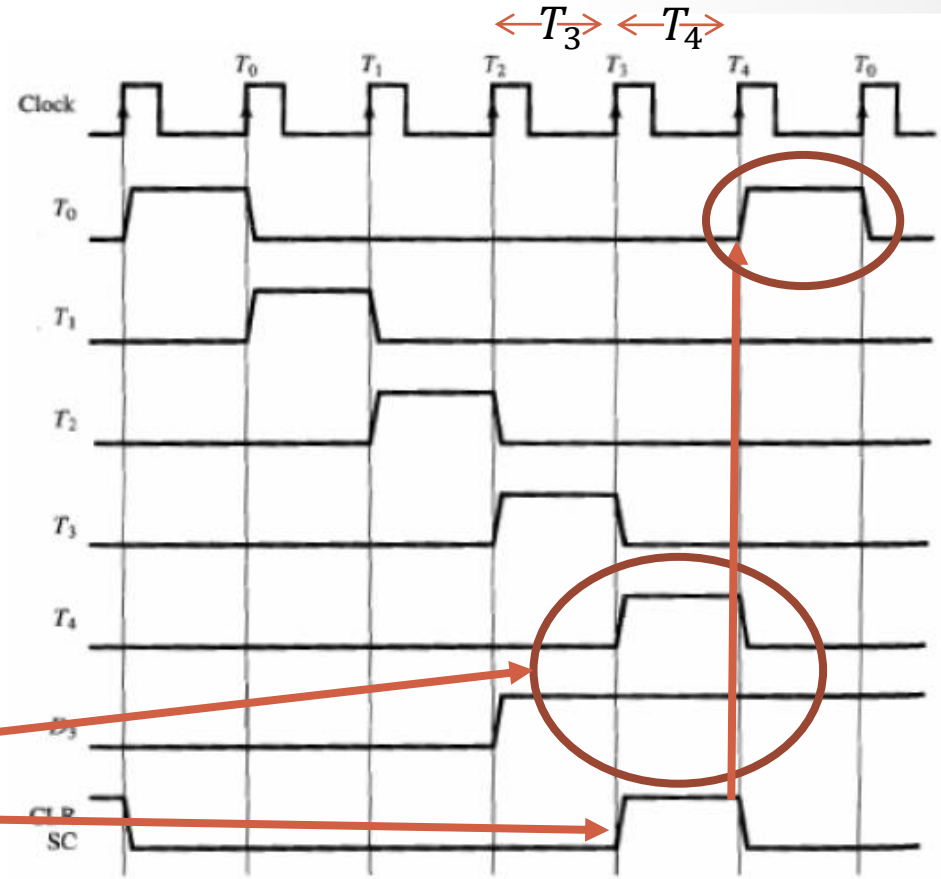
The Sequence Counter (SC) is incremented synchronously:
$T_0, T_1, T_2, T_3, ..$

The SC is cleared when in RTL we write symbolically:
$SC \leftarrow 0$



# The Control Unit




$$D_3T_4 : SC \leftarrow 0$$

**Figure 5-7** Example of control timing signals.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

# Clock and Memory

- The memory read/write cycle is initiated with the rising edge of a timing signal.
- It is assumed that a memory cycle time is less than the clock cycle time.
- The memory R/W cycle is complete by the time the next clock goes through its positive cycle.
- This assumption is not valid in most of computers.

# Clock and Timing Signals

Example:

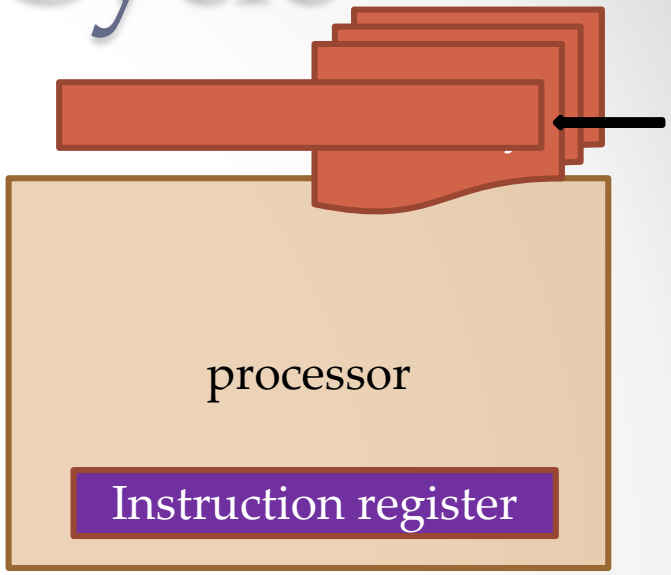
$$T_0 : AR \leftarrow PC$$

1. $T_0$ is active during the entire clock cycle interval.
2. During this time the content of the PC is placed onto the bus, and the LD of AR is enabled.
3. The actual transfer occurs when the clock goes through a positive transition (end of cycle).
4. On positive transition SC goes from 0000 to 0001.
5. $T_1$ is active and $T_0$ is inactive.

# Instruction Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory



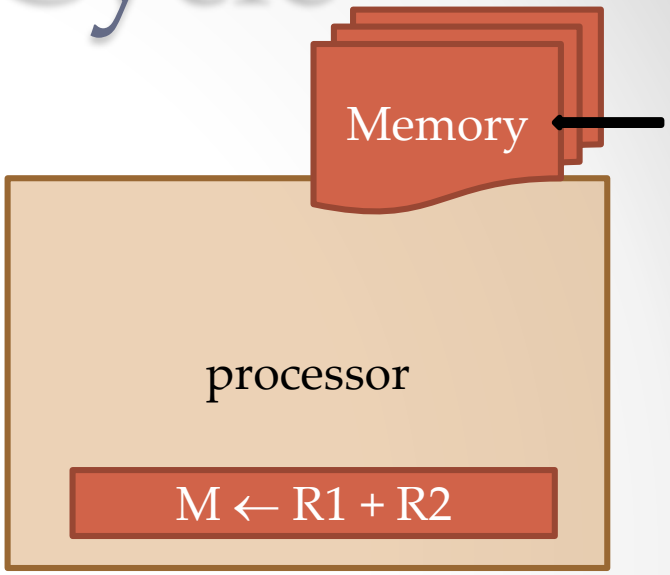
# Instruction Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory

*Instruction Decode*

Determine required actions and instruction size



# Instruction Cycle

*Instruction Fetch*

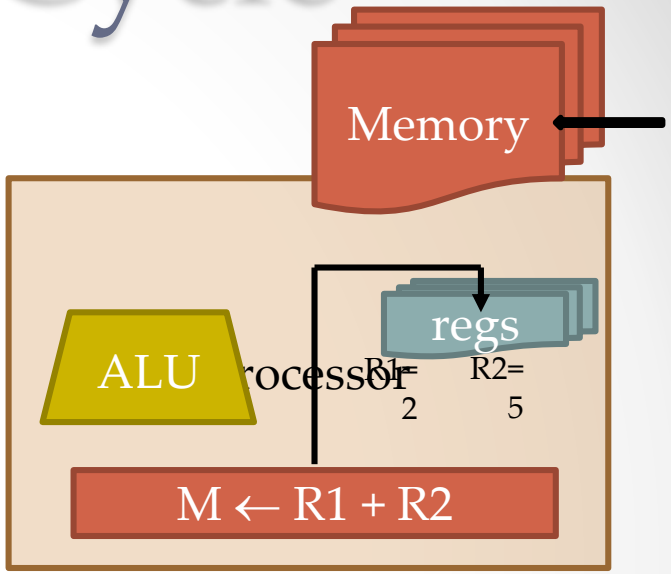
Obtain instruction from program storage in memory

*Instruction Decode*

Determine required actions and instruction size

*Operand Fetch*

Locate and obtain operand data



# Instruction Cycle

*Instruction Fetch*

Obtain instruction from program storage in memory

*Instruction Decode*

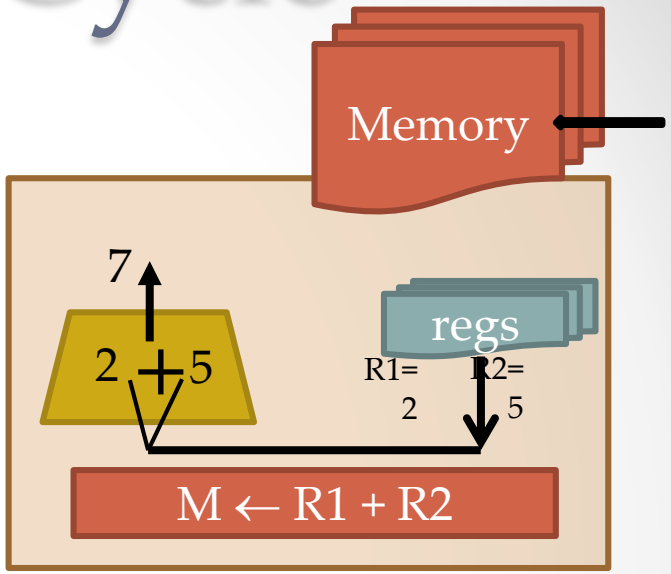
Determine required actions and instruction size

*Operand Fetch*

Locate and obtain operand data

*Execute*

Compute result value or status



# Instruction Cycle

*Instruction Fetch* → *Instruction Decode* → *Operand Fetch* → *Execute* → *Result Store*

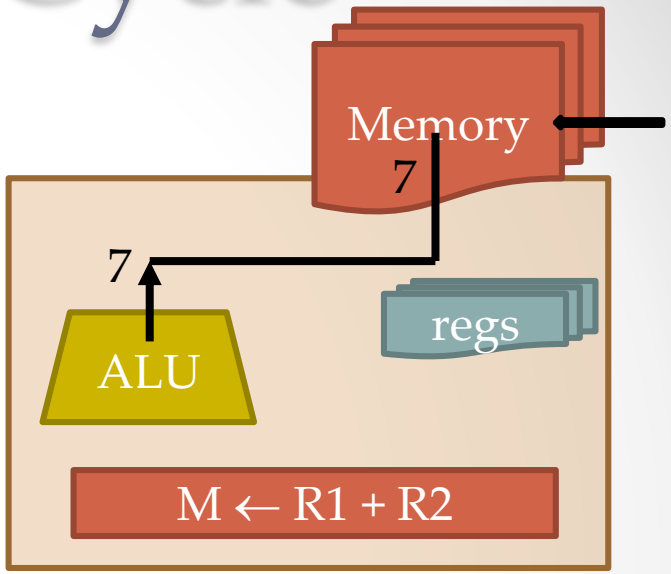
Obtain instruction from program storage in memory

Determine required actions and instruction size

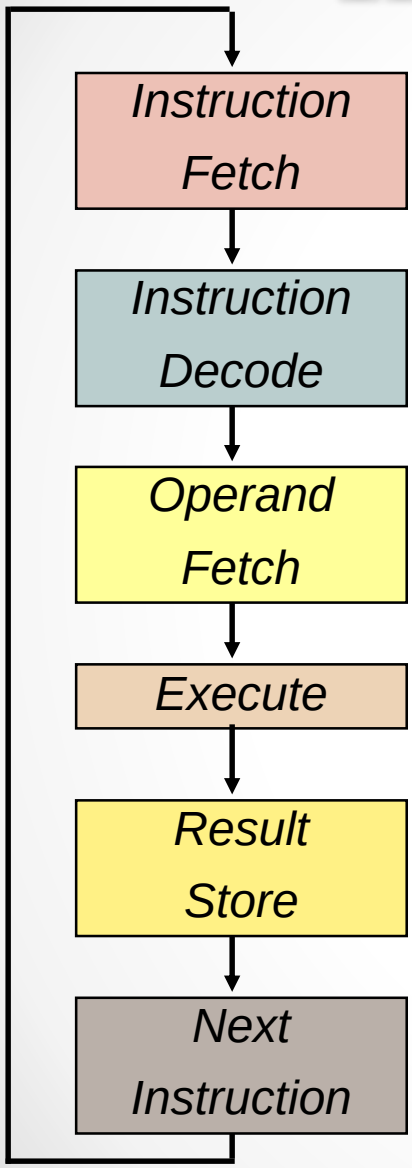
Locate and obtain operand data

Compute result value or status

Deposit results in storage



# Instruction Cycle




Obtain instruction from program storage in memory

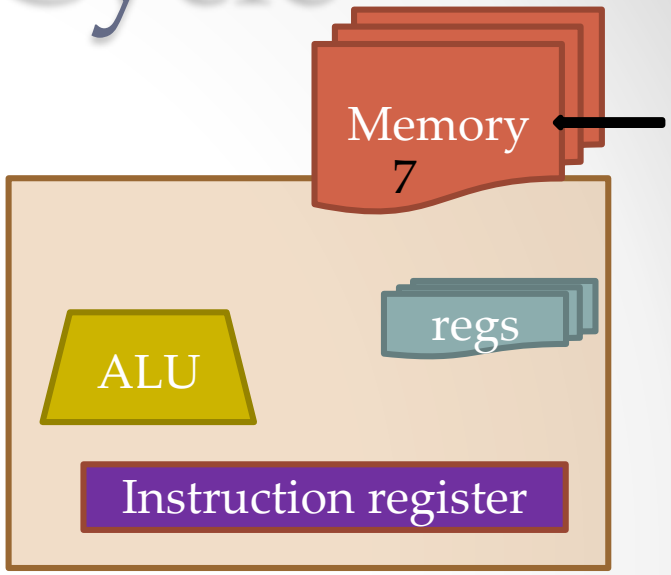
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage

Determine next instruction (not the next in case of **branch**)

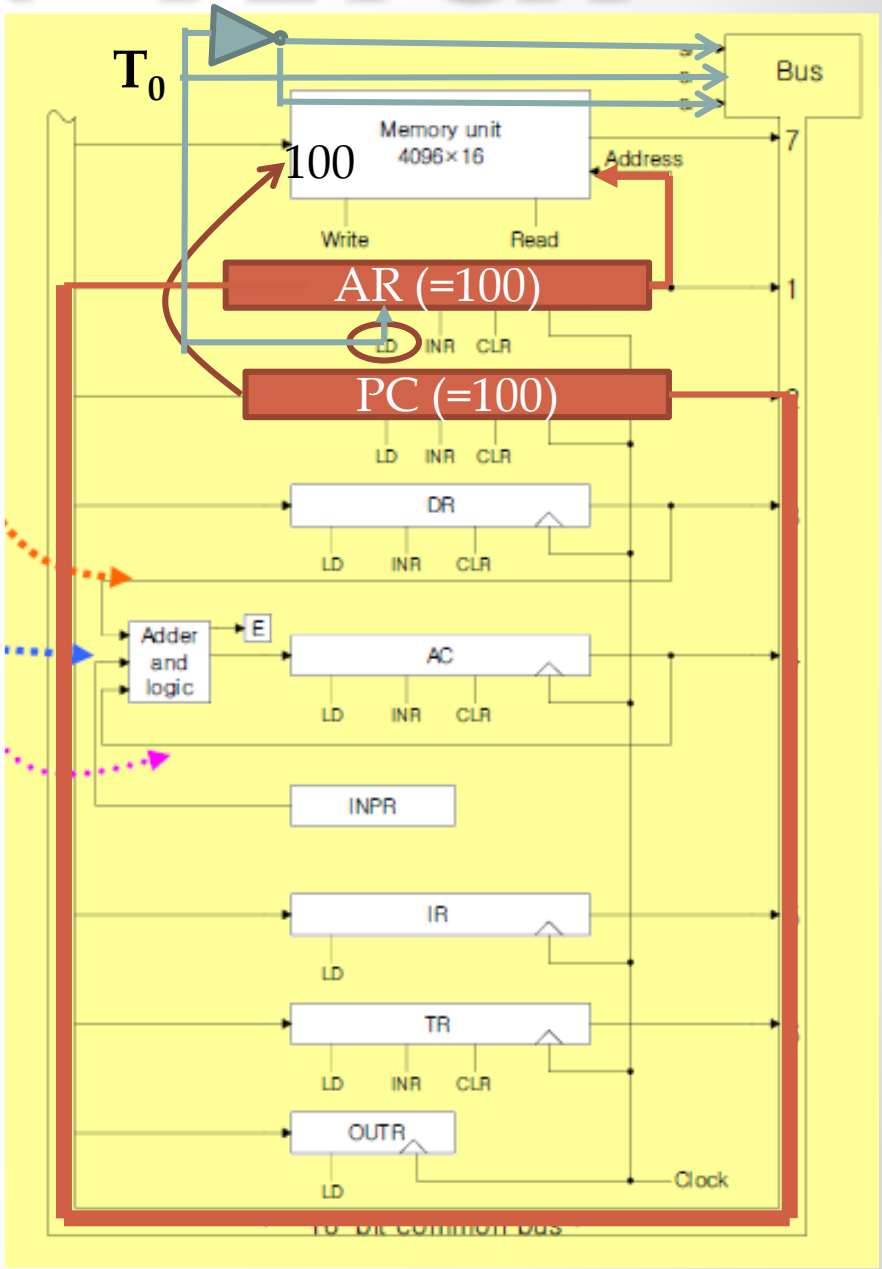


# Instruction - FETCH

*Instruction Fetch*

Obtain instruction from program storage in memory

$$T_0 : AR \leftarrow PC$$



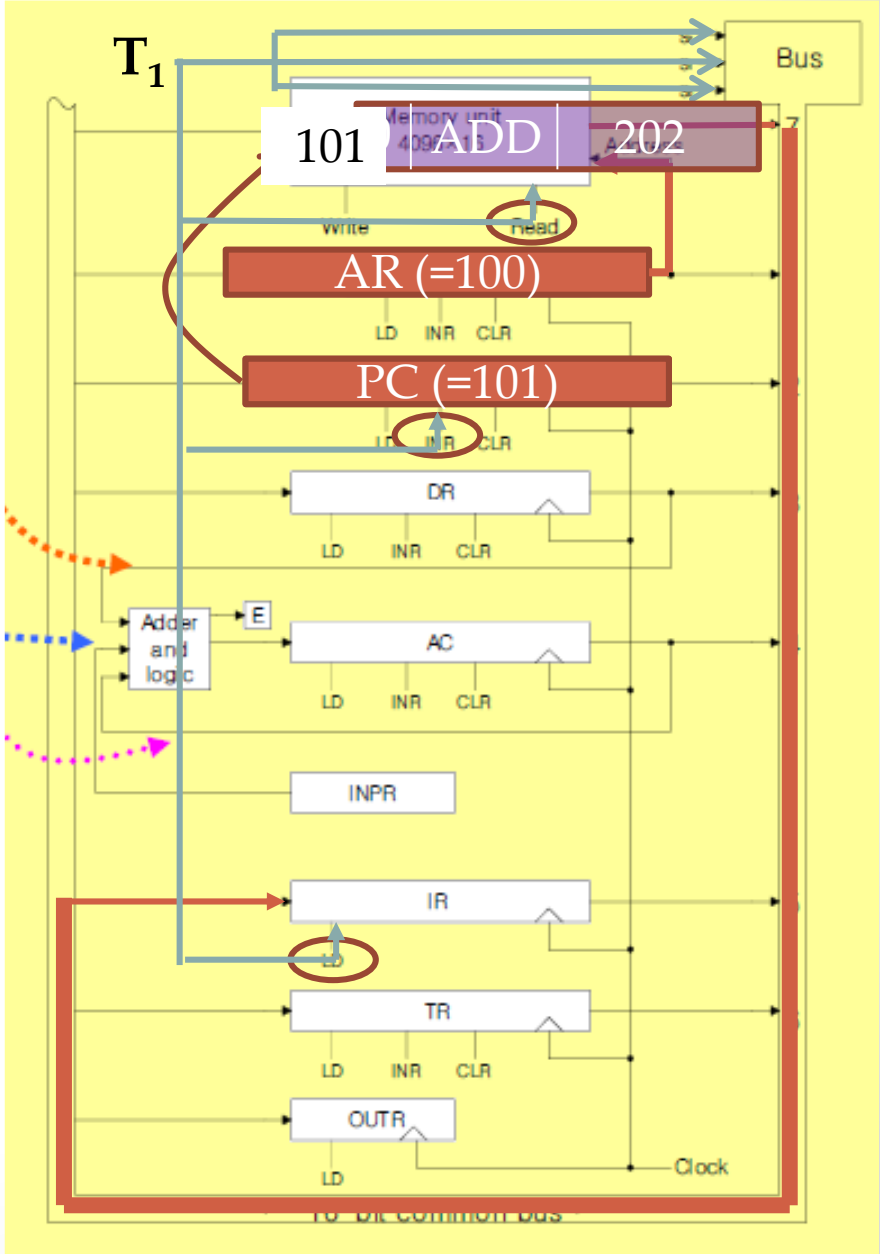
# Instruction - FETCH

*Instruction Fetch*

Obtain instruction from program storage in memory

$$T_0 : AR \leftarrow PC$$

$$T_1 : IR \leftarrow M[AR],$$
$$PC \leftarrow PC + 1$$



# Instruction - DECODE

*Instruction Fetch*

↓

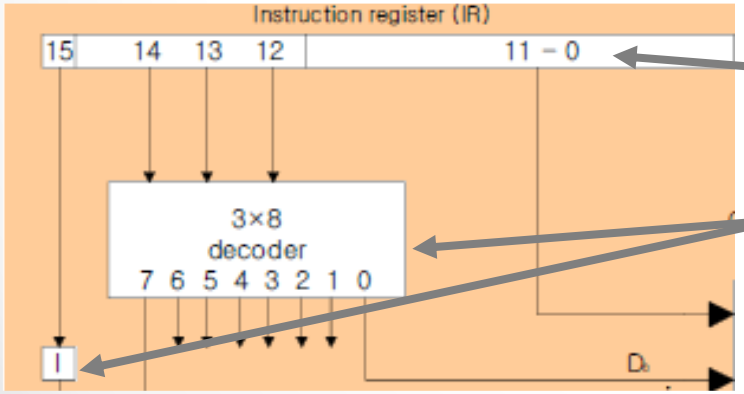
*Instruction Decode*

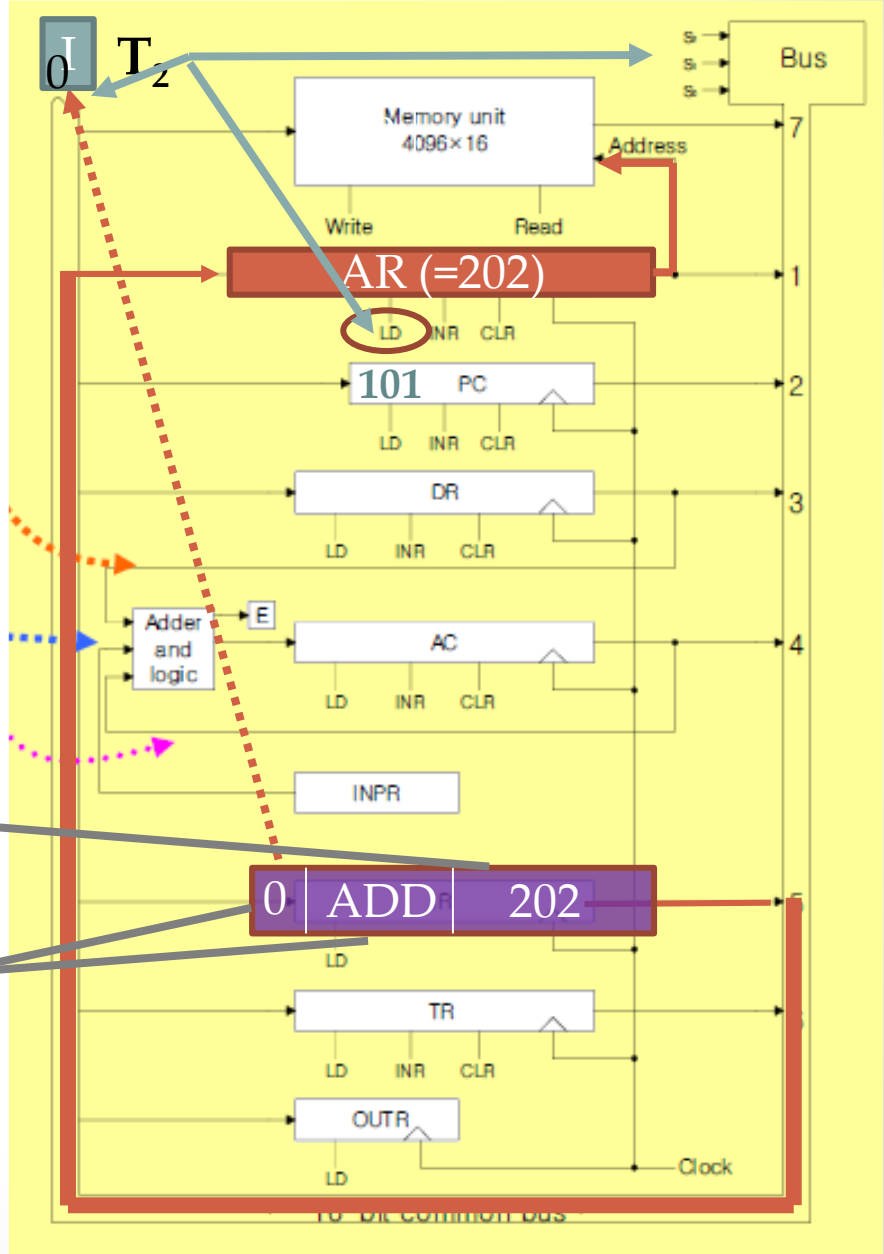
$T_0 : AR \leftarrow PC$

$T_1 : IR \leftarrow M[AR],$
$PC \leftarrow PC + 1$

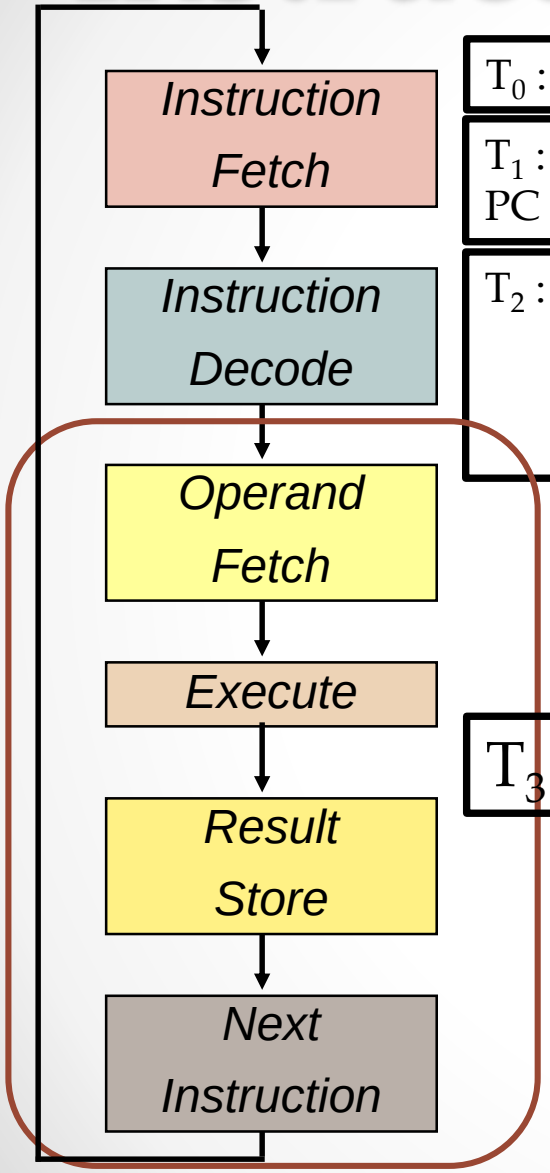
Determine required actions and instruction size

$T_2 : D_0, .., D_7 \leftarrow$ Decode IR(12-14)
$AR \leftarrow IR(0\text{-}11),$
$I \leftarrow IR(15)$



# Instruction - EXECUTION

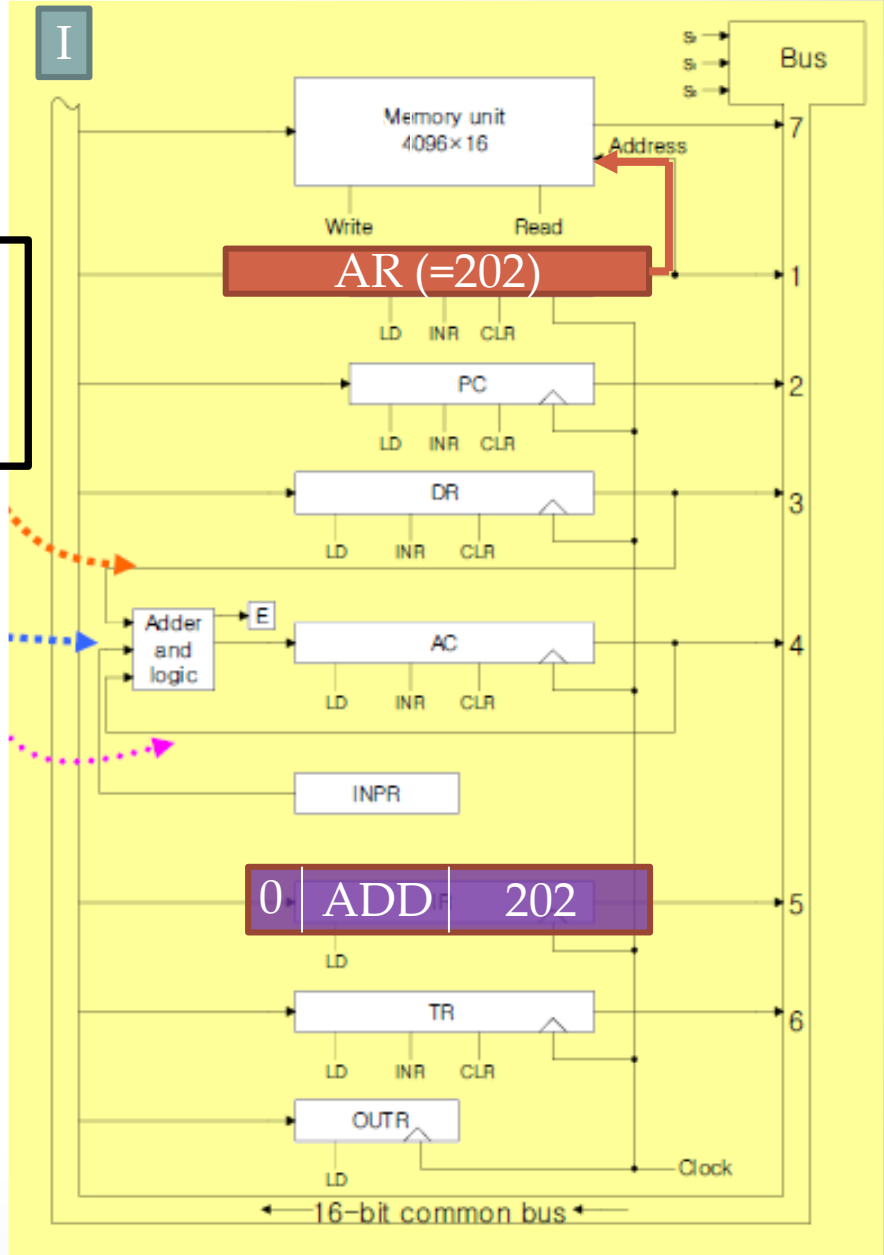



$T_0$ : AR ← PC

$T_1$ : IR ← M[AR], PC ← PC + 1

$T_2$ : $D_0$ ,.., $D_7$ ← Decode IR(12-14)
AR ← IR(0-11),
I ← IR(15)

$T_3$ $T_4$ $T_5$ $T_6$



# The Instruction Format

- 5-3 Computer Instruction
  - 3 Instruction Code Formats : ***Fig. 5-5***
    - Memory-reference instruction
      - » Opcode = 000 ~ 110
        - I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect

| 15 | 14 ... 12 | 11 ... 0 |
|---|---|---|
| I | Opcode | Address |

1/0

I

0

1

- Register-reference instruction
  - » 7xxx (7800 ~ 7001) : CLA, CMA, ....

| 15 | 14 | 13 | 12 | 11 ... 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | Register Operation |

- Input-Output instruction
  - » Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ....

| 15 | 14 | 13 | 12 | 11 ... 0 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | I/O Operation |

| Symbol | Hex Code I = 0 | Hex Code I = 1 | Description |
|---|---|---|---|
| AND | 0xxx | 8xxx | And memory word to AC |
| ADD | 1xxx | 9xxx | Add memory word to AC |
| LDA | 2xxx | Axxx | Load memory word to AC |
| STA | 3xxx | Bxxx | Store content of AC in memory |
| BUN | 4xxx | Cxxx | Branch unconditionally |
| BSA | 5xxx | Dxxx | Branch and Save return address |
| ISZ | 6xxx | Exxx | Increment and skip if zero |
| CLA | 7800 | | Clear AC |
| CLE | 7400 | | Clear E |
| CMA | 7200 | | Complement AC |
| CME | 7100 | | Comp m e |
| CIR | 7080 | | Circulate right AC and E |
| CIL | 7040 | | Circulate left AC and E |
| INC | 7020 | | Increment AC |
| SPA | 7010 | | Skip next instruction if AC positive |
| SNA | 7008 | | Skip next instruction if AC negative |
| SZA | 7004 | | Skip next instruction if AC zero |
| SZE | 7002 | | Skip next instruction if E is 0 |
| HLT | 7001 | | Halt computer |
| INP | F800 | | Input character to AC |
| OUT | F400 | | Output character from AC |
| SKI | F200 | | Skip on input flag |
| SKO | F100 | | Skip on output flag |
| ION | F080 | | Interrup |
| IOF | F040 | | Inter |

# The Instruction Format

- 5-3 Computer Instruction
  - 3 Instruction Code Formats : ***Fig. 5-5***
    - Memory-reference instruction
      - » Opcode = 000 ~ 110
        - I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect

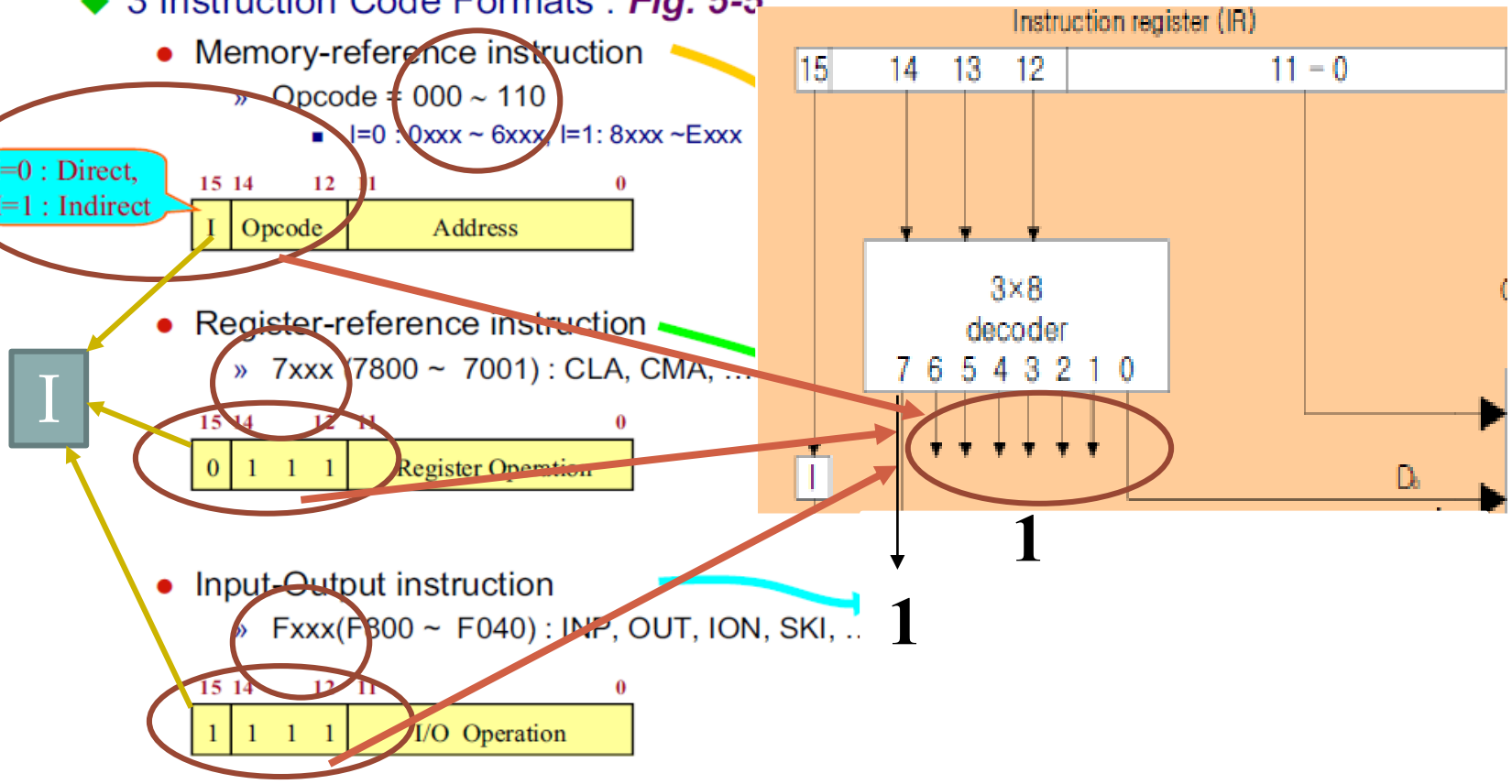
| 15 | 14 – 12 | 11 – 0 |
|---|---|---|
| I | Opcode | Address |

- Register-reference instruction
  - » 7xxx (7800 ~ 7001) : CLA, CMA, …

| 15 | 14 | 13 | 12 | 11 – 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | Register Operation |

- Input-Output instruction
  - » Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ..

| 15 | 14 | 13 | 12 | 11 – 0 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | I/O Operation |



# Determine Instruction Type

- 5-3 Computer Instruction
  - 3 Instruction Code Formats : F
    - Memory-reference instruction
      - » Opcode = 000 ~ 110
        - I=0 : 0xxx ~ 6xxx, I=1: 8xx

I=0 : Direct,
I=1 : Indirect

| 15 | 14 … 12 | 11 … 0 |
|---|---|---|
| I | Opcode | Address |

I

- Register-reference instruction
  - » 7xxx (7800 ~ 7001) : CLA, C

| 15 | 14 | 13 | 12 | 11 … 0 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | Register Operation |

- Input-Output instruction
  - » Fxxx(F800 ~ F040) : INP, O

| 15 | 14 | 13 | 12 | 11 … 0 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | I/O Operation |



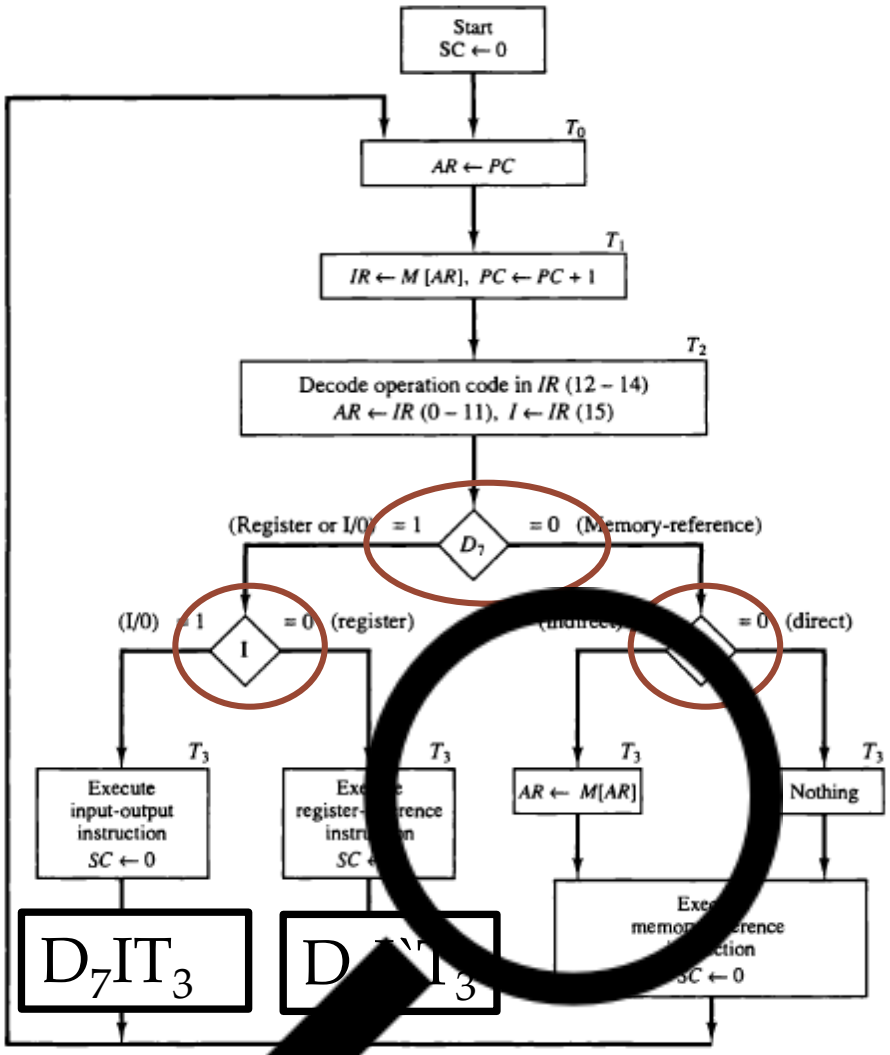
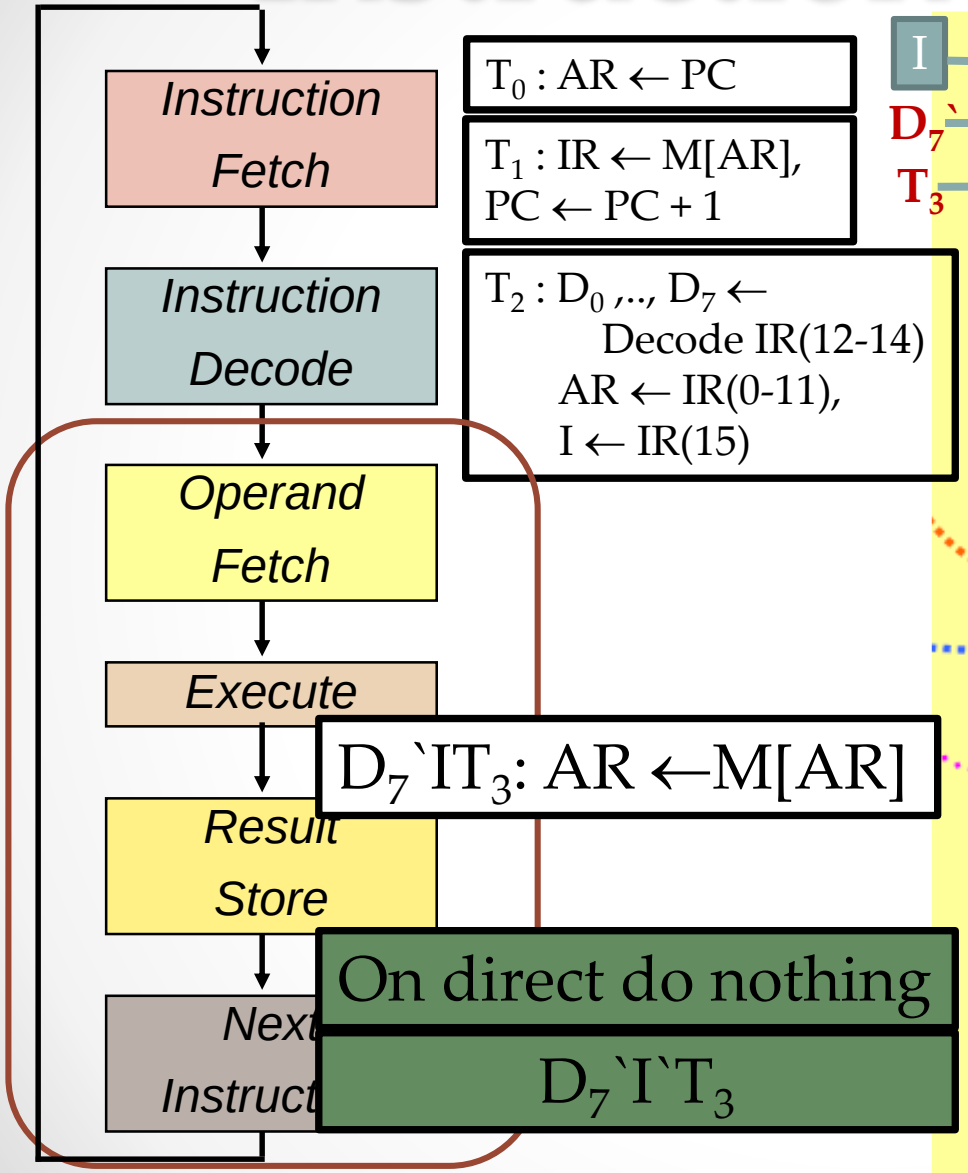



Figure 5-9 rt for instruction cycle (initial configuration).

# Instruction - Indirect




$T_0 : AR \leftarrow PC$

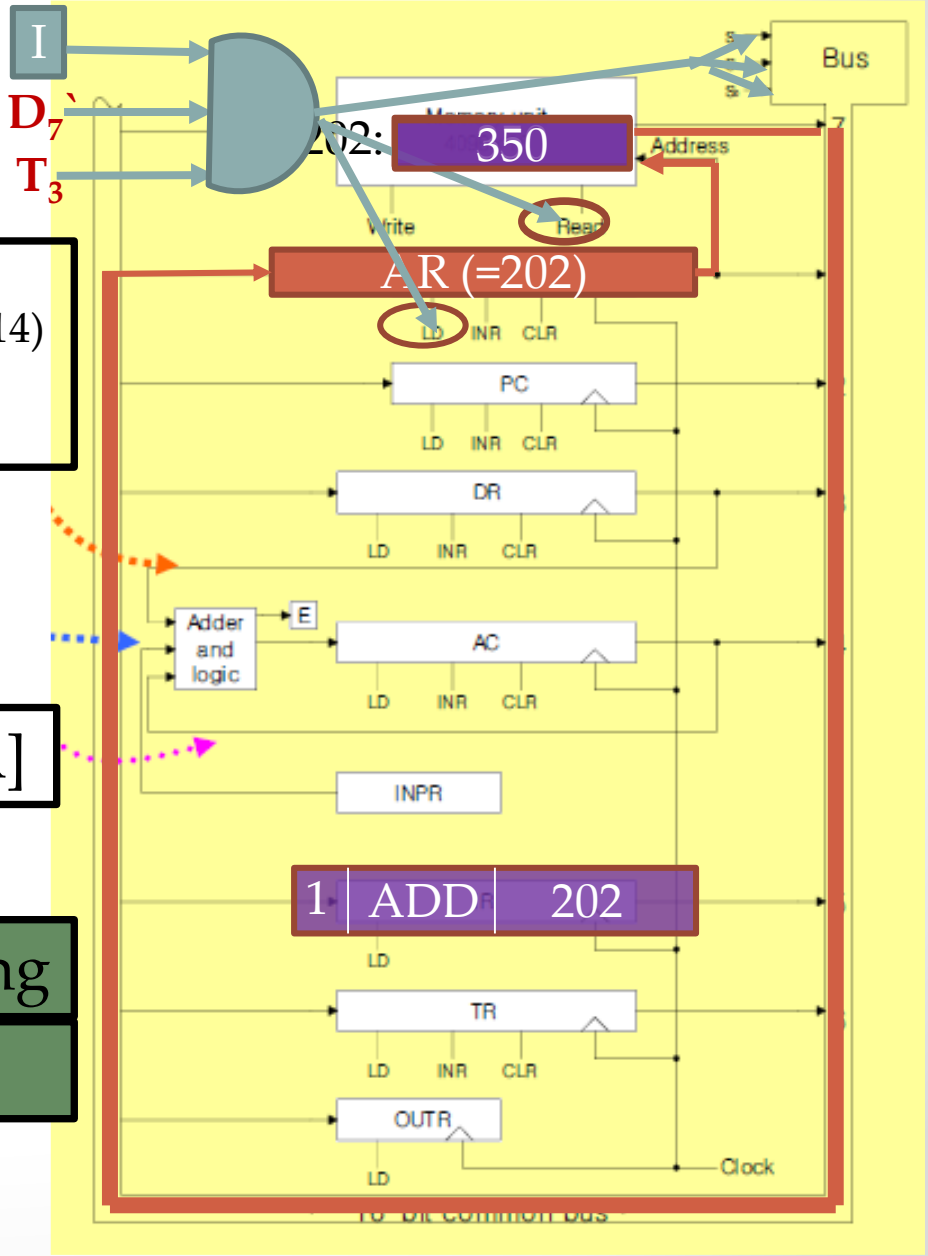
$T_1 : IR \leftarrow M[AR],$
$PC \leftarrow PC + 1$

$T_2 : D_0, .., D_7 \leftarrow$ Decode IR(12-14)
$AR \leftarrow IR(0-11),$
$I \leftarrow IR(15)$

$D_7`IT_3$: $AR \leftarrow M[AR]$

On direct do nothing

$D_7`I`T_3$



# Register Reference Instructions

- Executed with the clock transition associated with timing variable $T_3$.
- Each control function needs the Boolean relation $D_7I'T_3$
  - for convenience let $r \equiv D_7I'T_3$.
- The control function is distinguished by one of the bits in **IR(0-11)**.
- Assign the symbol $B_i$ to bit $i$ of **IR**,
  - all control functions can be simply denoted by $rB_i$.
- After completion
  - The sequence counter **SC** is cleared to **0**
  - The control goes back to fetch the next instruction with timing signal $T_0$.

**Alon Schclar, Tel-Aviv College, 2009**

# Register Reference Instructions

**TABLE 5-3** Execution of Register-Reference Instructions

$D_7 I' T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

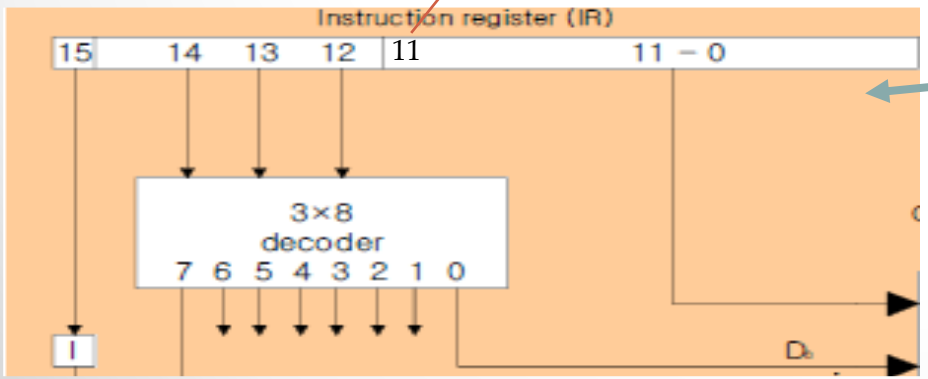
| | | | |
|---|---|---|---|
| | $r$: | $SC \leftarrow 0$ | Clear $SC$ |
| CLA | $rB_{11}$: | $AC \leftarrow 0$ | Clear $AC$ |
| CLE | $rB_{10}$: | $E \leftarrow 0$ | Clear $E$ |
| CMA | $rB_9$: | $AC \leftarrow \overline{AC}$ | Complement $AC$ |
| CME | $rB_8$: | $E \leftarrow \overline{E}$ | Complement $E$ |
| CIR | $rB_7$: | $AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$ | Circulate right |
| CIL | $rB_6$: | $AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$ | Circulate left |
| INC | $rB_5$: | $AC \leftarrow AC + 1$ | Increment $AC$ |
| SPA | $rB_4$: | If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$ | Skip if positive |
| SNA | $rB_3$: | If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$ | Skip if negative |
| SZA | $rB_2$: | If $(AC = 0)$ then $PC \leftarrow PC + 1)$ | Skip if $AC$ zero |
| SZE | $rB_1$: | If $(E = 0)$ then $(PC \leftarrow PC + 1)$ | Skip if $E$ zero |
| HLT | $rB_0$: | $S \leftarrow 0$ ($S$ is a start–stop flip-flop) | Halt computer |

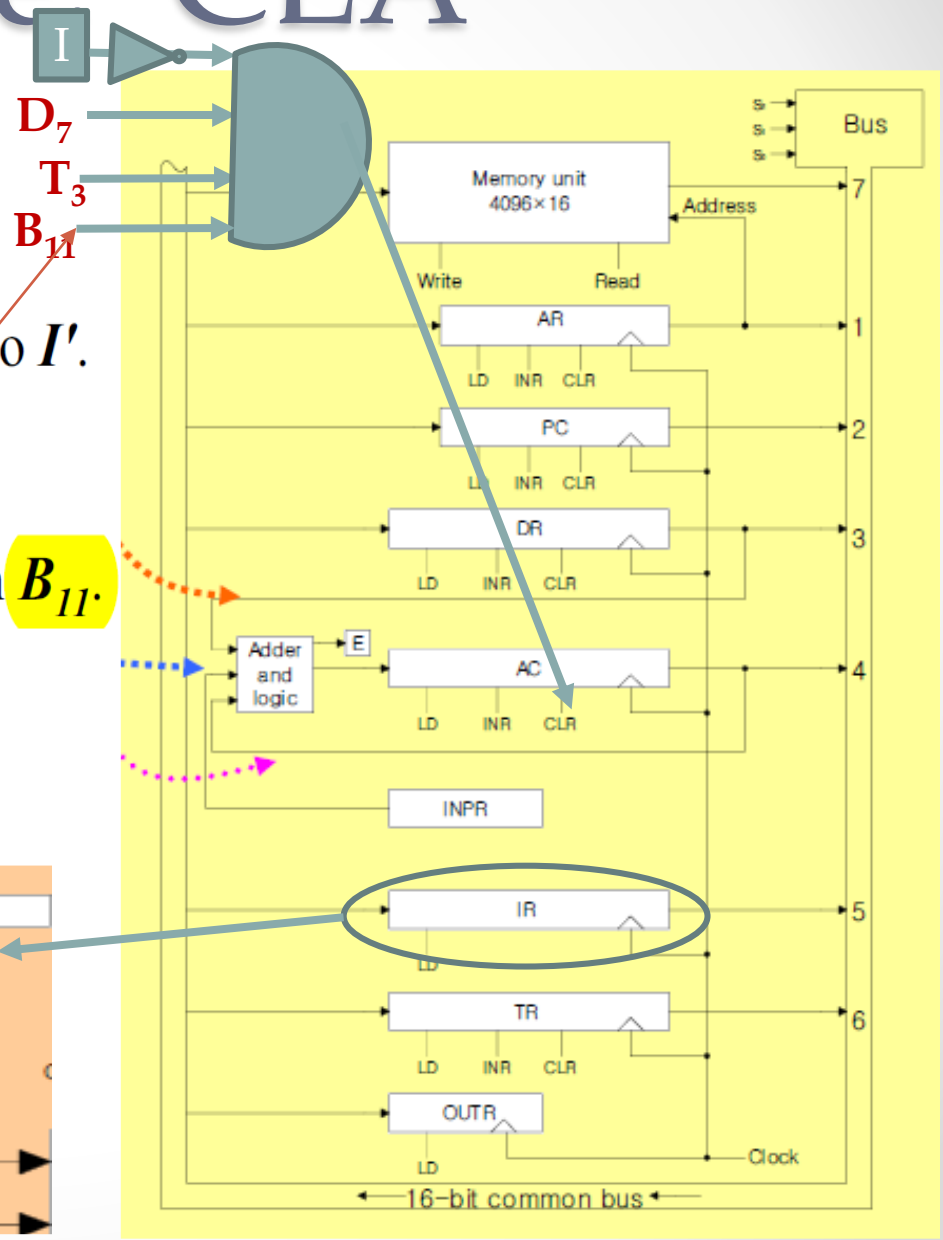
if-else loops

Taken from: M. Mano/Computer Design and Architecture 3rd Ed.

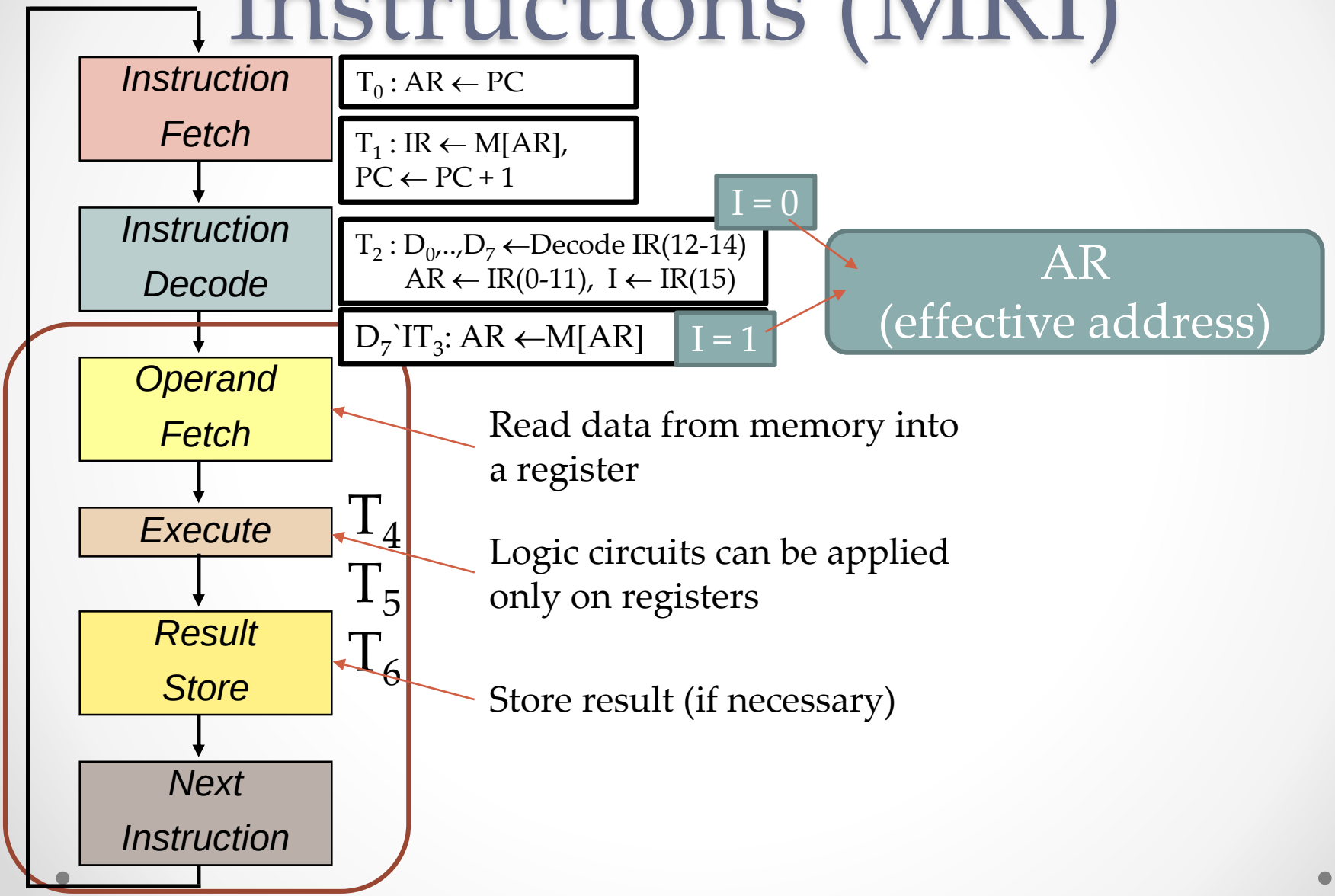
# Example: CLA

- Hexadecimal code $(7800)_{16} = (0111\ 1000\ 0000\ 0000)_2$.
- The first bit is a zero and is equivalent to $I'$.
- Next three bits are the opcode and are recognized from decoder output $D_7$.
- Bit *11* in **IR** is *1* and is recognized from $B_{11}$.
- The control function that initiates the microoperation for this instruction is $D_7I'T_3B_{11} = rB_{11}$.



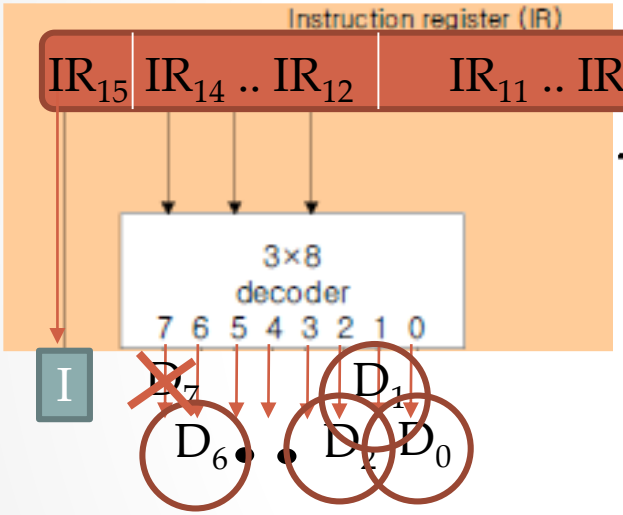



# Memory Reference Instructions (MRI)



# Memory Reference Instructions (MRI)

At the end of the $T_3$ cycle, AR holds the effective Address




| Symbol | Operation decoder | Symbolic description |
|---|---|---|
| AND | $D_0$ | $AC \leftarrow AC \wedge M[AR]$ |
| ADD | $D_1$ | $AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$ |
| LDA | $D_2$ | $AC \leftarrow M[AR]$ |
| STA | $D_3$ | $M[AR] \leftarrow AC$ |
| BUN | $D_4$ | $PC \leftarrow AR$ |
| BSA | $D_5$ | $M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$ |
| ISZ | $D_6$ | $M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$ |

# MRI - AND

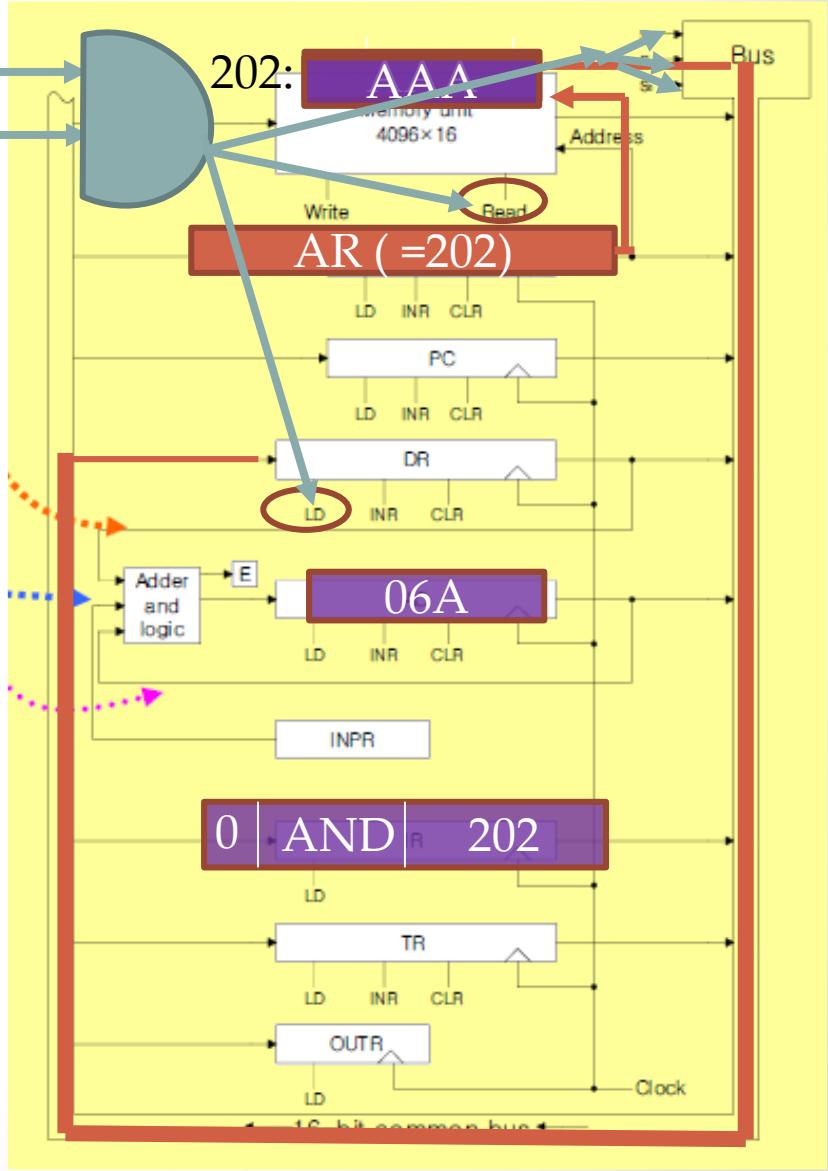
At the end of the $T_3$ cycle, AR holds the **<u>effective Address</u>**

$D_0$

$T_4$

$$\text{AND} : AC \leftarrow AC \wedge M[AR]$$

$$D_0T_4 : \; DR \leftarrow M[AR]$$
$$D_0T_5 : \; AC \leftarrow AC \wedge DR, \; SC \leftarrow 0$$



# MRI - AND

At the end of the $T_3$ cycle, AR holds the **effective Address**

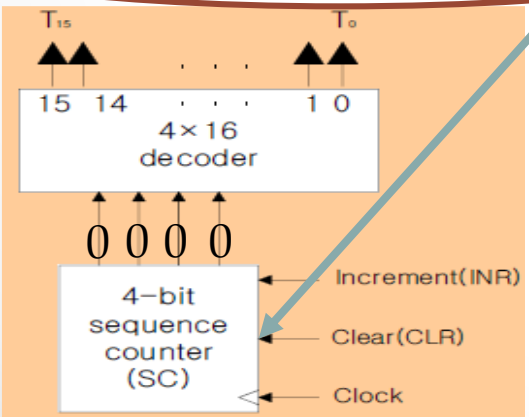
$D_0$

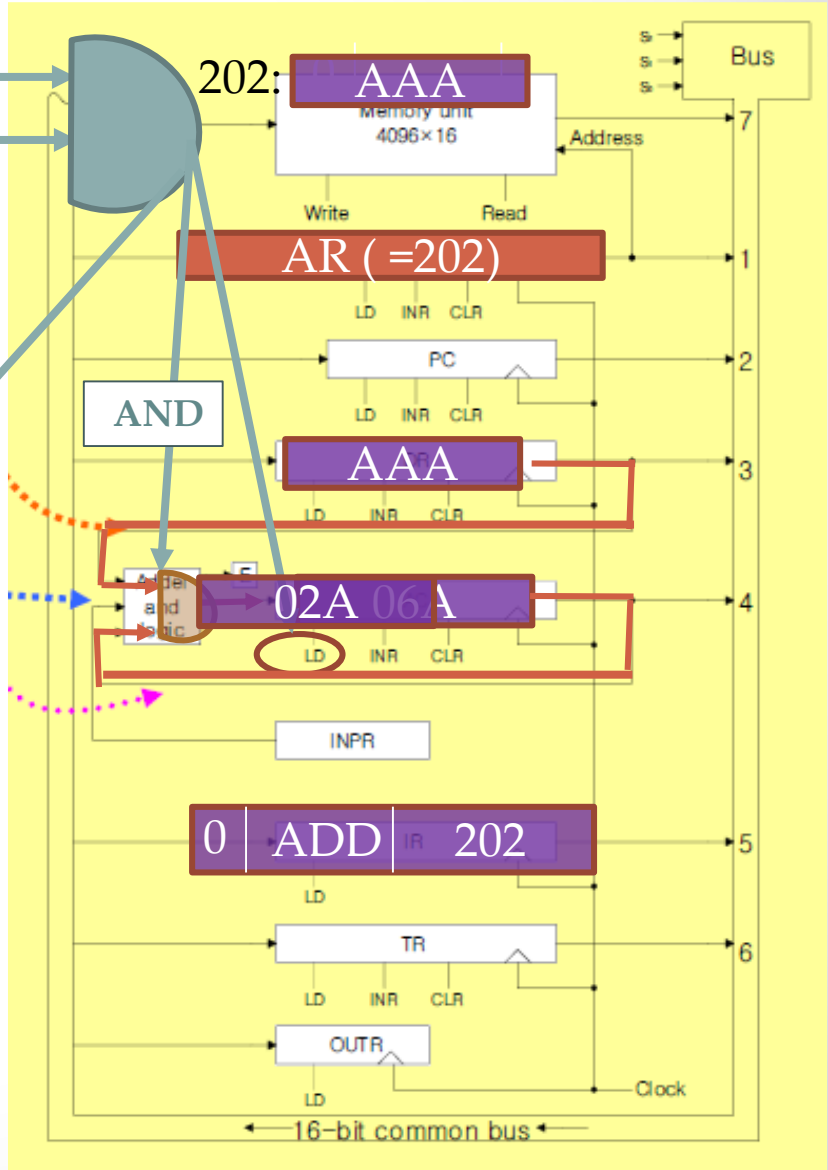
$T_5$

$$\text{AND} : AC \leftarrow AC \wedge M[AR]$$

$$D_0T_4 : \; DR \leftarrow M[AR]$$

$$D_0T_5 : \; AC \leftarrow AC \wedge DR, \; SC \leftarrow 0$$



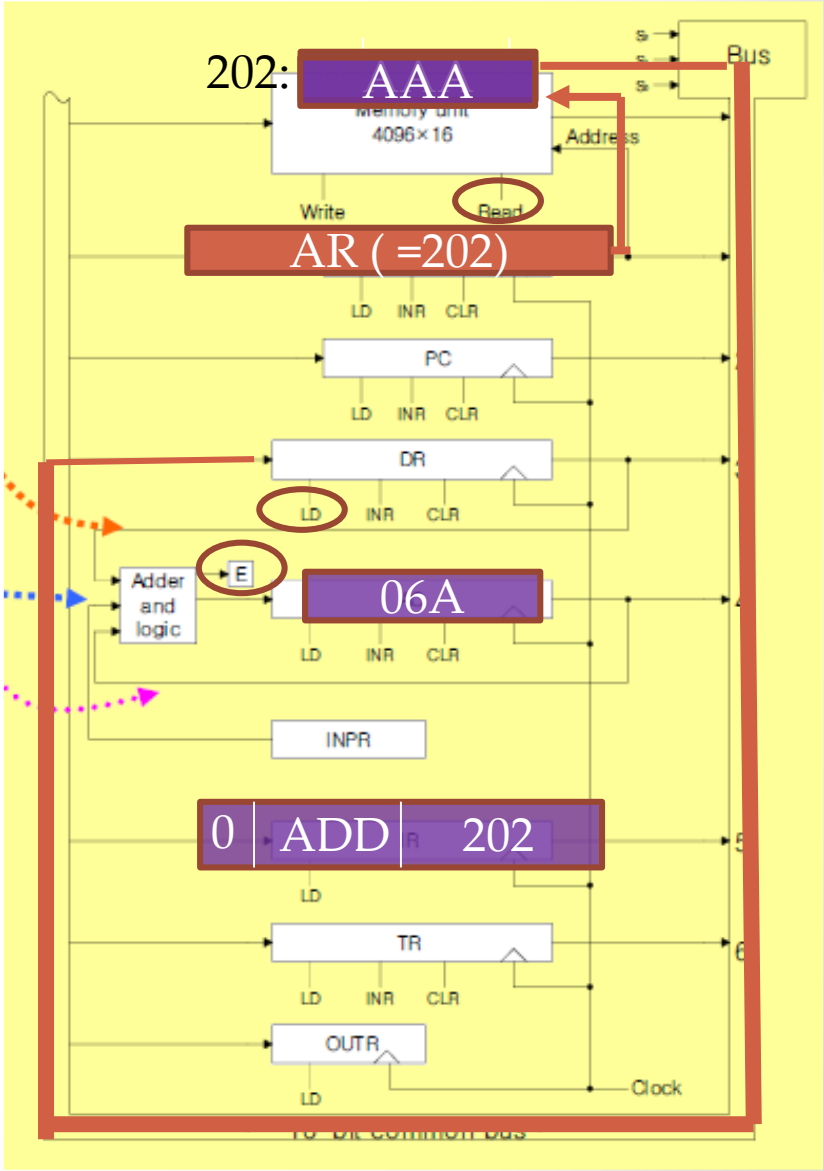



# MRI - ADD

At the end of the $T_3$ cycle, AR holds the **<u>effective Address</u>**

$$\text{ADD}: AC \leftarrow AC + M[AR],\quad E \leftarrow C_{out}$$

$$D_1T_4: \; DR \leftarrow M[AR]$$
$$D_1T_5: \; AC \leftarrow AC + DR,\quad E \leftarrow C_{out},\; SC \leftarrow 0$$



# MRI - LDA

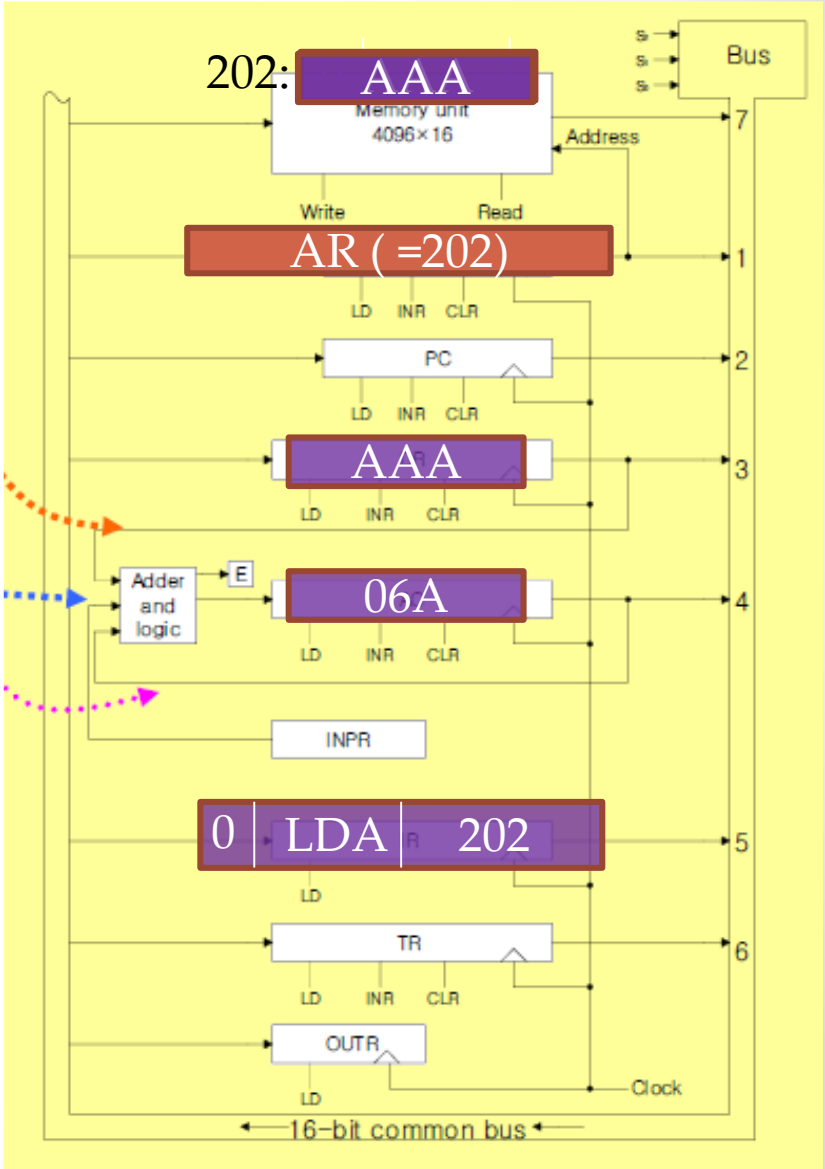
At the end of the $T_3$ cycle, AR holds the **<u>effective Address</u>**

$$\text{LDA} : AC \leftarrow M[AR]$$

$$D_2T_4 : \; DR \leftarrow M[AR]$$
$$D_2T_5 : \; AC \leftarrow DR, \; SC \leftarrow 0$$

Why not connecting the bus to the inputs of ***AC*** ?

- a delay is encountered in the adder and logic circuit.
  ***Time(Mem read) + Time(Bus transfer) + Time(A&L) > 1 cycle***
- Not connecting the bus to the inputs of AC maintains **one clock cycle per microoperation**.

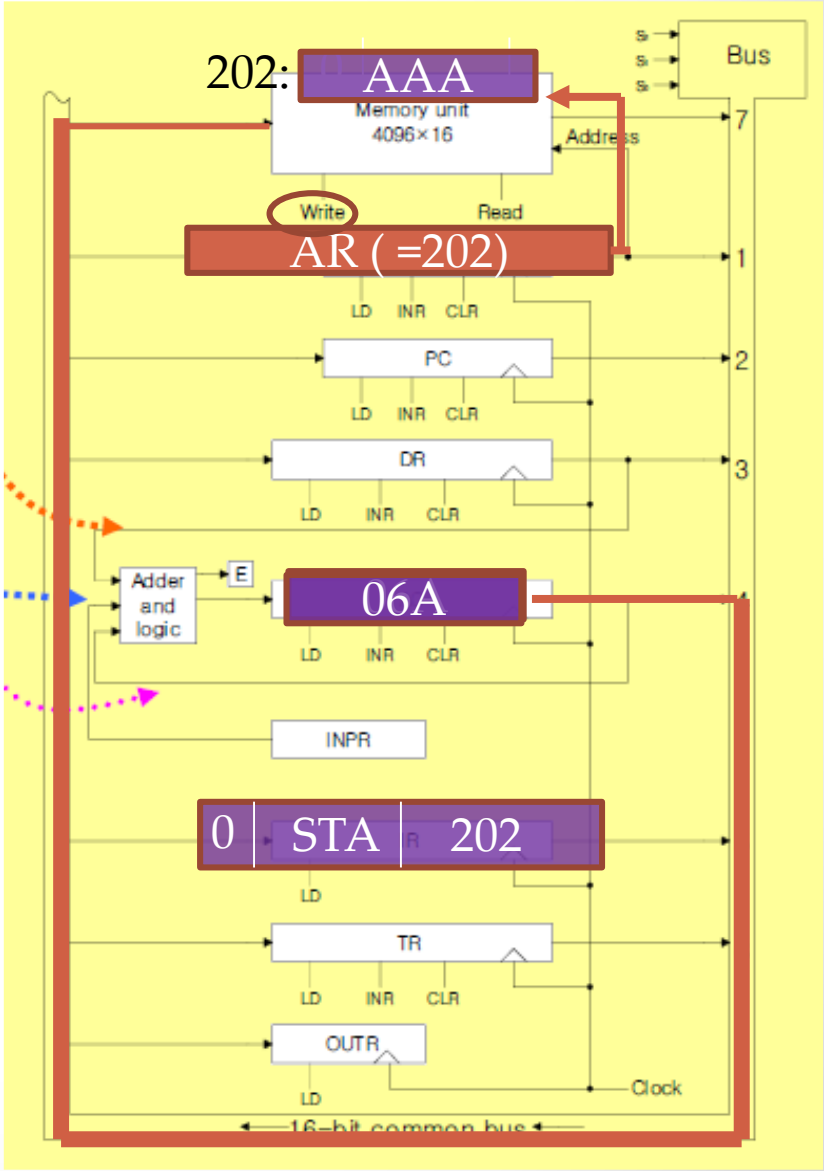


# MRI - STA

At the end of the $T_3$ cycle, AR holds the **<u>effective Address</u>**

$$STA : M[AR] \leftarrow AC$$

$$D_3T_4 : \; M[AR] \leftarrow AC, SC \leftarrow 0$$



# MRI - BUN

PC = 150

PC = 151

PC = 152

PC = 153

$T_0$ : AR ← PC

$T_1$ : IR ← M[AR]

PC ← PC + 1

PC = 200

$T_2$ : $D_0$,..,$D_7$ ←Decode IR(12-14)

AR ← IR(0-11), I ← IR(15)

$D_7$`$IT_3$: AR ←M[AR]

$D_4T_4$ : PC ← AR, SC ← 0

150 : ADD 230

151 : INC

152 : BUN 200

…

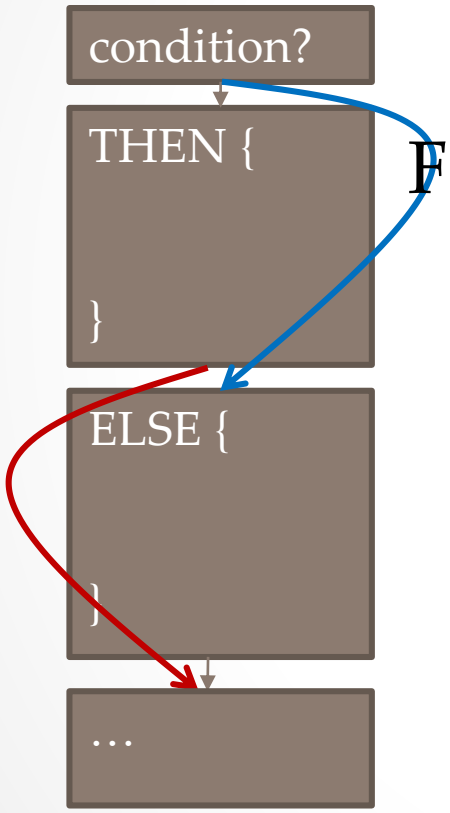
AR = 200

200 : AND AB0

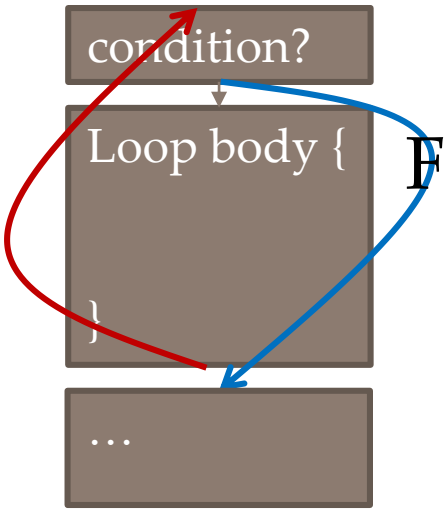
# Conditional & Unconditional Jumps

conditional → unconditional →

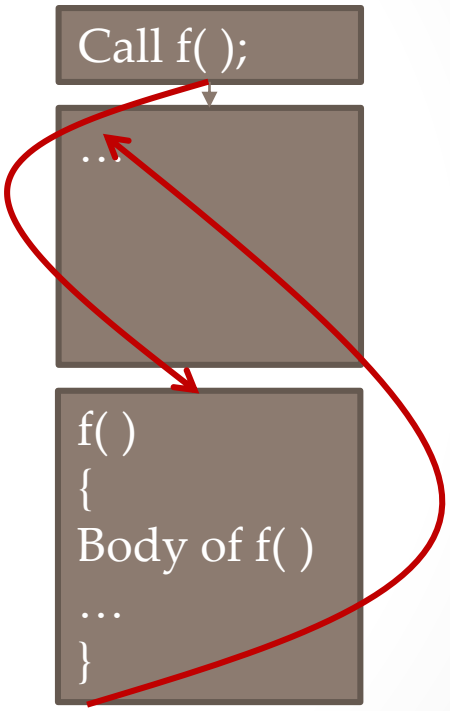
If..then..else




loops




subroutines




for ( ),
while,
do .. while

goto

switch / case

# MRI - BSA

Branch to subroutine and save the return address

$BSA : M[AR] \leftarrow PC, PC \leftarrow AR + 1$

$T_0 : AR \leftarrow PC$

$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$

$T_2 : D_0,..,D_7 \leftarrow$ Decode $IR(12\text{-}14)$
$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$

$D_7`IT_3 : AR \leftarrow M[AR]$

$D_5T_4 : M[AR] \leftarrow PC, AR \leftarrow AR + 1$

$D_5T_5 : PC \leftarrow AR, SC \leftarrow 0$

main

150 : BSA 230 //subroutine call

151 : INC

…

230 : 151 [redacted] routine begins here

231 : …

232 : …

…

255 : 1 BUN 230 //subroutine ends here

PC = 231

AR = 231

IR = BSA 230

# MRI – BUN (cont. BSA)

Branch to subroutine and save the return address

BSA : M[AR] ← PC, PC ← AR + 1

$T_0$ : AR ← PC

$T_1$ : IR ← M[AR], PC ← PC + 1

$T_2$ : $D_0$,..,$D_7$ ←Decode IR(12-14)
AR ← IR(0-11), I ← IR(15)

$D_7$`$IT_3$: AR ←M[AR]

$D_4T_4$ : PC ← AR, SC ← 0

main

150 : BSA 230 //subroutine call
151 : INC
…

230 : 151 [redacted]routine begins here
231 : …
232 : …

…

255 : 1 BUN 230 //subroutine ends here

PC = 151

AR = 151

IR = 1 BUN 230

I = 1

# MRI - ISZ

Increment memory word specified by the effective address

- if the incremented value is equal to 0, PC is incremented by 1.

Useful for loop indices:

- Place a negative number in memory word
- Increment with each loop iteration
- eventually reaches the value of zero
- **At that time PC is incremented by one in order to skip the next instruction in the program.**

No single microoperation to increment a word inside the memory

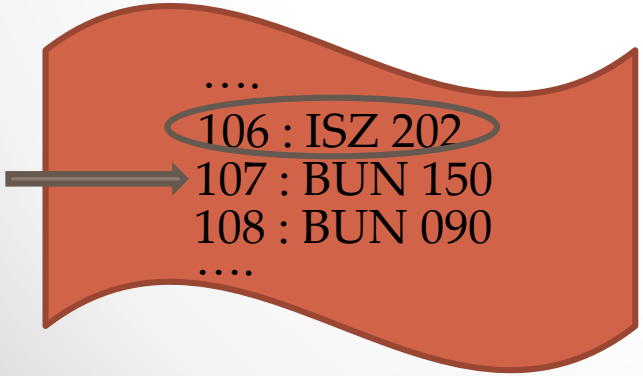
- First read the word into DR,
- increment DR,
- store the word back into memory

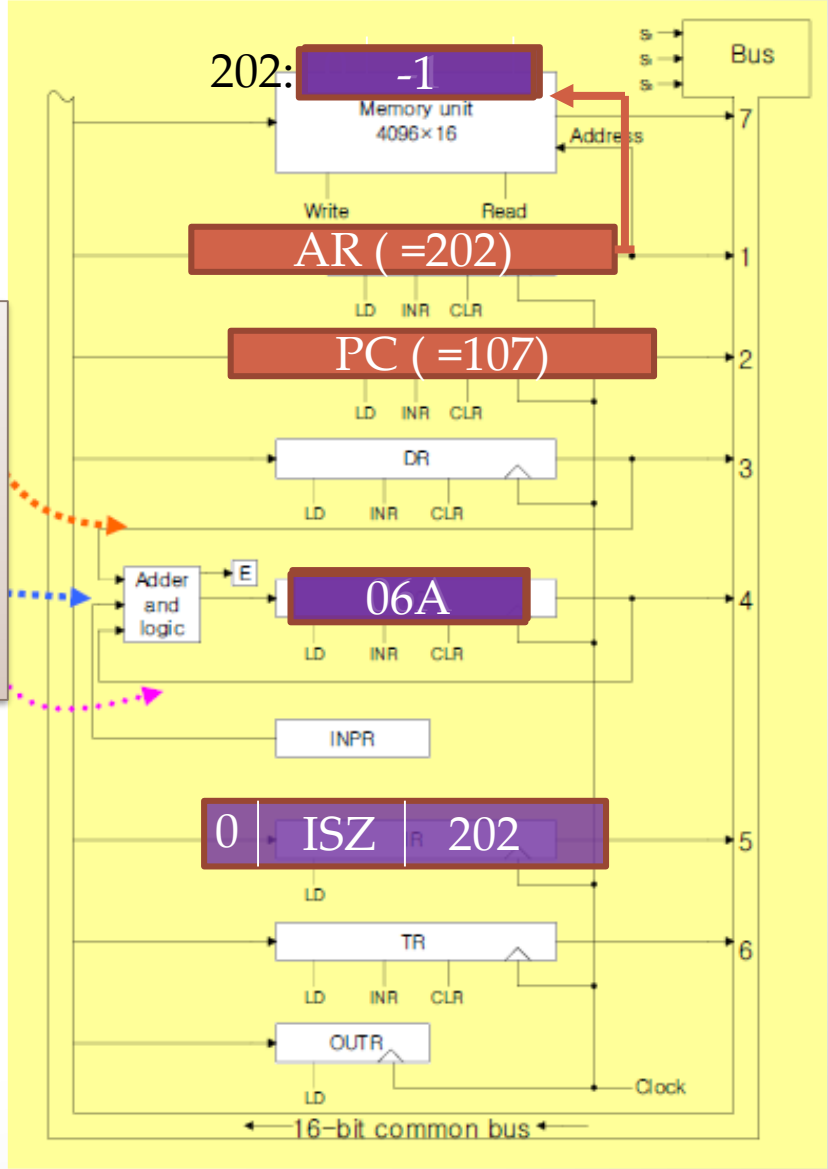
```
// set CTR to -100
LOP,    ...
        ...
        ISZ  CTR
        BUN  LOP
```

# MRI - ISZ

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0

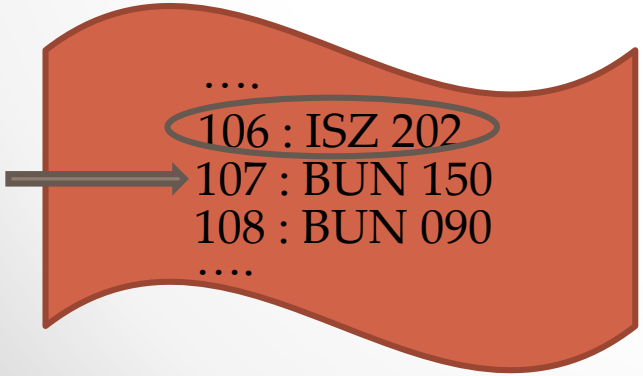


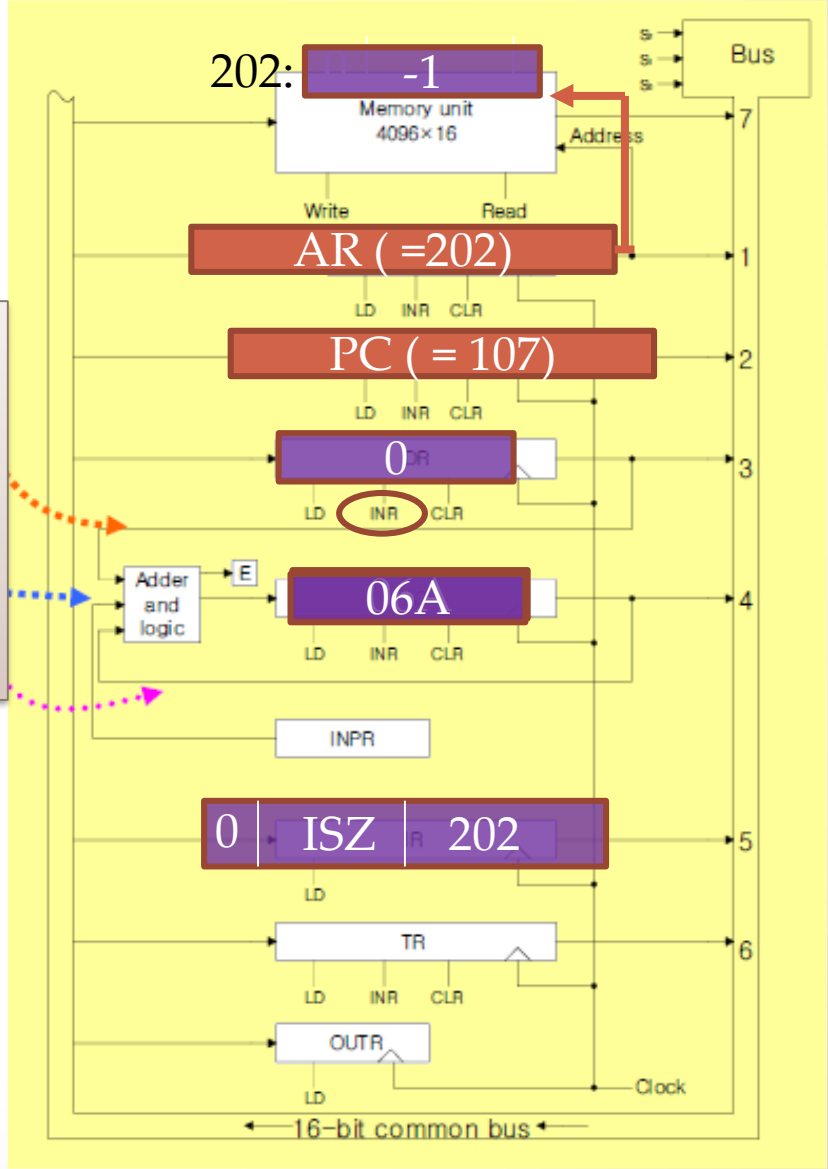



# MRI - ISZ

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0

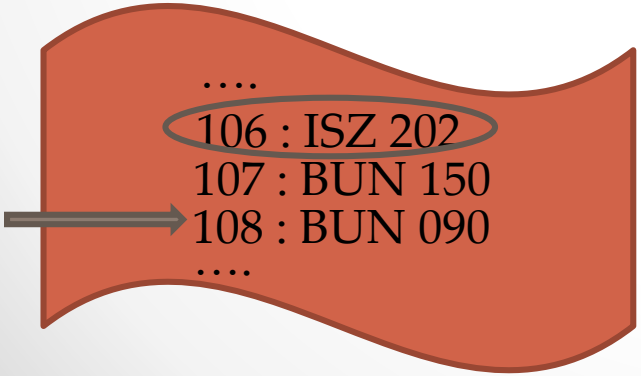


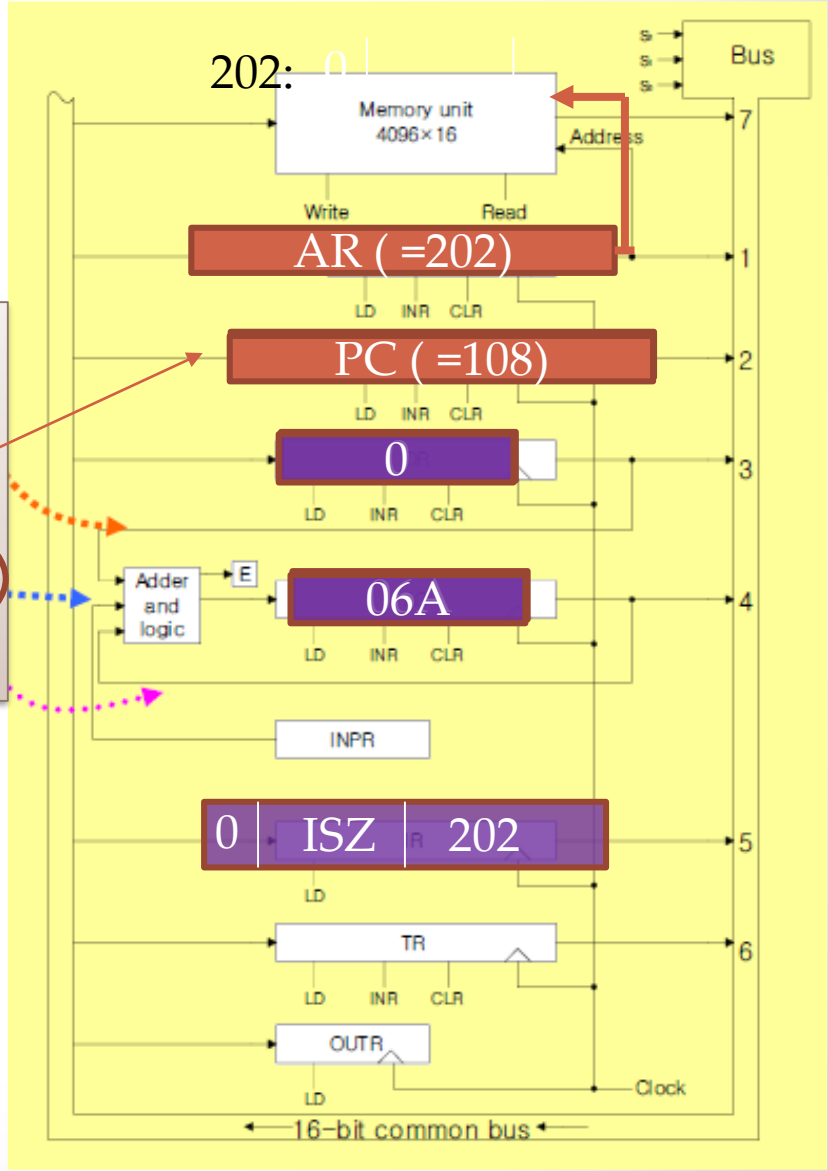



# MRI - ISZ

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0

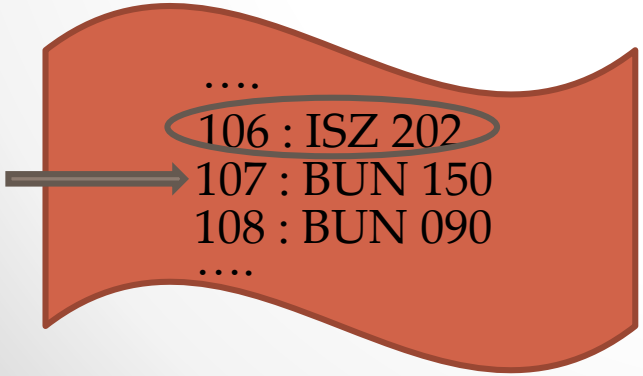


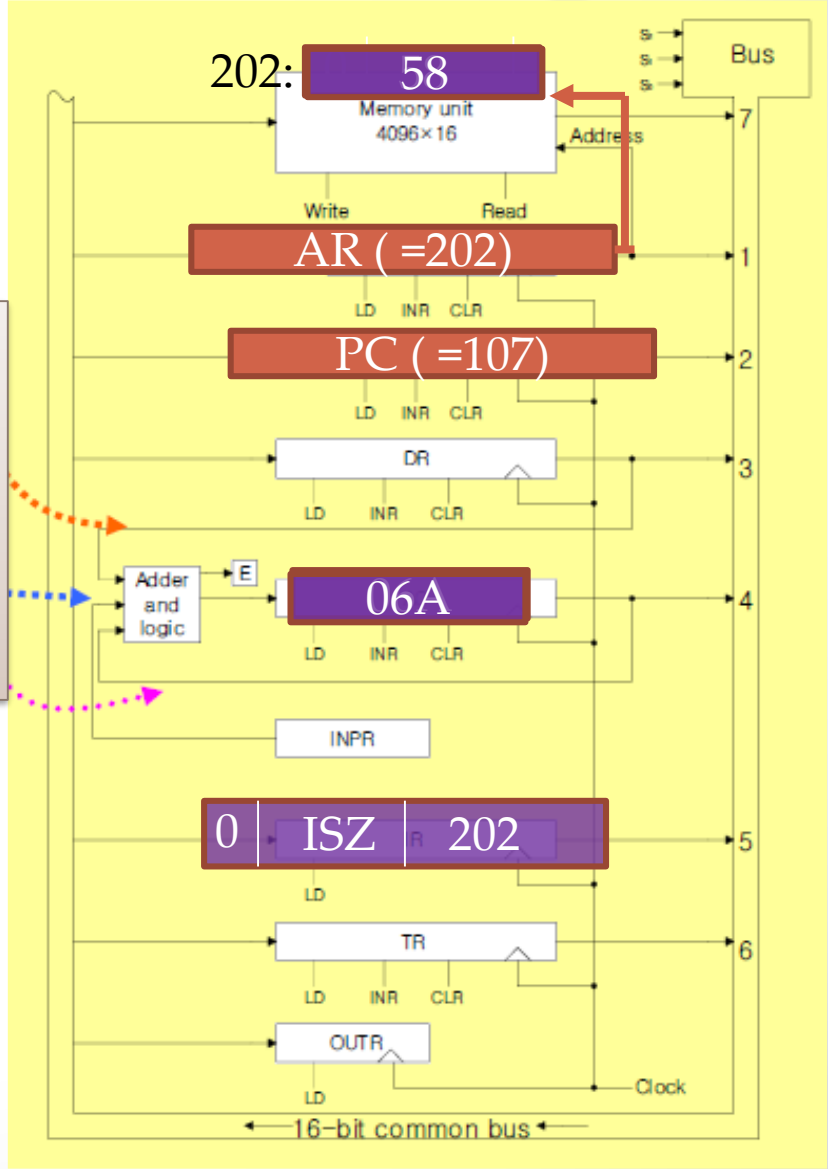



# MRI – ISZ (another example)

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0

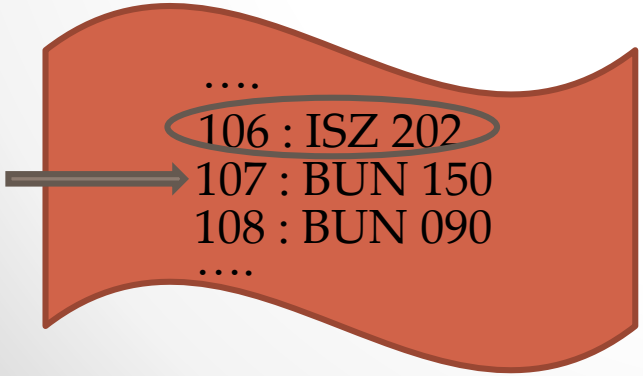


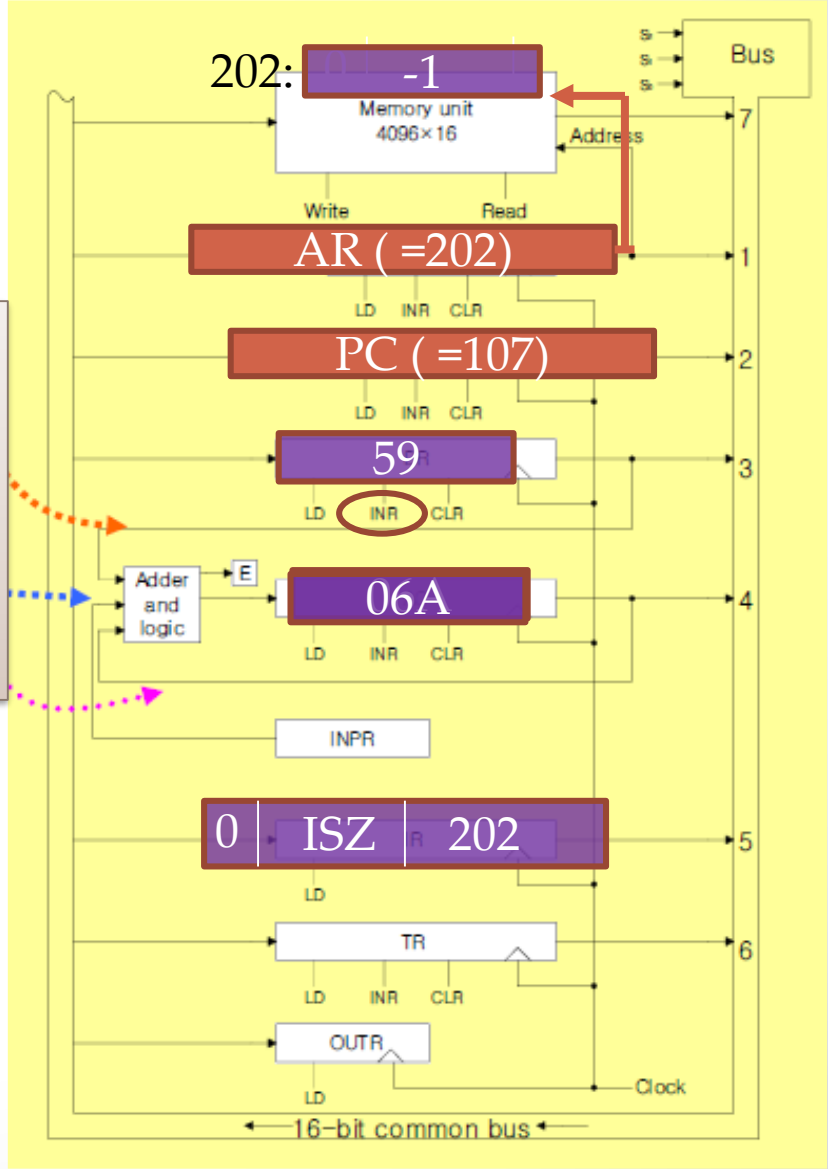



# MRI - ISZ

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0

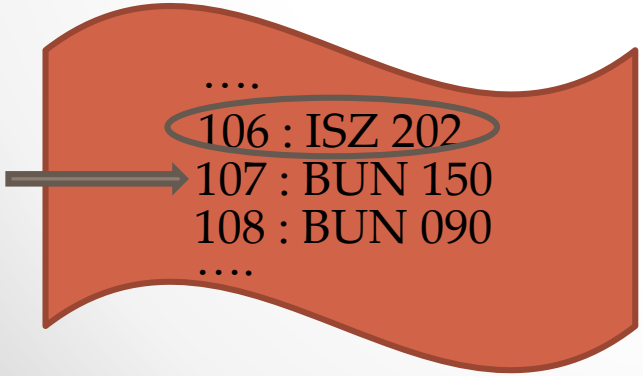


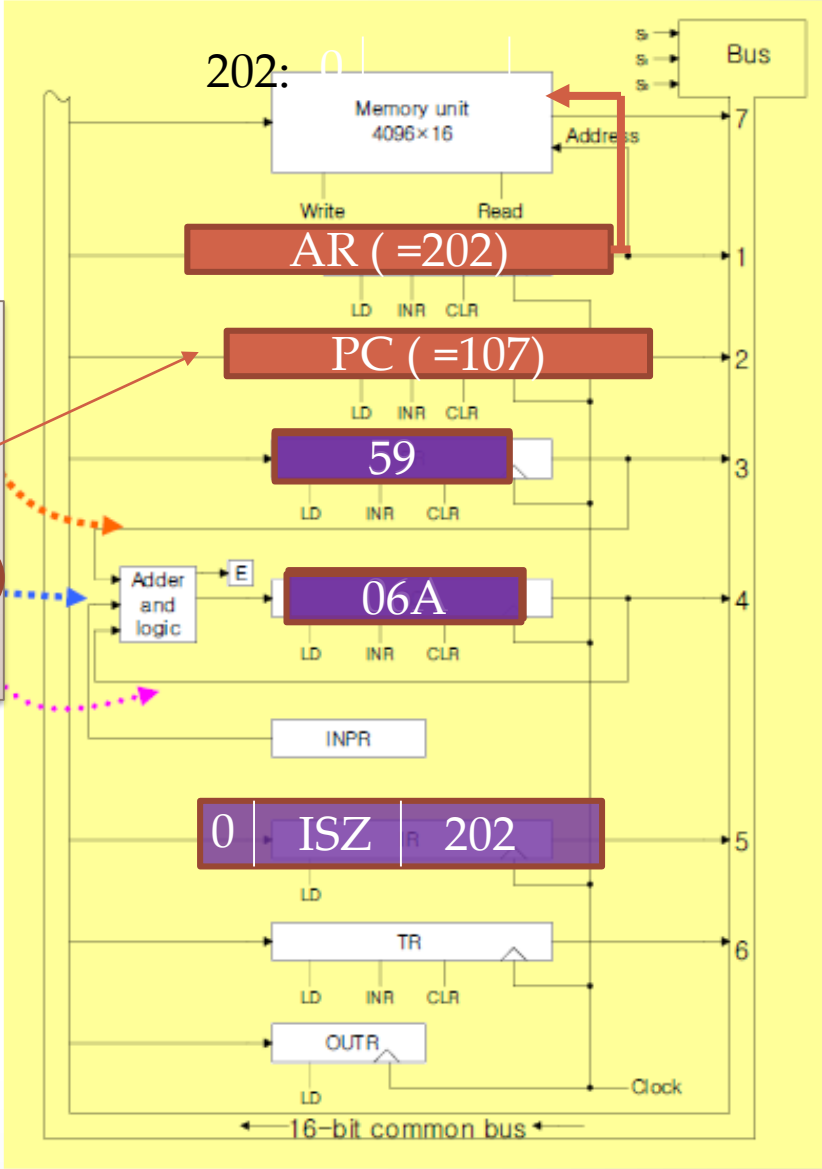



# MRI - ISZ

Increment memory word and skip next instruction if memory word equals to zero.

$D_6T_4$ : DR ← M[AR]
$D_6T_5$ : DR ← DR + 1
$D_6T_6$ : M[AR] ← DR,
if (DR = 0) then PC ← PC + 1,
SC ← 0



# Memory Reference Flowchart

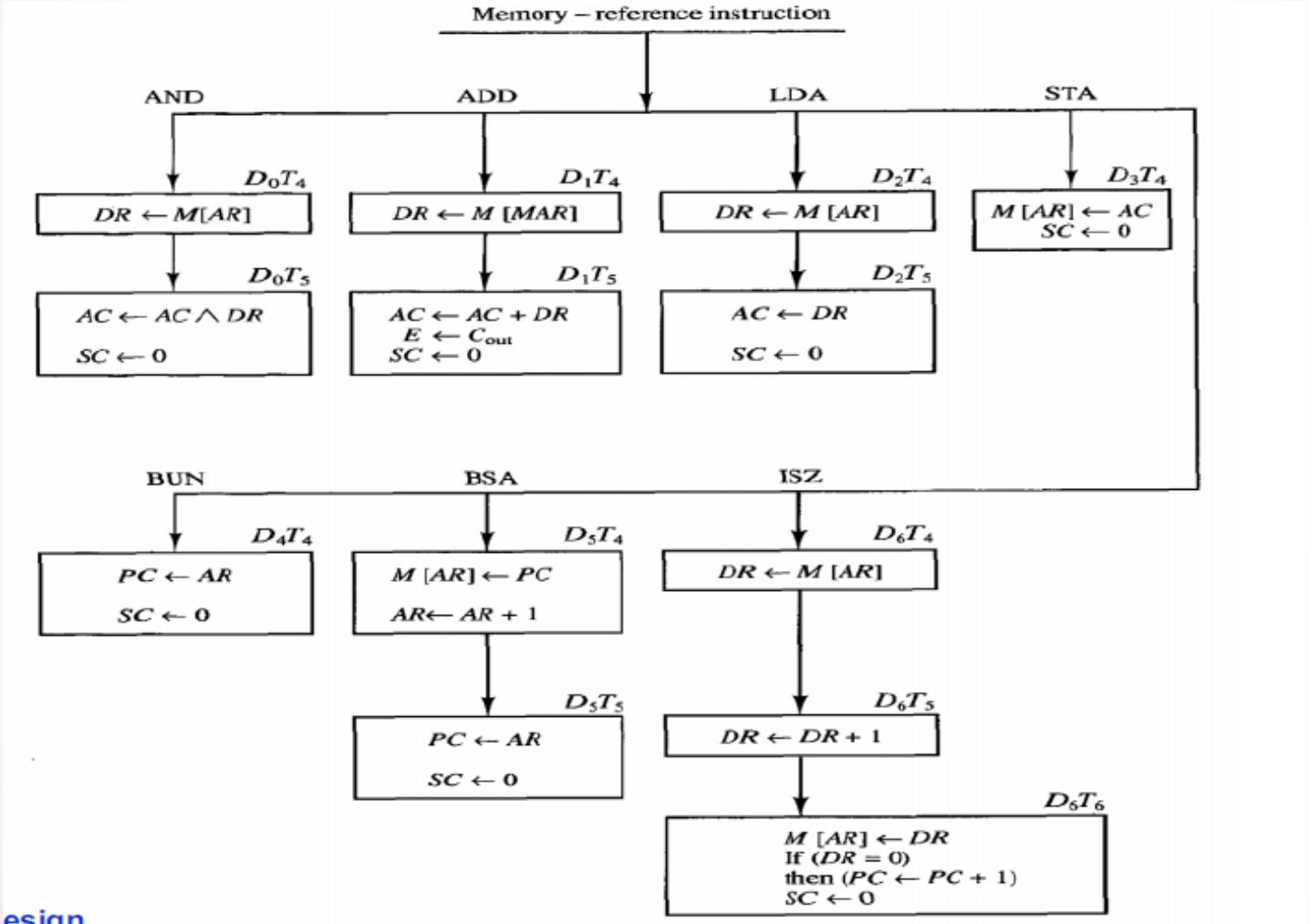



**Figure 5-11** Flowchart for memory-reference instructions.

esign
rd Ed.