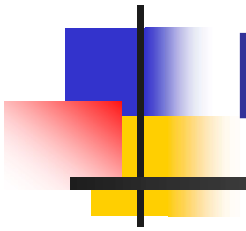
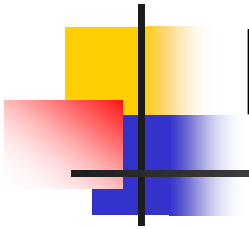
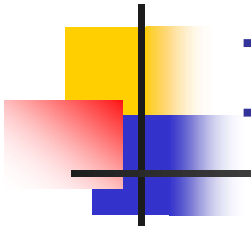


Unit 4
Basic Computer:
Hardwired Control Unit



Purpose of This Chapter

- In this chapter we introduce a basic computer and show how its operation can be specified with register transfer statements.

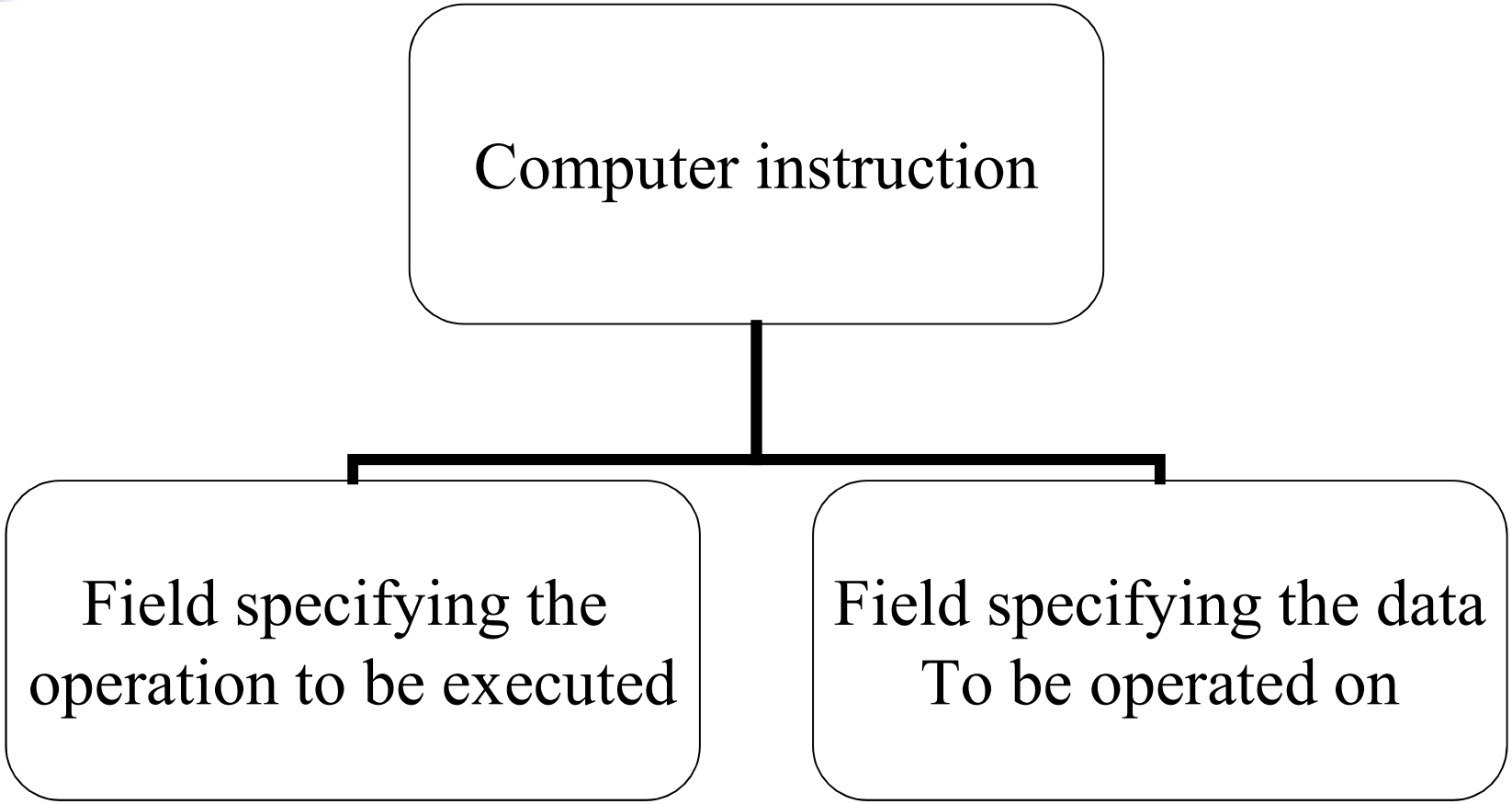


Instruction Codes

A process is controlled by a ***program***

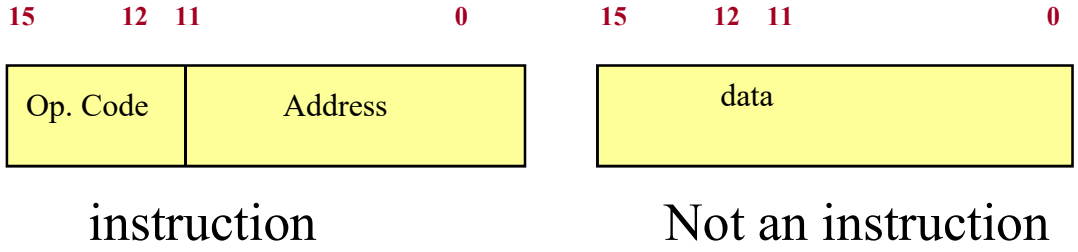
- A program is a set of ***instructions*** that specify the operations, data, and the control sequence
- An instruction is stored in binary code that specifies a sequence of microoperations
- Instruction codes together with data are stored in memory (Stored Program Concept)

Program statements and computer instructions



Instruction code format

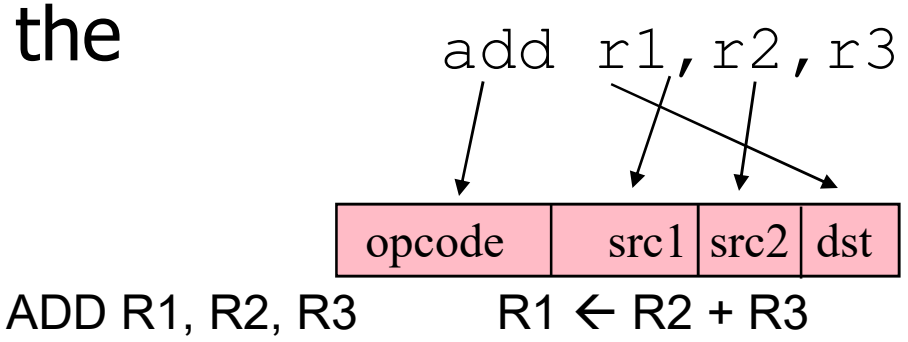
- Instruction code format with two parts : Op. Code + Address
  - Op. Code : specify 16 possible operations(4 bits)
  - Address : specify the address of an operand(12 bits)
  - If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction(***address field***) can be used for other purpose



Components of Instructions

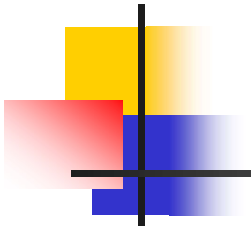
- Operations (opcodes)
- Number of operands (Number of data locations)

opcode:add value in src1 to value in src2 and place the result in dst.



- Instruction encodings

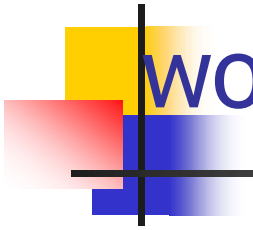
Number of Operands per instruction



No Operands	HALT NOP	
1 operand	NOT R4	R4 ← R4
2 operands	ADD R1, R2	R1 ← R1 + R2
3 operands	ADD R1, R2, R3	R1 ← R2 + R3
> 3 operands	MADD R4,R1,R2,R3	R4 ← R1+(R2*R3)

- Each specify one operation and 1,2, 3 or 4 data locations.

Instructions are read from memory as words



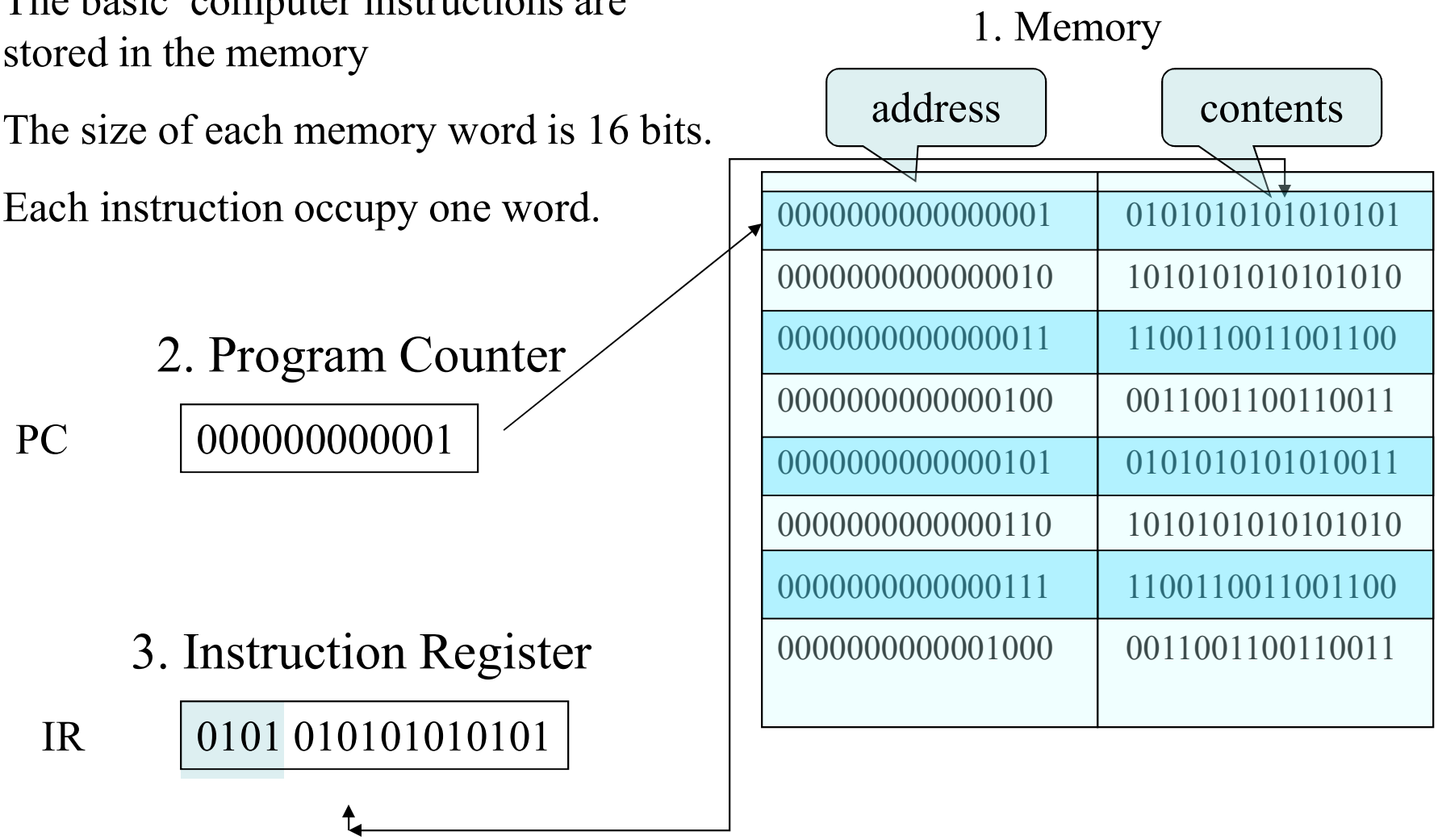
- Instructions can be formatted to fit in one or more memory words.
- An instruction may contain
  - An opcode + data (immediate operand)
  - An opcode + the address of data (direct addressing)
  - An opcode + an address where the address of the data is found (indirect addressing)
  - Data only (location has no instructions)
  - An opcode only (register-reference or input/output instruction)

Building A Basic Computer!

The basic computer instructions are stored in the memory

The size of each memory word is 16 bits.

Each instruction occupy one word.



The address register is connected to the memory

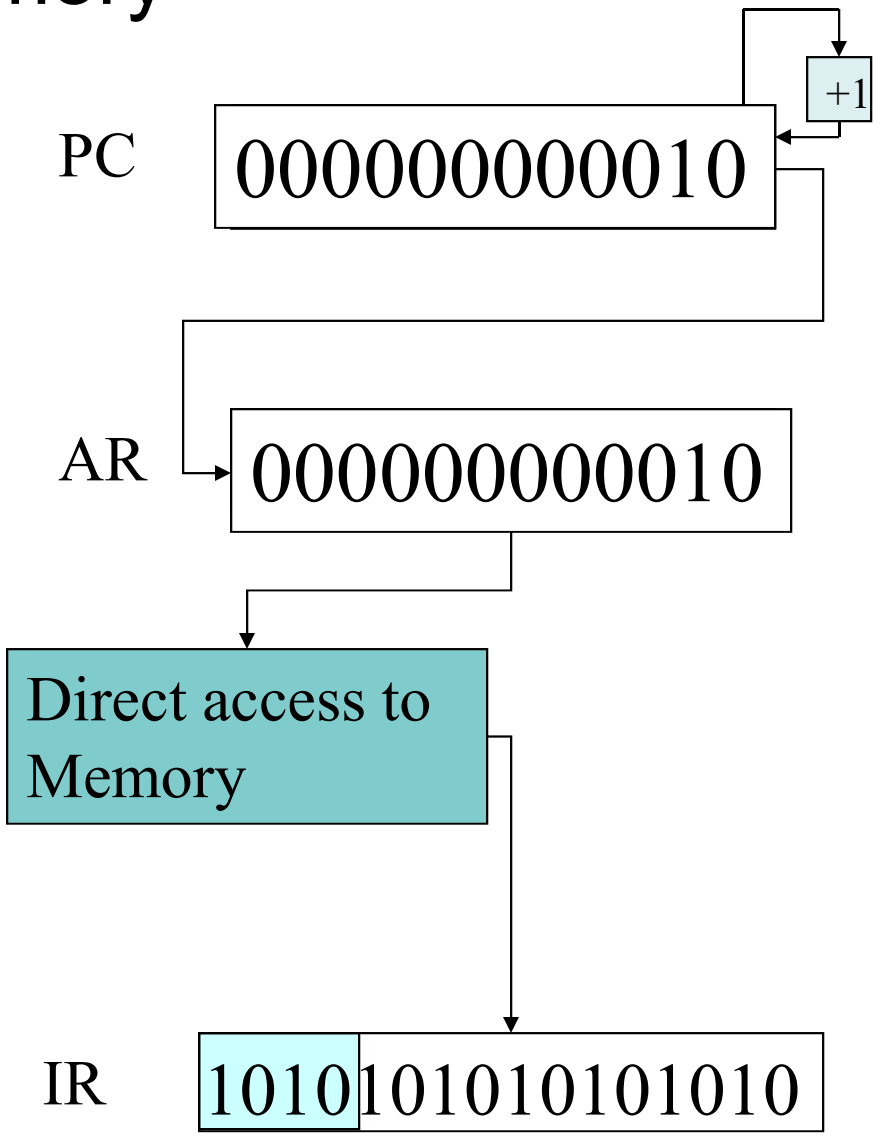
The Program Counter points to the next address of the program

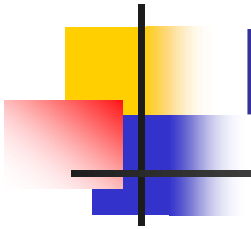
1. Program Counter Increments

by units of addresses

2. The next address is put on the bus and is loaded into the Address Register

3. The Bits of the AR are wired directly to the RAM Address lines to enable loading the memory into the Instruction R.





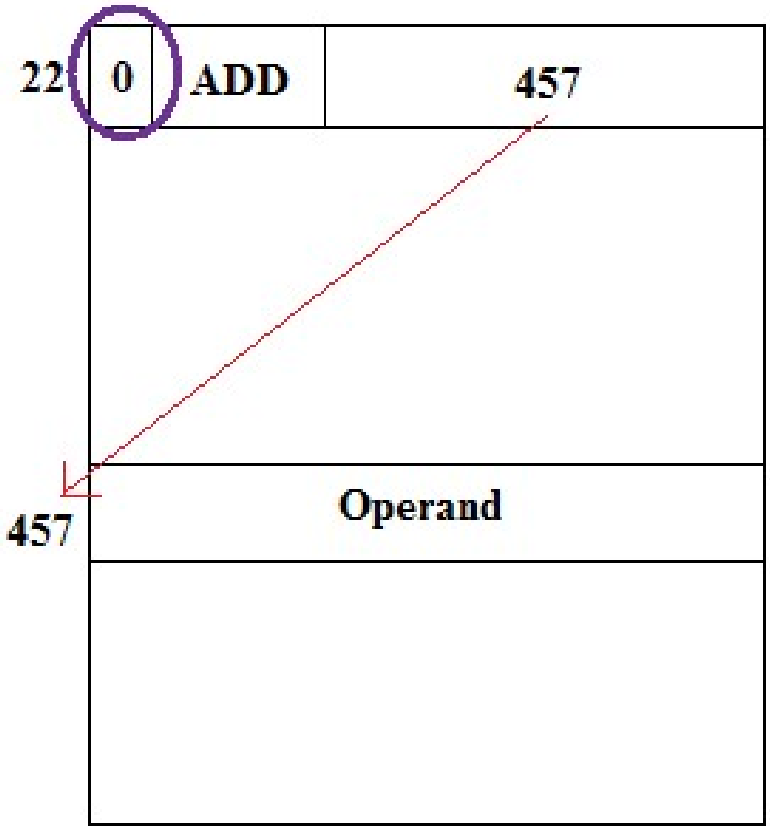
Direct address

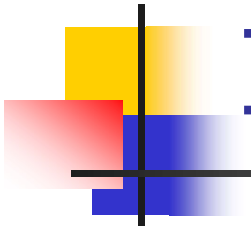
Occurs When the Operand Part Contains the Address of Needed Data.

1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and its Data placed on the bus to be loaded into the Data Register to be used for requested instructions

Direct address





Indirect address

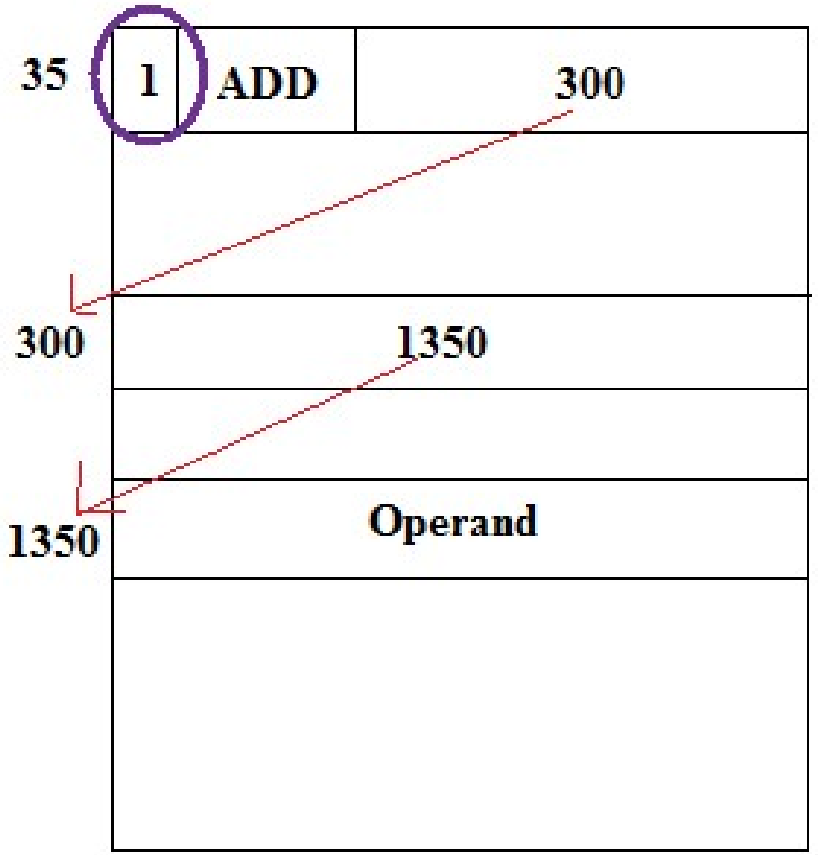
Occurs When the Operand Contains the Address of the Address of Needed Data.

1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and placed on the bus to be loaded Back into the AR

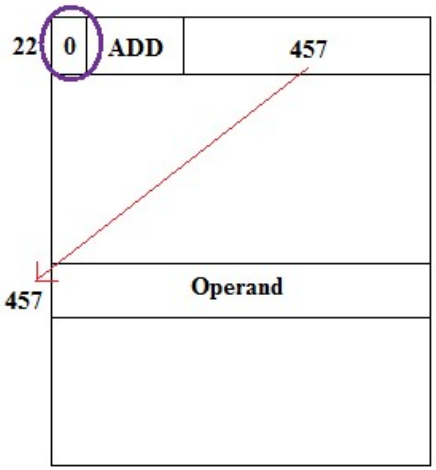
3. New Address is selected in memory and placed on the bus to be loaded into the DR to use later

Indirect address

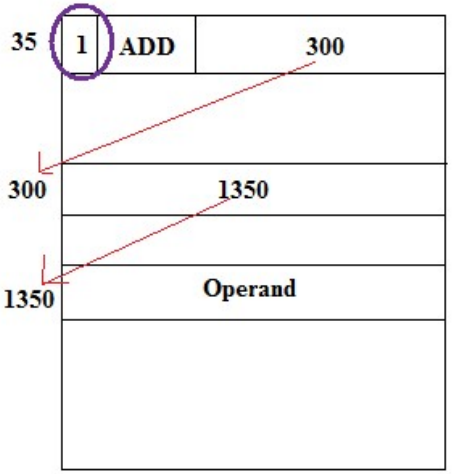


Effective address:

- **Effective address**: Address where an operand is physically located

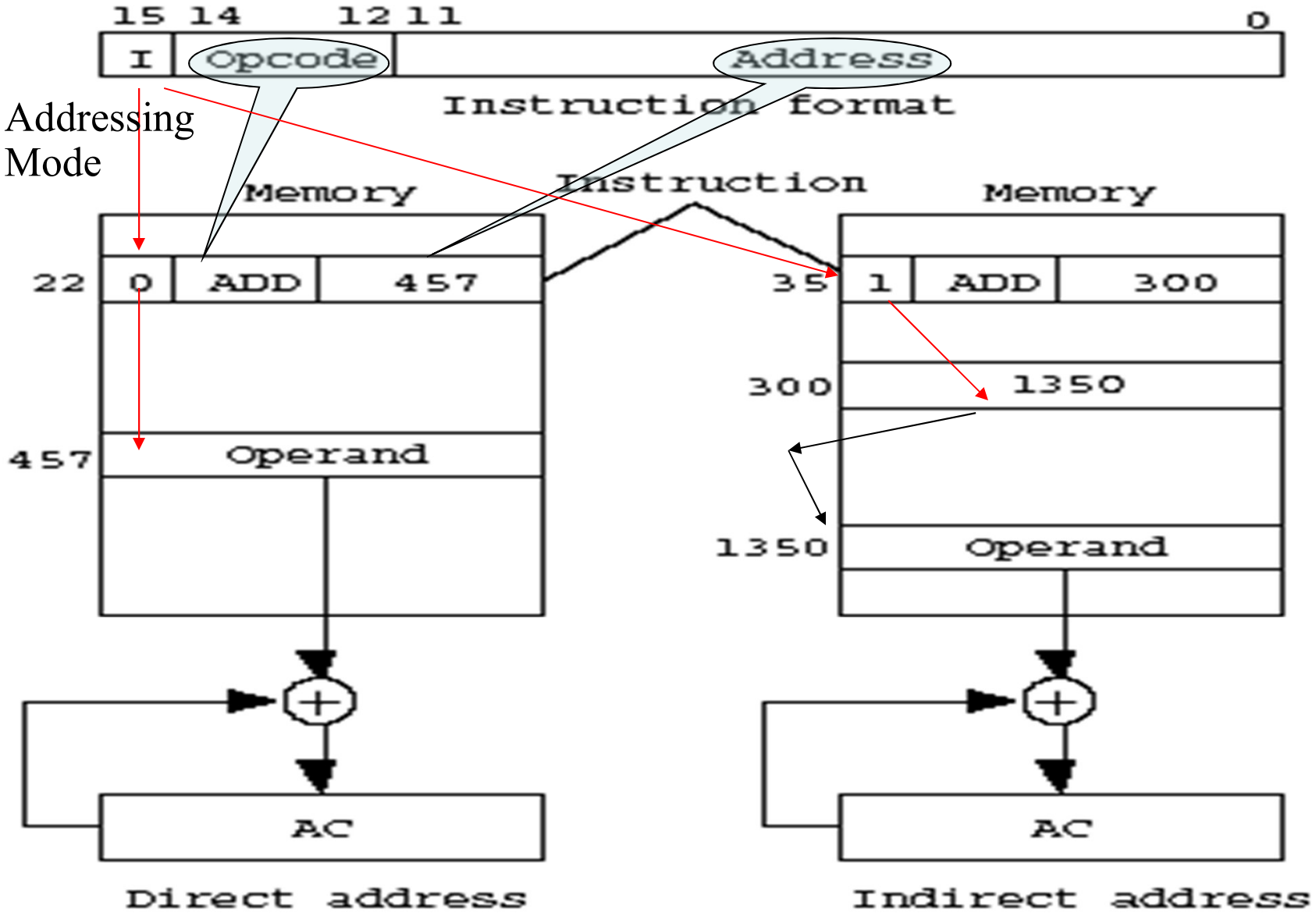


Effective address: 457

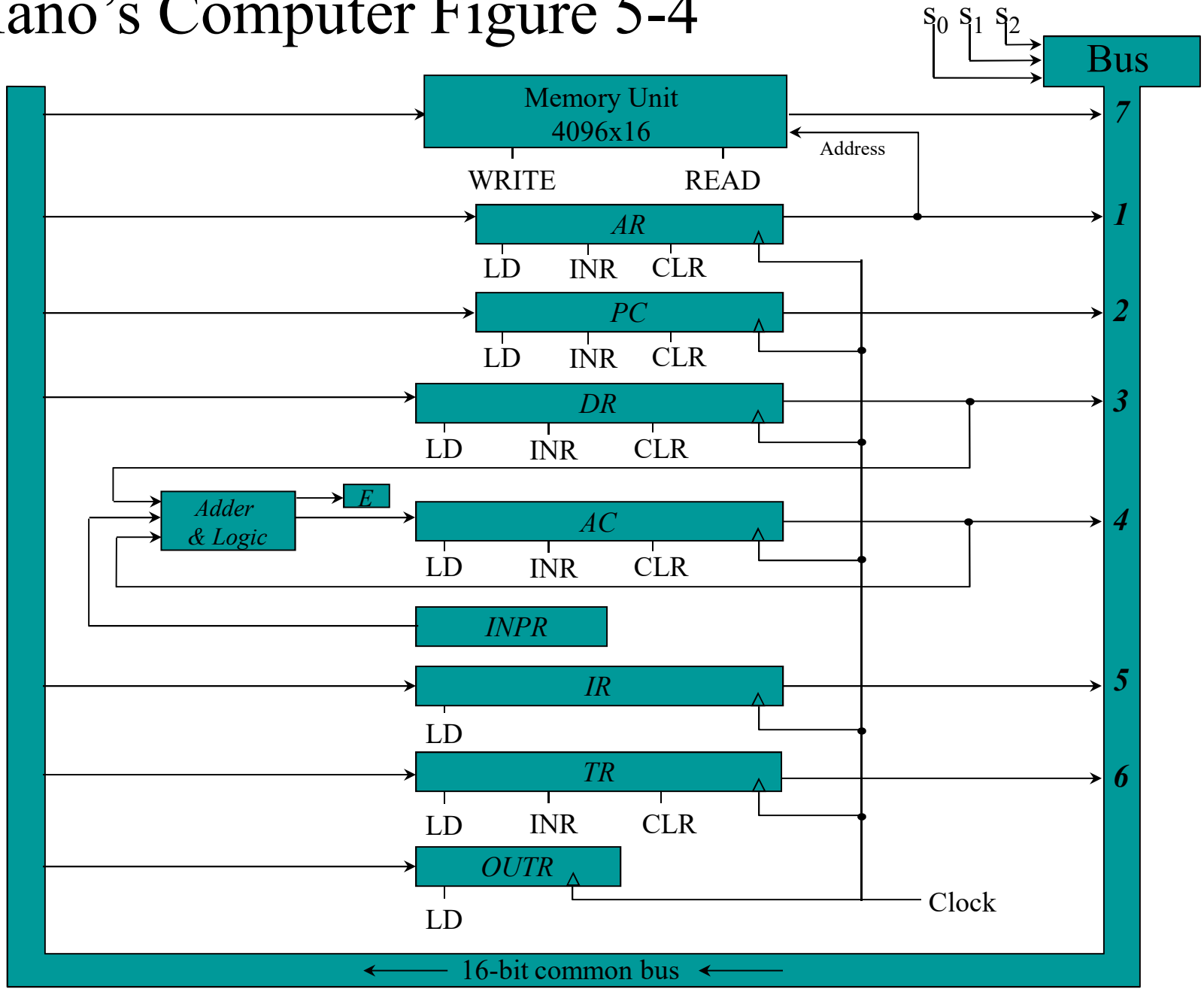


Effective address: 1350

Direct and Indirect addressing example



Mano's Computer Figure 5-4



Computer Registers

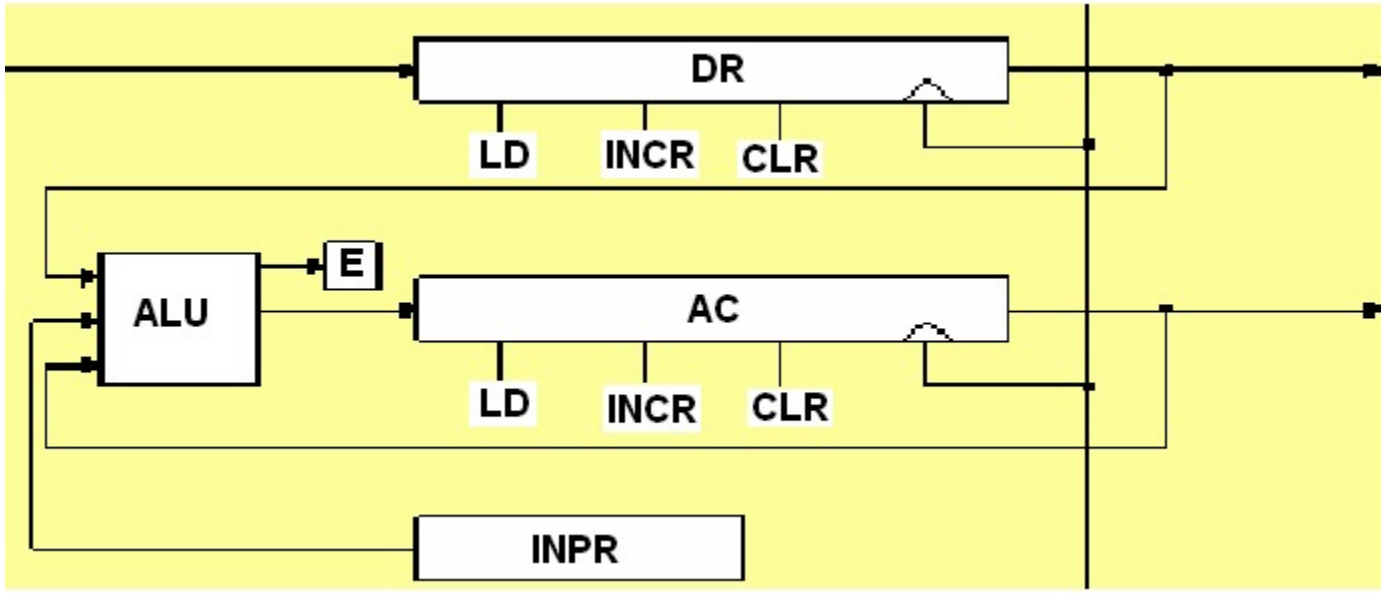
–Accumulator(AC) : takes input from ALU

»The ALU takes input from DR, AC and INPR :

»**ADD** DR **to** AC, **AND** DR **to** AC

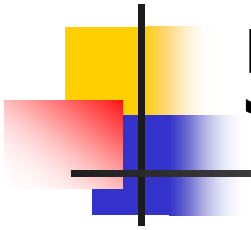
–Note) Input register is not connected to the bus.

–The input register is connected only to the ALU



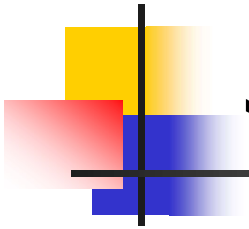
5-2 Computer Registers

- Data Register(***DR***) : hold the operand(Data) read from memory
- Accumulator Register(***AC***) : general purpose processing register
- Instruction Register(***IR***) : hold the instruction read from memory
- Temporary Register(***TR***) : hold a temporary data during processing
- Address Register(***AR***) : hold a memory address, 12 bit width



5-2 Computer Registers

- Program Counter(***PC***) :
  - hold the address of the next instruction to be read from memory after the current instruction is executed
  - Instruction words are read and executed in sequence unless a branch instruction is encountered
  - A branch instruction calls for a transfer to a nonconsecutive instruction in the program
  - The address part of a branch instruction is transferred to PC to become the address of the next instruction
  - To read instruction, memory read cycle is initiated, and PC is incremented by one(next instruction fetch)



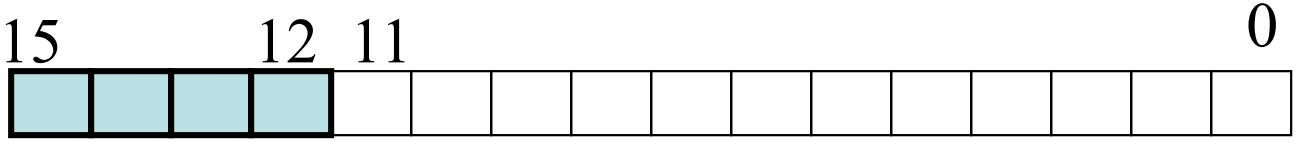
5-2 Computer Registers

- Input Register(***INPR***) : receive an 8-bit character from an input device
- Output Register(***OUTR***) : hold an 8-bit character for an output device

5-2 Computer Registers

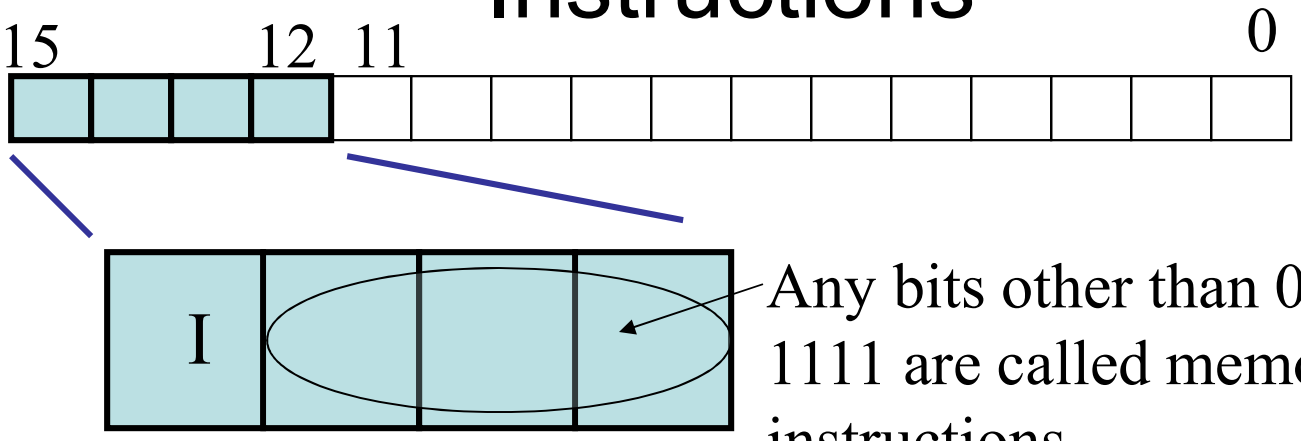
Register symbol	Number of bits	Register name	Register Function
DR	16	Data register	Holds memory operands
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Mano's Computer: each instruction occupies one Memory Words



- 4-bit opcode Bits 15-12
- How many possible instructions?
  - 2^4=16
- This leaves 12 bits for the address
  - How many words of memory?
  - $2^{12} = 2^2 \bullet 2^{10}$ = 4K = 4096 16-bit words

Mano's simple Computer: Instructions



000 AND

001 ADD

010 LDA

(Load Accumulator)

011 STA

(Store Accumulator)

100 BUN

(Branch Unconditional)

101 BSA

(Branch and Store Address)

110ISZ

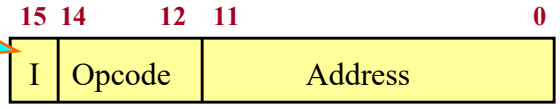
(Increment and Skip if Zero)

Symbol	Hex Code I = 0	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
CMS	7200		Complement AC
CME	m 7100	e	Comp
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

5-3. Computer Instruction

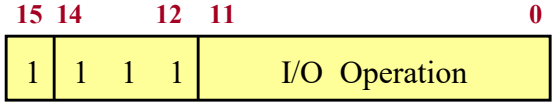
- 3 Instruction Code Formats : ***Fig. 5-5***
  - **Memory-reference instruction**
    - –Opcode = 000 ~ 110
      - »I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect

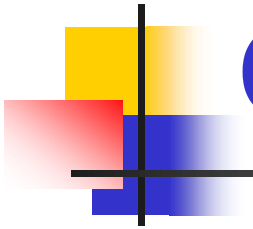


      - »**Register-reference instruction**
    - –7xxx (7800 ~ 7001) : CLA, CMA,

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



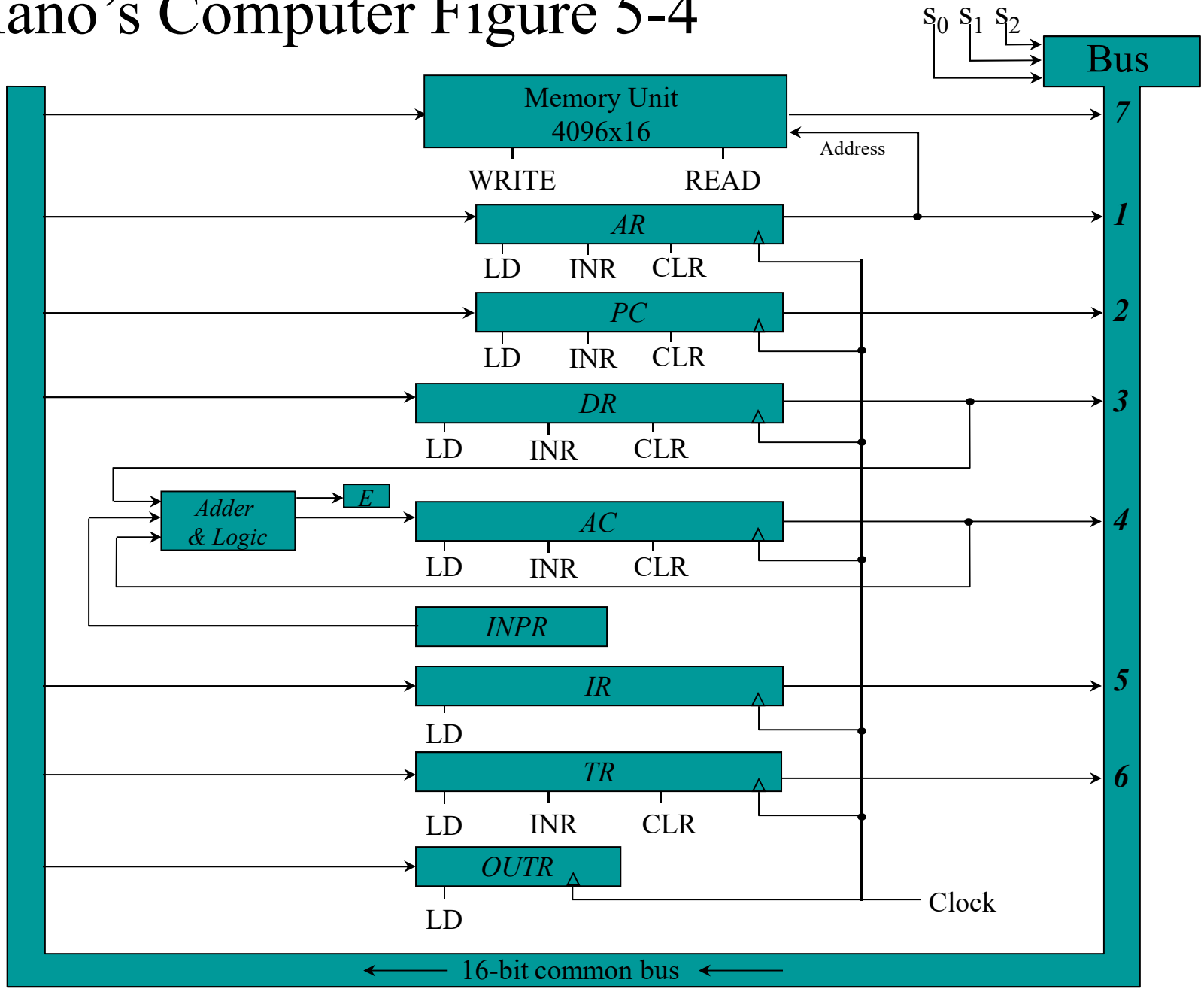
Symbol	Hex Code I = 0	I = 1	Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and Save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMS	7200		Complement AC
CME	m 7100	e	Comp
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



Common Bus System

- The basic computer has eight registers, a memory unit, and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and registers
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.

Mano's Computer Figure 5-4



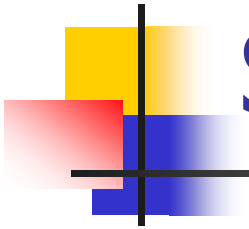
Common Bus System

- The connection of the registers and memory of the basic computer to a common bus system :
  - The outputs of seven registers and memory are connected to the common bus
  - The specific output is selected by mux(S0, S1, S2) :
    - Memory(7), AR(1), PC(2), DR(3), AC(4), IR(5), TR(6)
    - When LD(Load Input) is enable, the particular register receives the data from the bus
  - Control Input : LD, INC, CLR, Write, Read

COMMON BUS SYSTEM

- **Control variables:** the bus is controlled by

- 1- Selection switches for selecting the source of information and
- 2- Enable switches at the destination device to accept the information.



Selection variables

- **Selection variables**: select a register or the memory whose output is used as an input to the common bus.

- To select one device out of 8, we need 3 select variables.
- For example, if S2S1S0 = 011, the output of DR is selected as an output of the common bus.

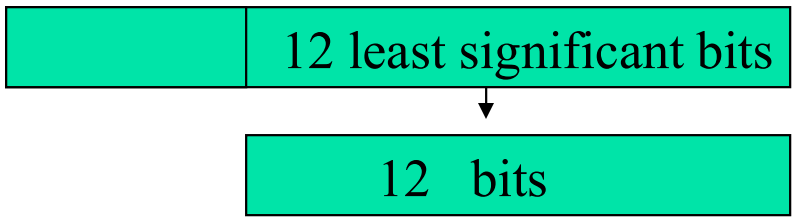
Load input

Load input (LD): Enables the input of a register to download bits form the common bus. When LD = 1 for a register, the data on the common bus is read into the register during the next clock pulse transition.

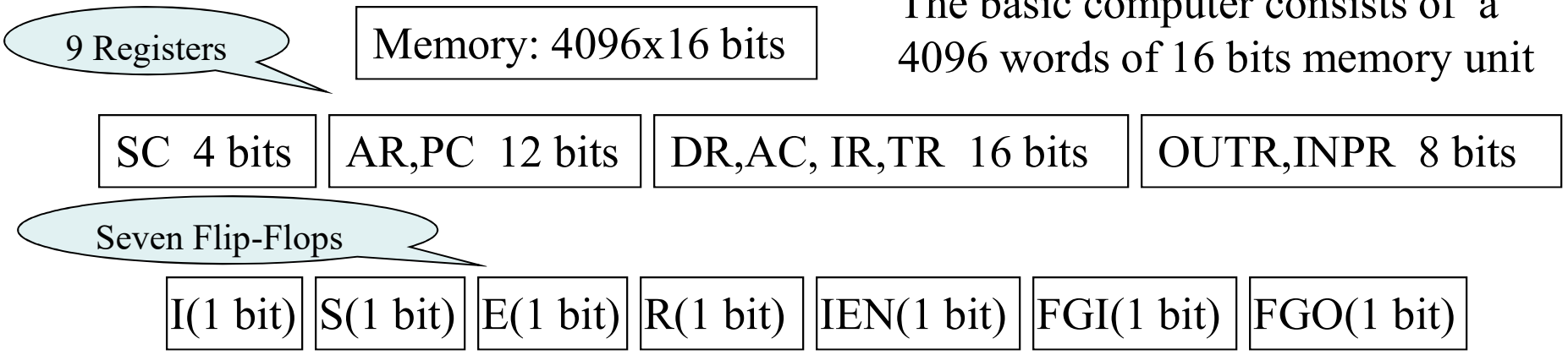
> **Increment input (INR)**: Increments the content of a register.
> **Clear input (CLR)**: Clear the content of a register to zero.

Incompatibility in register sizes

- • When the contents of AR or PC (12 bits) are applied to the 16-bit common bus, the four most significant bits are set to zero. When AR or PC receives information from the bus, only the 12 least significant bits are transferred to the register.



Mano's Example of Basic Computer (Section 5.9)



The basic computer consists of a 4096 words of 16 bits memory unit

Adder and Logic circuit connected to the AC input

Control Logic Gates:

Signals to control the
the nine registers' inputs
memory read and write
F/Fs set, clear, or
complement
S2 S1 S0 bus selection
the AC ,ALU circuit
Two decoders: 3 x 8(opcode)
and 4 x 16 timing decoder

ALU

16 bits

Control Unit

Logic gates

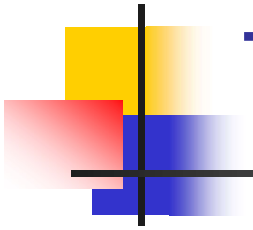
3x8 DEC, 4x16 DEC

BUS: 8x1 MUX

16 bits

Nine registers : AR,
PC(12bits each), DR, AC,
IR, TR(16 bits each),
OUTR, INPR(8 bit each),
and SC(4bits)
Seven F/Fs : I, S, E, R,
IEN, FGI, and FGO (1 bit
each)
A 16-bit common bus

IR and TR



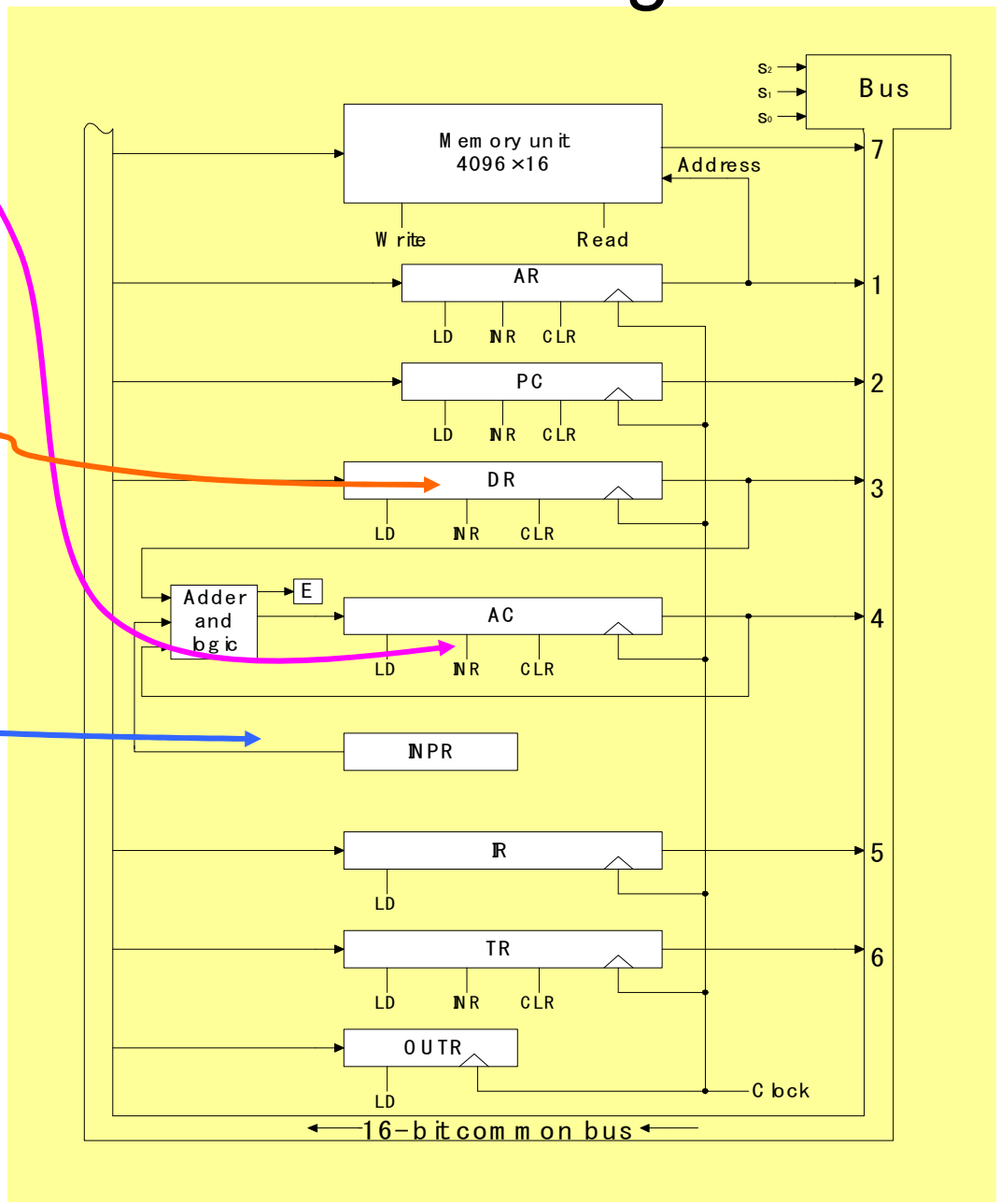
- The instruction register, IR, can only be loaded; it cannot be incremented nor cleared. Its output is used to generate Di's and Ti's control signals.
- TR is a temporary register. The CPU uses this register to store intermediate results of operations. It is not accessible by the external programs. It is loaded, incremented and cleared like the other registers.

Operations involve AC and DR Registers

Accumulator(AC) :
Main Register
Microoperation : clear AC, shift AC

Data Register : **ADD** DR **to** AC, **AND** DR **to** AC

3) INPR: Input device



Computer Instruction

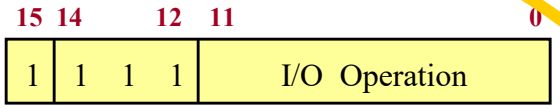
3 Instruction Code Formats :

1-Register-reference instruction

–7xxx (7800 ~ 7001) : CLA, CMA, ….

2-Input-Output instruction

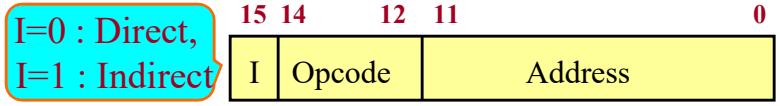
–Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ….



3-Memory-reference instruction

Opcode = 000 ~ 110

I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

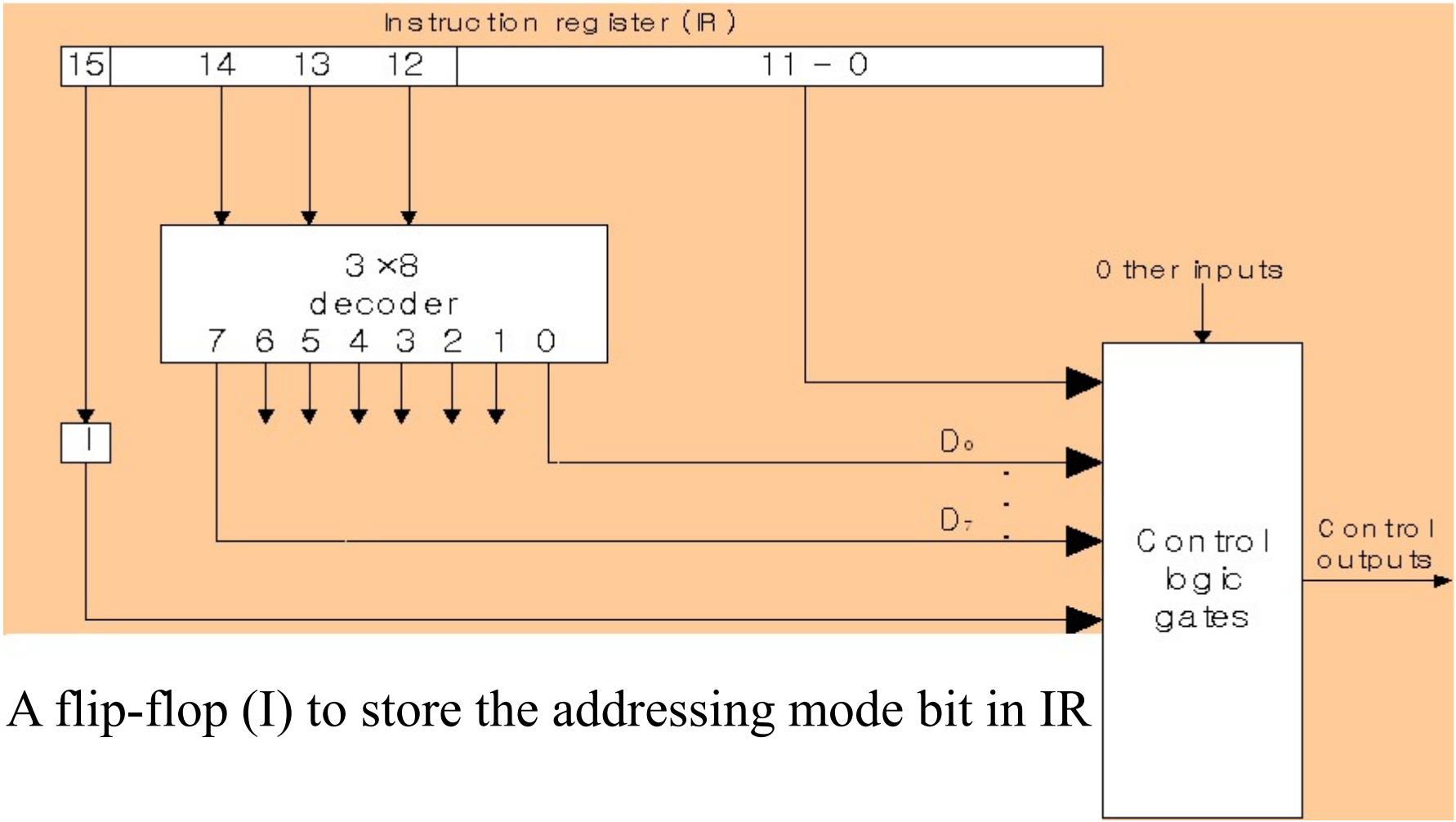


Symbol	Hexadecimal code I = 0	Hexadecimal code I = 1	Description
Register Reference			
CLA	0111 1000 0000 0000		Clear AC
CLE	0111 0100 0000 0000		Clear E
CMA	0111 0010 0000 0000		Complement AC
CME	0111 0001 0000 0000		Complement E
CIR	0111 0000 1000 0000		Circulate right AC and E
CIL	0111 0000 0100 0000		Circulate left AC and E
INC	0111 0000 0010 0000		Increment AC
SPA	0111 0000 0001 0000		Skip next instruction if AC positive
SNA	0111 0000 0000 1000		Skip next instruction if AC negative
SZA	0111 0000 0000 0100		Skip next instruction if AC zero
SZE	0111 0000 0000 0010		Skip next instruction if E is 0
HLT	0111 0000 0000 0001		Halt computer
			7800 (B_{11} bit)
Input-Output			
INP		1111 1000 0000 0000	Input character to AC
OUT		1111 0100 0000 0000	Output character from AC
SKI		1111 0010 0000 0000	Skip on input flag
SKO		1111 0001 0000 0000	Skip on output flag
ION		1111 0000 1000 0000	Interrupt on
IOF		1111 0000 0100 0000	Interrupt off
Memory Reference			
AND	0000 xxxx xxxx xxxx	1000 xxxx xxxx xxxx	AND memory word to AC
ADD	0001 xxxx xxxx xxxx	1001 xxxx xxxx xxxx	ADD memory word to AC
LDA	0010 xxxx xxxx xxxx	1010 xxxx xxxx xxxx	Load memory word to AC
STA	0011 xxxx xxxx xxxx	1011 xxxx xxxx xxxx	Store content of AC in memory
BUN	0100 xxxx xxxx xxxx	1100 xxxx xxxx xxxx	Branch unconditionally
BSA	0101 xxxx xxxx xxxx	1101 xxxx xxxx xxxx	Branch and save return address
ISZ	0110 xxxx xxxx xxxx	1110 xxxx xxxx xxxx	Increment and skip if zero

CONTROL UNIT HARDWARE (Hardwired)

- Inputs to the control unit come from IR where an instruction is stored.
- A hardwired control is implemented in the example computer using:

> A 3x8 decoder to decode opcode bits 12-14 into signals D0, ..., D7;



A flip-flop (I) to store the addressing mode bit in IR

A 4-bit binary **sequence counter** (SC) to count from 0 to 15 to achieve time sequencing;
> A 4x16 decoder to decode the output of the counter into 16 timing signals, T0, ..., T15

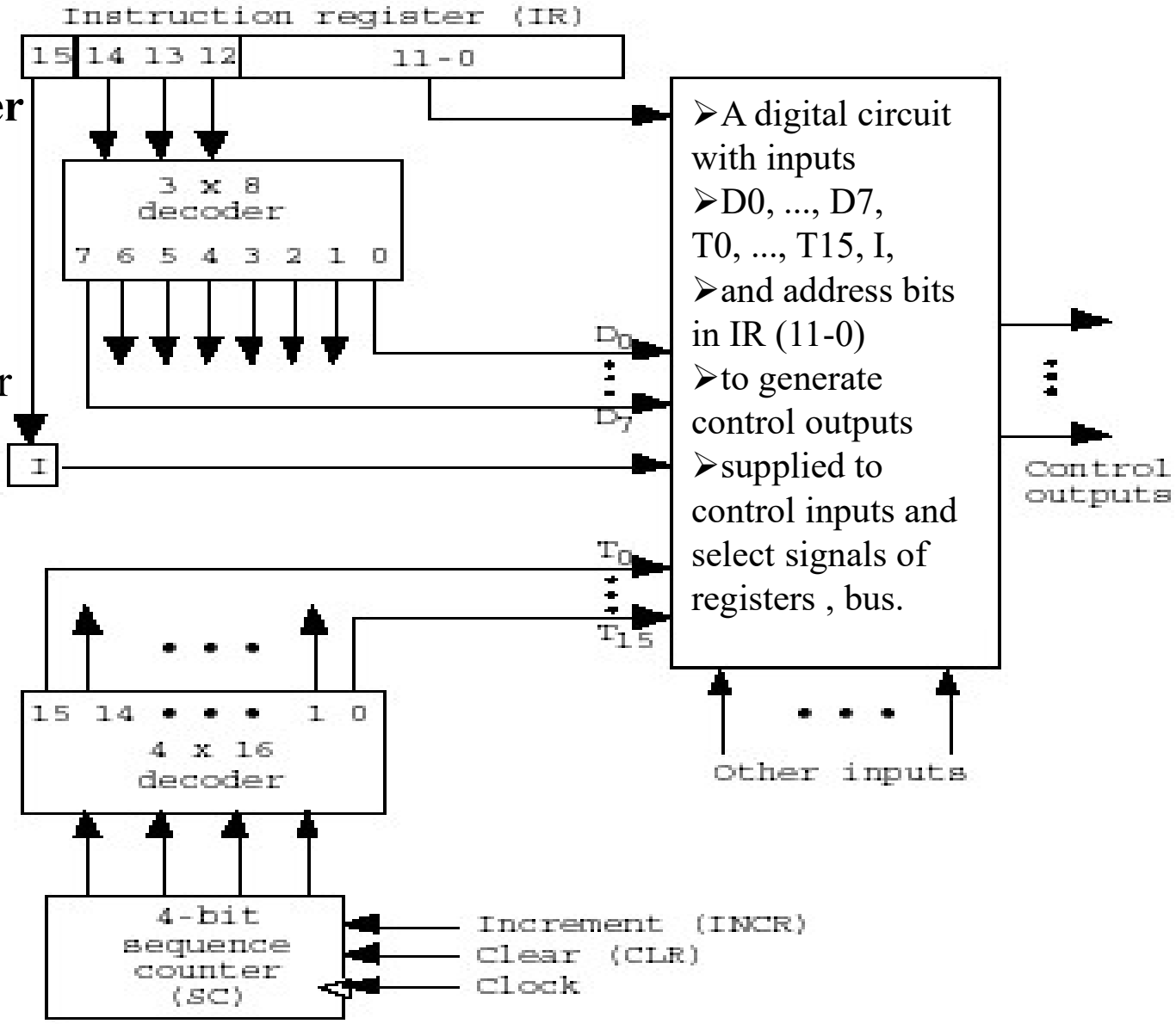
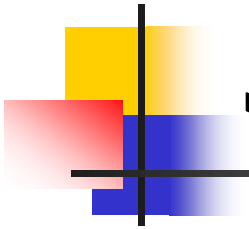


Figure: Control unit hardware.



5.5 Instruction Cycle

- A computer goes through the following instruction cycle repeatedly:

do

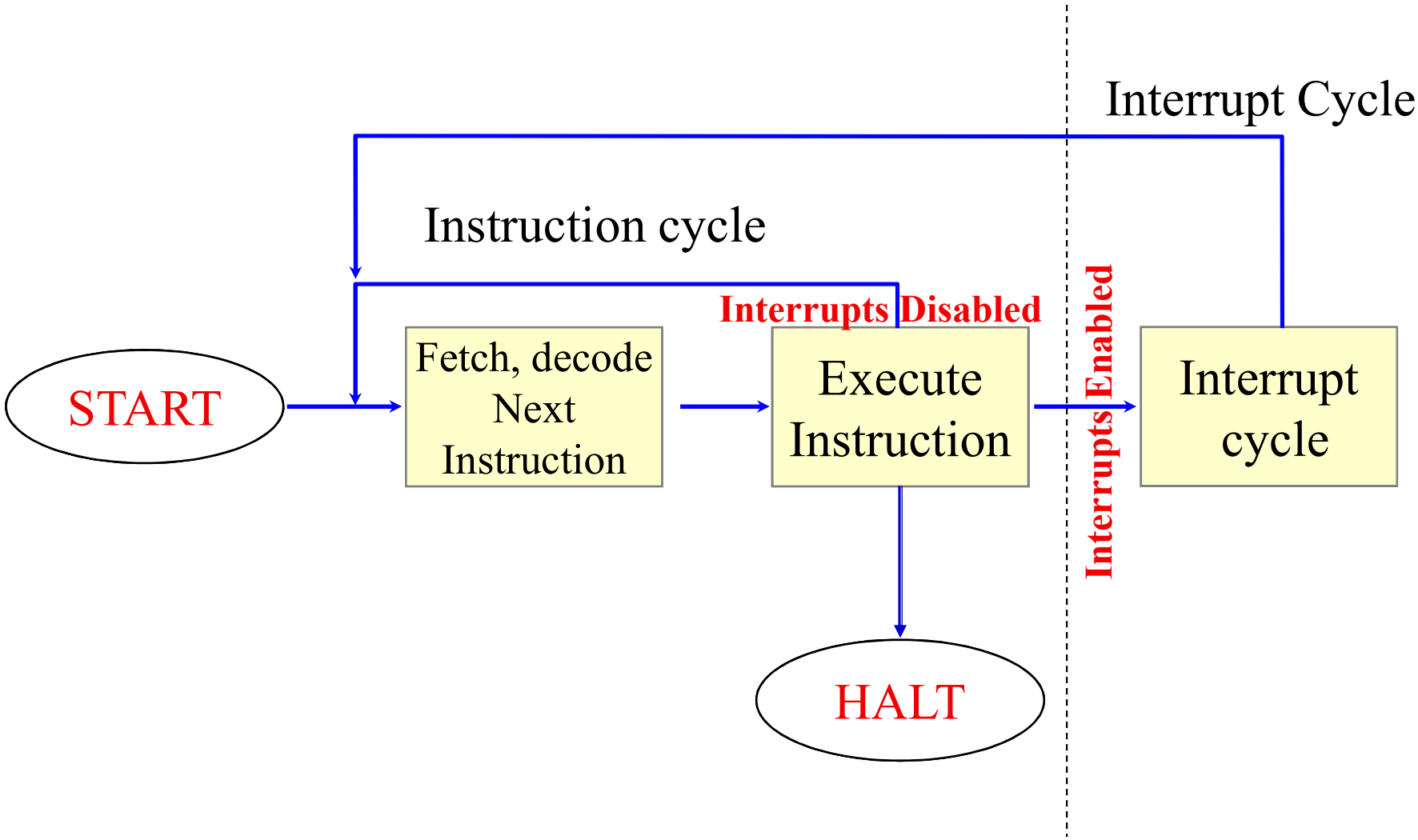
1. Fetch an instruction from memory

2. Decode the instruction

3. Read the effective address from memory if the instruction has an indirect address

4. Execute the instruction until a HALT instruction is encountered

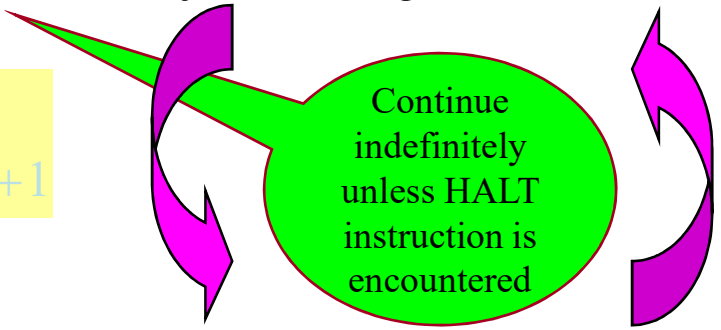
Instruction and Interrupt cycles



Instruction Fetch

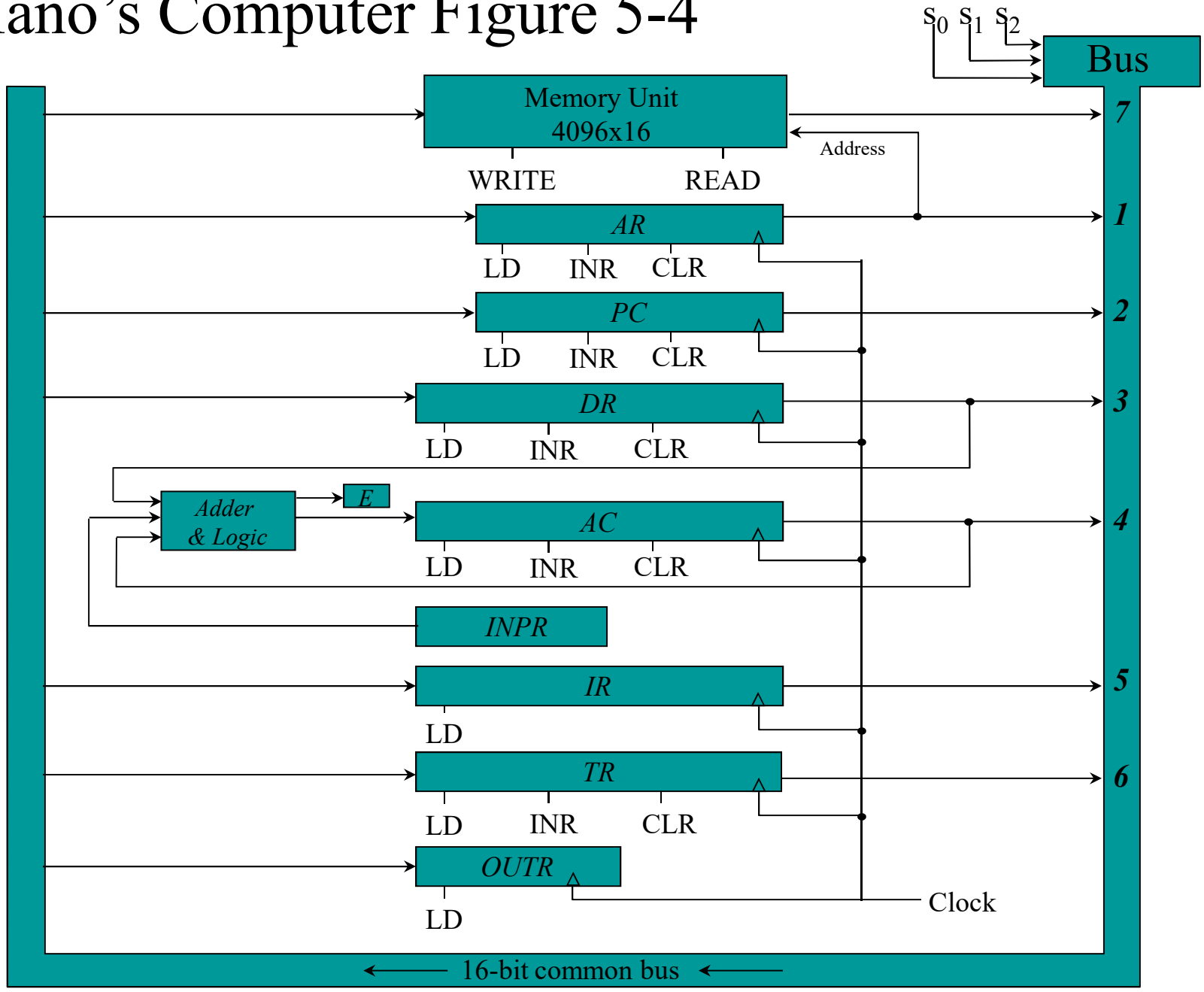
- Instruction Fetch : T0, T1
  - T0 = 1
    - 1) Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$=010
    - 2) Transfer the content of the bus to AR by enabling the LD input of AR

$$T_0 : AR \leftarrow PC$$
$$T_1 : IR \leftarrow M[AR],\ PC \leftarrow PC + 1$$



- T1 = 1
  - 1) Enable the read input memory
  - 2) Place the content of memory onto the bus by making $S_2S_1S_0$= 111
  - 3) Transfer the content of the bus to IR by enable the LD input of IR
  - 4) Increment PC by enabling the INR input of PC

Mano's Computer Figure 5-4



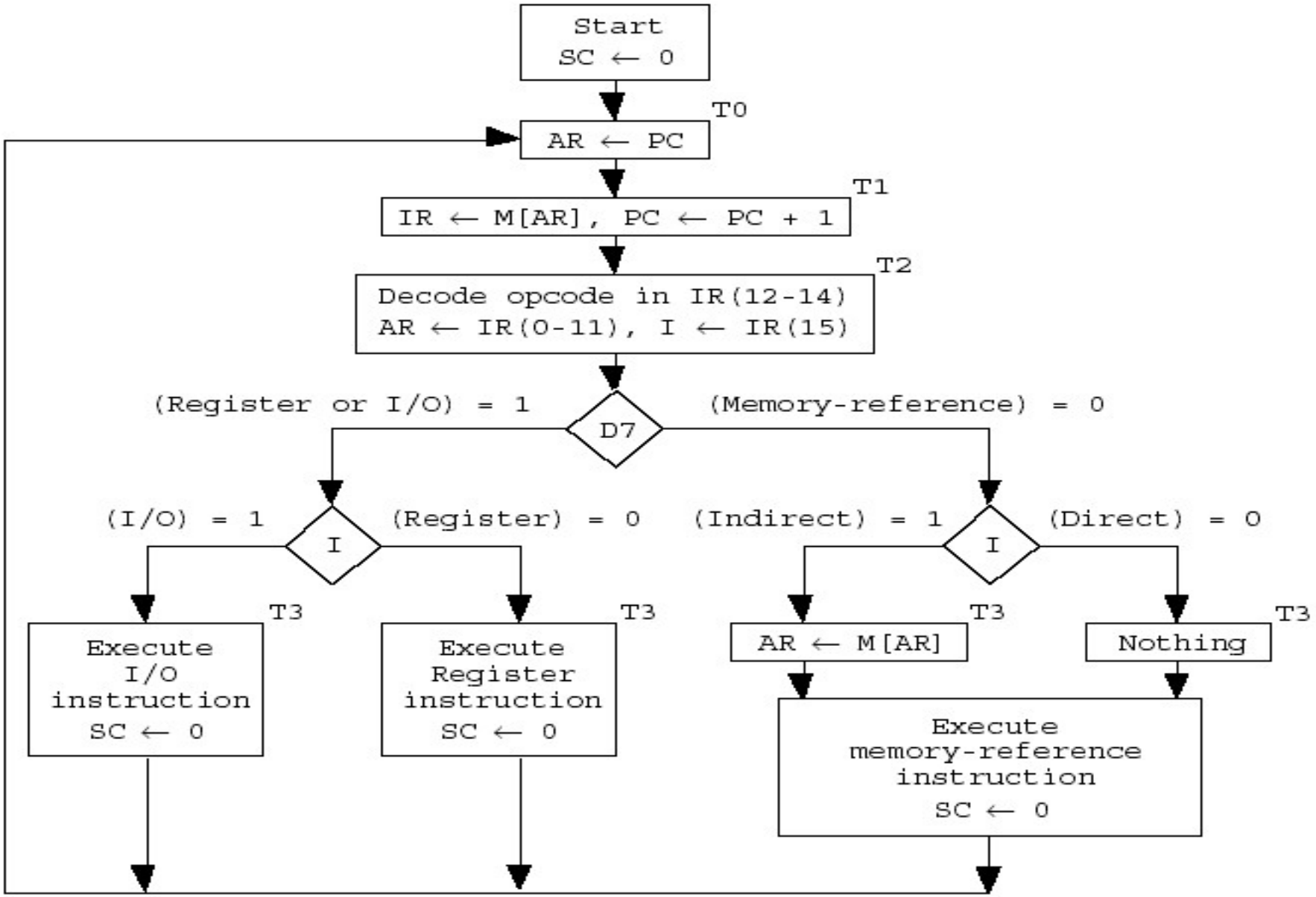


Figure: Flowchart for fetch & decode phases.

Instruction Cycle

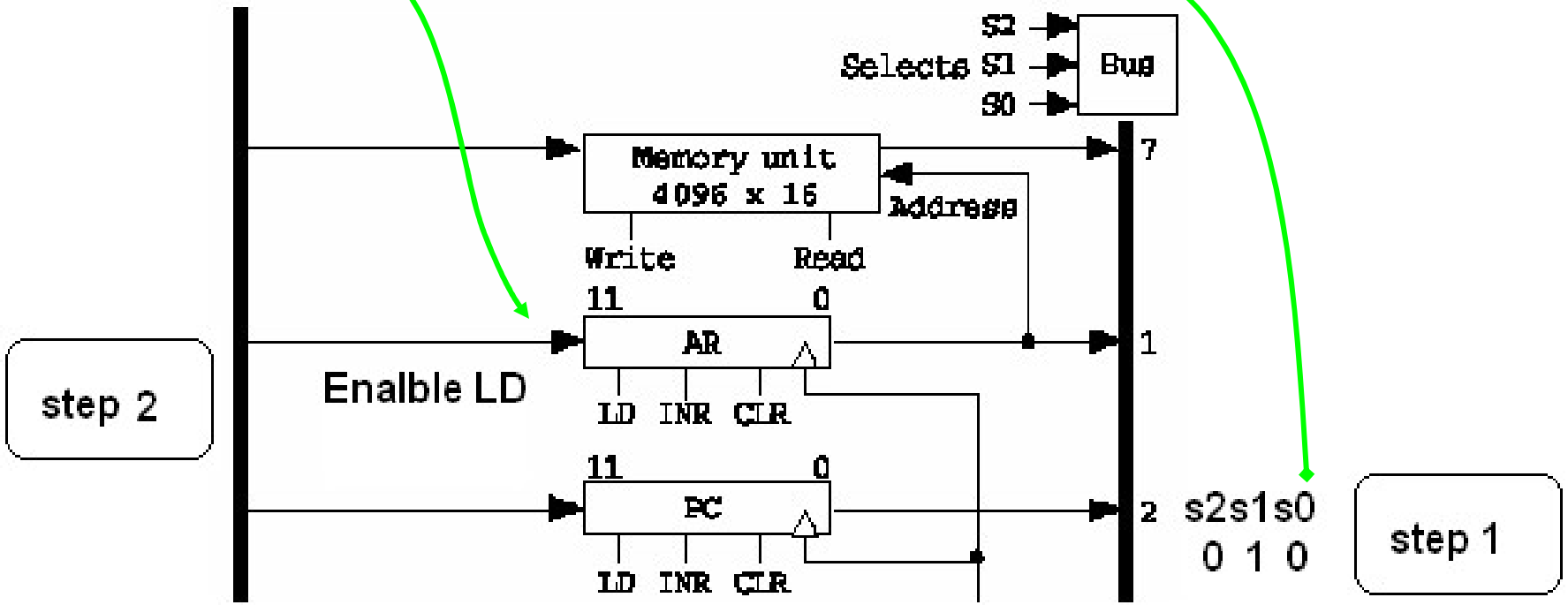
- At T3, microoperations which take place depend on the type of instruction. The four different paths are symbolized as follows,

Control function	Microoperation
D7`IT3: transfer	AR ← M[AR], indirect memory
D7`I`T3: transfer	Nothing, direct memory
D7I`T3: instruction	Execute a register-reference
D7IT3: instruction	Execute an I/O

Address transfer between PC and AR

T0: Since only AR is connected to the address inputs of memory, the address of instruction is transferred from PC to AR.

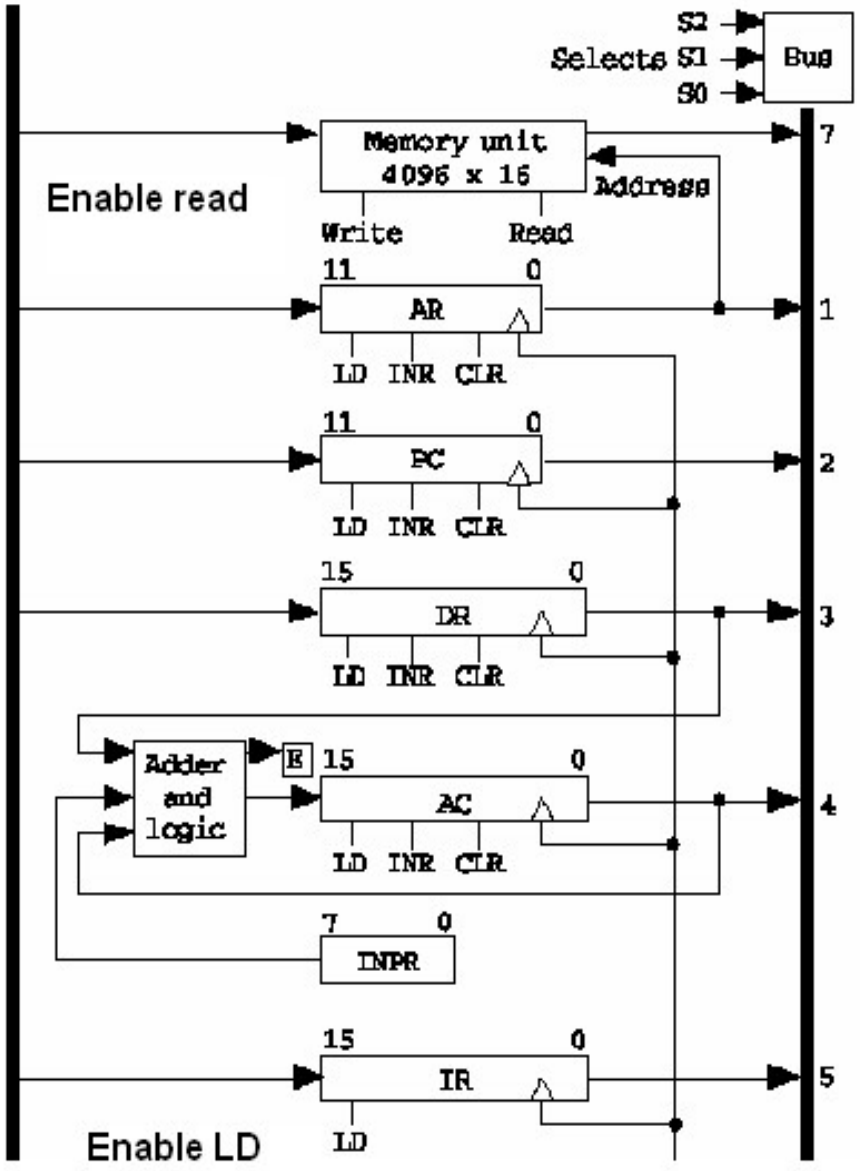
1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 = 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR (AR ←PC).



Data transfer between Memory and IR

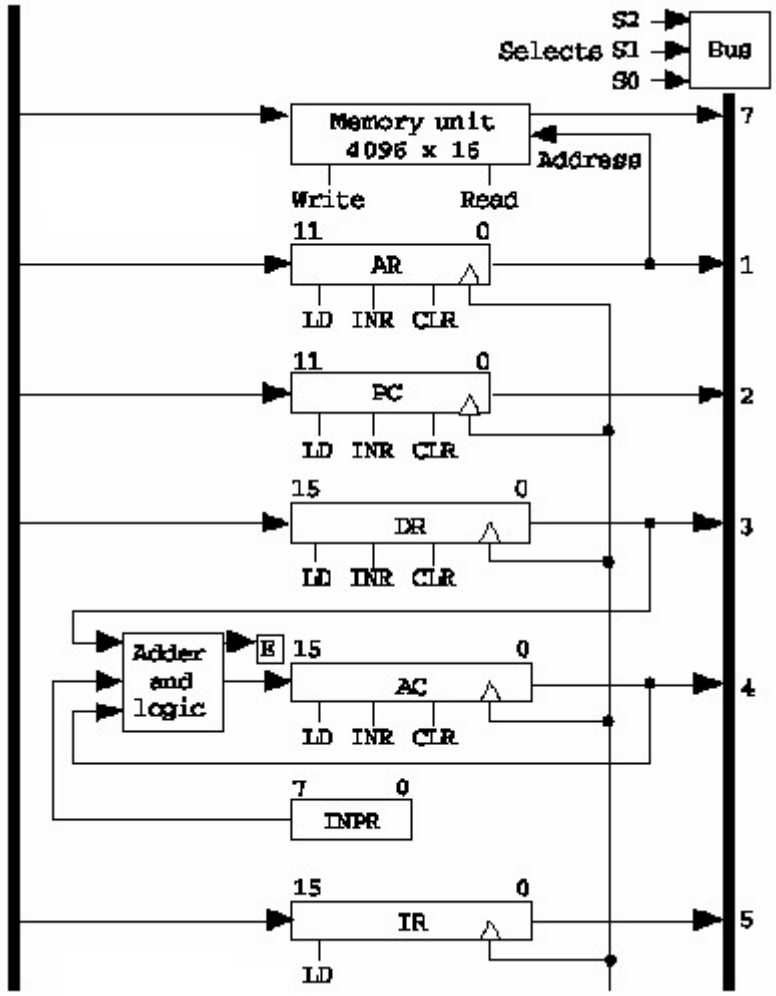
T1: The instruction read from memory is placed in IR. At the same time, PC is incremented to the address of the next instruction.

1. Enable ‘read input’ of the memory.
2. Place the content of memory onto the bus using the bus selection inputs S2S1S0 = 111. (Note that the address lines are always connected to AR, and the next instruction address has been already placed in AR.)
3. Transfer the content of the bus to IR by enabling LD input to IR
   (IR ←M[AR]).
4. Increment PC by enabling the INR input of PC (PC ←PC + 1).



Decoding at T2

T2: The operation code in IR is decoded; the indirect bit is transferred to I; the address part of the instruction is transferred to AR.



REGISTER-REFERENCE INSTRUCTIONS

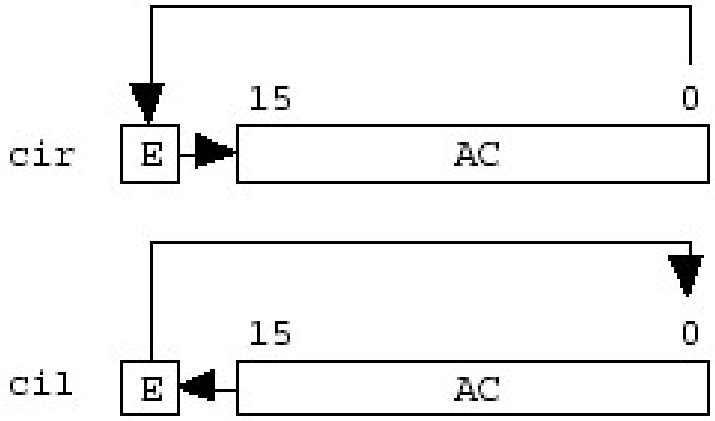
• The 12 register-reference instructions are recognized by I = 0 and D7 = 1 (IR(12-14) = 111). Each operation is designated by the presence of 1 in one of the bits in IR(0-11). Therefore D7I`T3 ≡ r = 1 is common to all register-transfer instructions.

Symbol	Control	Microoperations	Description
	r	SC ← 0 (Common to all, done in 1 cycle)	Clear SC
CLA	rB_{11}	AC ← 0	Clear AC
CLE	rB_{10}	E ← 0	Clear E
CMA	rB_9	AC ← $\underline{AC}$	Complement AC
CME	rB_8	E ← $\underline{E}$	Complement E
CIR	rB_7	AC ← shr AC, AC(15) ← E, E ← AC(0)	Circular right
CIL	rB_6	AC ← shl AC, AC(0) ← E, E ← AC(15)	Circular left
INC	rB_5	AC ← AC + 1	Increment AC
SPA	rB_4	If AC(15)=0 then PC ← PC + 1	Skip if positive
SNA	rB_3	If AC(15)=1 then PC ← PC + 1	Skip if negative
SZA	rB_2	If AC=0 then PC ← PC + 1	Skip if AC zero
SZE	rB_1	If E=0 then PC ← PC + 1	Skip if E zero
HLT	rB_0	S ← 0 (S is a start-stop flip-flop)	Halt computer

For example

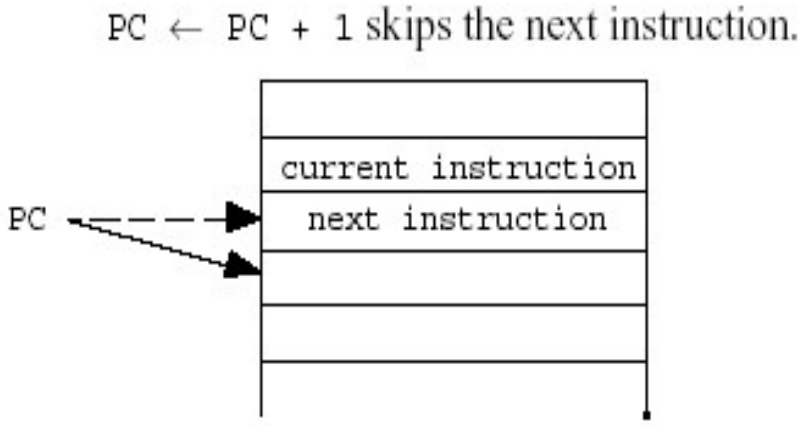
- B7 = 007 (in hexadecimal)., In binary this is equivalent to: 0000 0000 0111 (CIR)
- B6 = 006 (in hexadecimal)., In binary this is equivalent to: 0000 0000 0110 (CIL)

CIR and CIL microoperations.



For example

- B3 = 008 (in hexadecimal)., In binary this is equivalent to: 0000 0000 1000 (Complement E)
- B4 = 010 (Bi=bit in position i =4) in binary is 0000 0001 0000 (skip if positive)

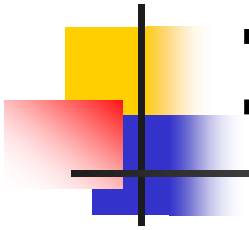


5.6 Memory Reference Instructions

• Opcode (000 - 110) or the decoded output Di (i = 0, ..., 6) are used to select one memory-reference operation out of 7.

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

Memory Reference Instructions



- Since the data stored in memory cannot be processed directly (the memory unit is not connected to the ALU), the actual execution in the bus system require a sequence of microoperations.

- (Note that T0-T2 for fetch an instruction; T3 for **AR ← M[AR]** if indirect memory addressing.

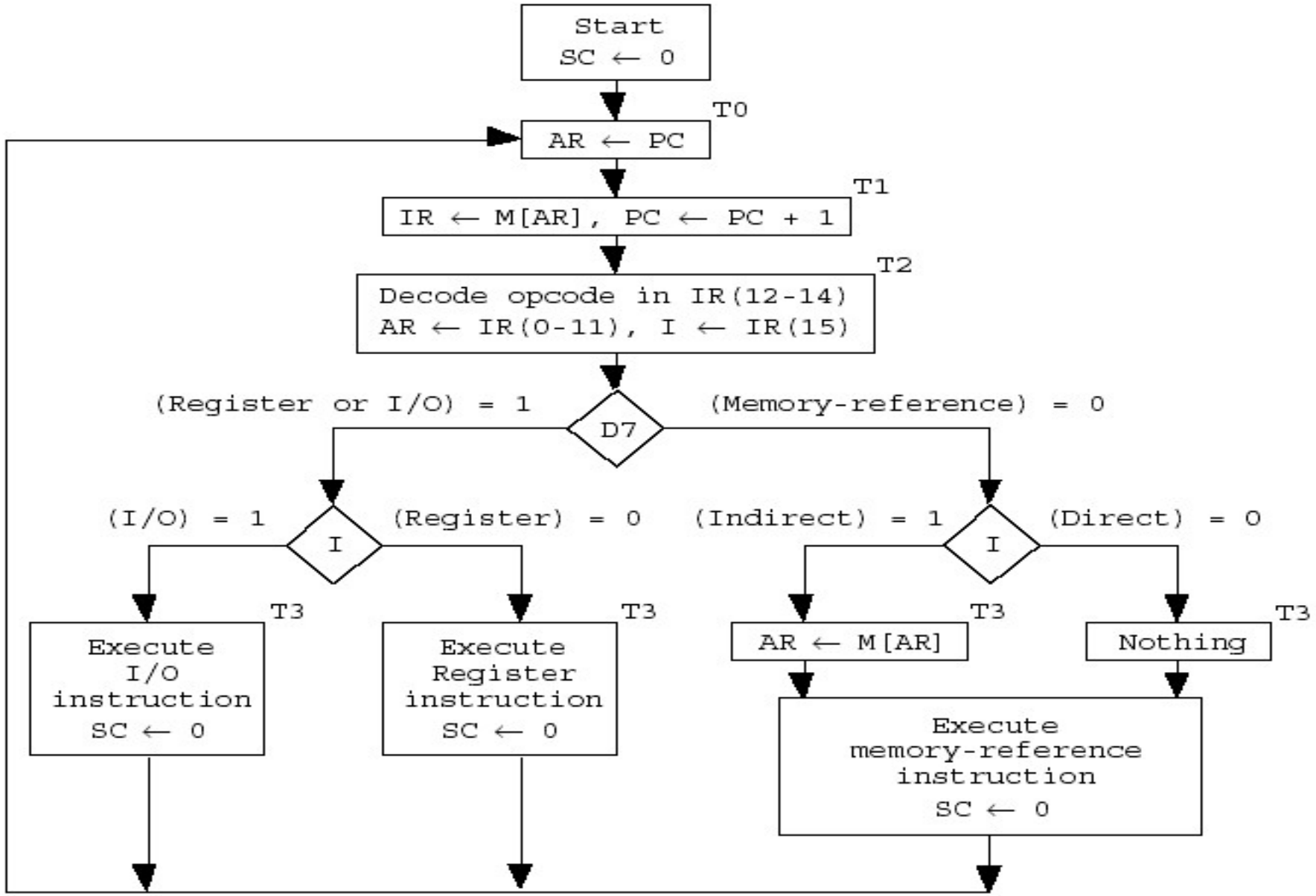


Figure: Flowchart for fetch & decode phases.

Computer Registers

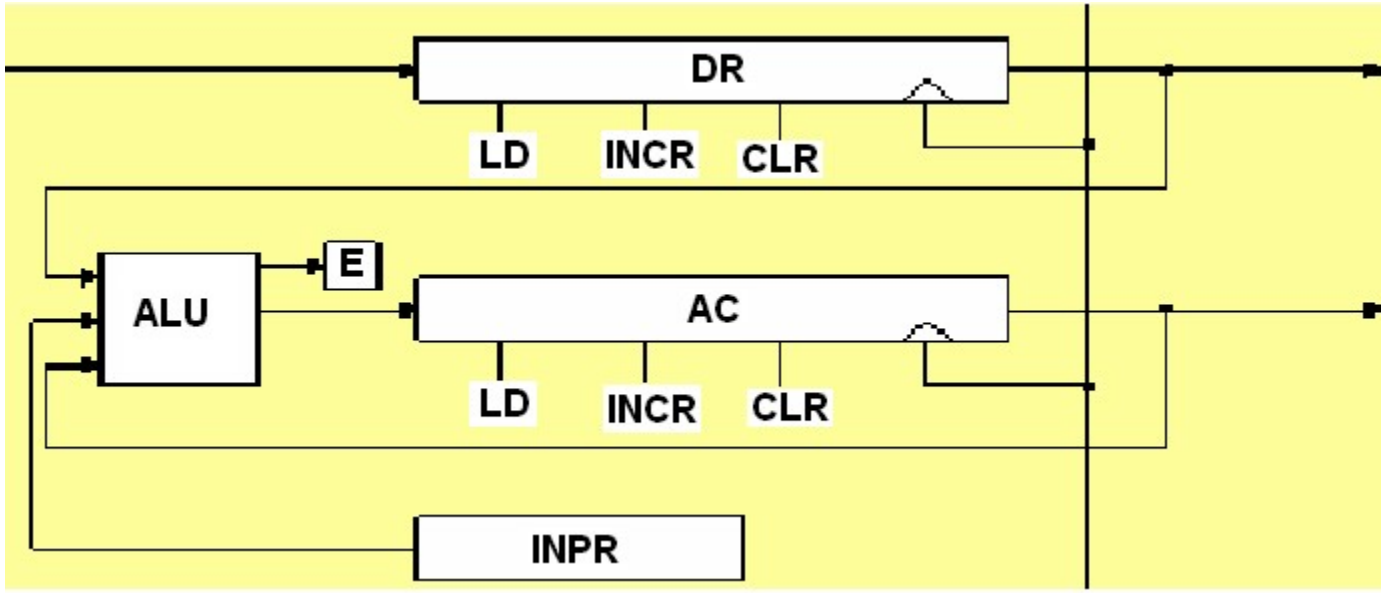
–Accumulator(AC) : takes input from ALU

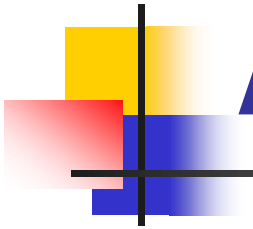
»The ALU takes input from DR, AC and INPR :

»**ADD** DR **to** AC, **AND** DR **to** AC

–Note) Input register is not connected to the bus.

–The input register is connected only to the ALU

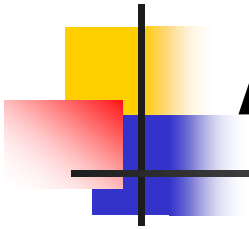




AND to AC

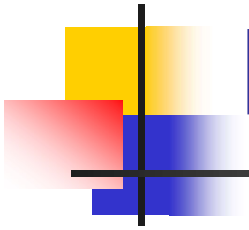
- **AND to AC**: Logical AND operation between AC and the memory word specified by AR.
- (Note that T0-T2 for fetch an instruction; T3 for **AR ← M[AR]** if indirect memory addressing.

- Need 2 more cycles for the AND logical operation since only DR is connected to ALU.)
- **D0T4: DR ← M[AR]**
- **D0T5: AC ← AC ^ DR, SC ← 0**
  - **SC – start counter**



ADD to AC

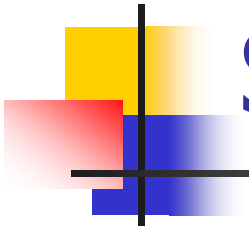
- **ADD to AC**: Arithmetic addition operation between AC and the memory word specified by AR.
- **D1T4: DR ← M[AR]**
- **D1T5: AC ← AC + DR, SC ← 0**



Load to AC

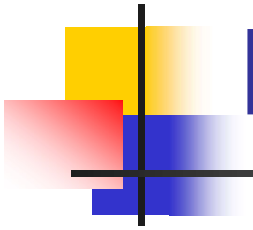
- **LDA**: Load to AC.
- (Need 2 cycles since AC input is not connected to the bus.)

- **D2T4: DR ← M[AR]**
- **D2T5: AC ← DR, SC ← 0**



Store AC

- **STA**: Store AC.
- **D3T4: M[AR] ← AC, SC ← 0**
- **BUN**: Branch unconditionally. Transfers the program to the instruction specified by AR. (Note that the branch target must be in AR beforehand.)
- **D4T4: PC ← AR, SC ← 0**

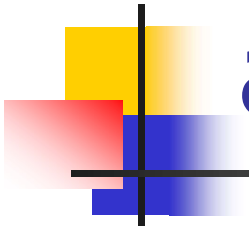


Branch unconditionally

- **BUN**: Branch unconditionally. Transfers the program to the instruction specified by AR. (Note that the branch target must be in AR beforehand.)

- **D4T4: PC ← AR, SC ← 0**

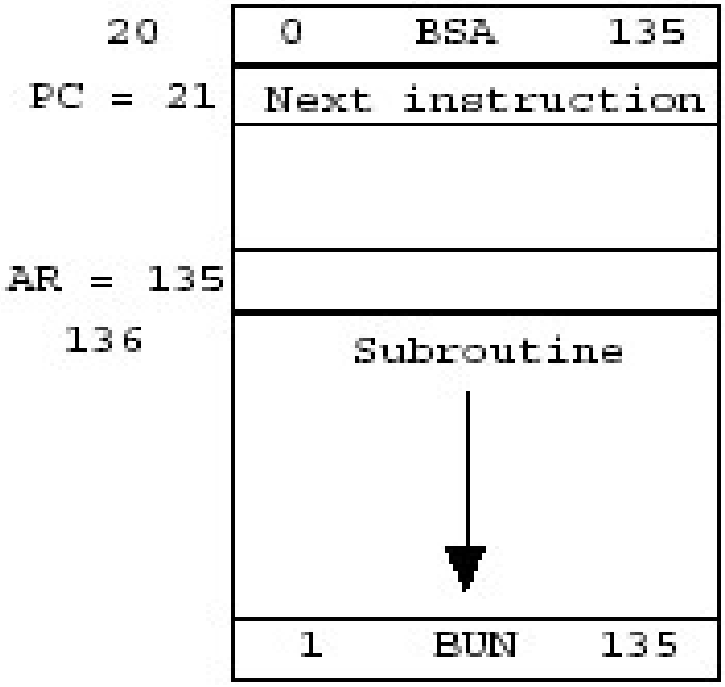
Branch and save return address



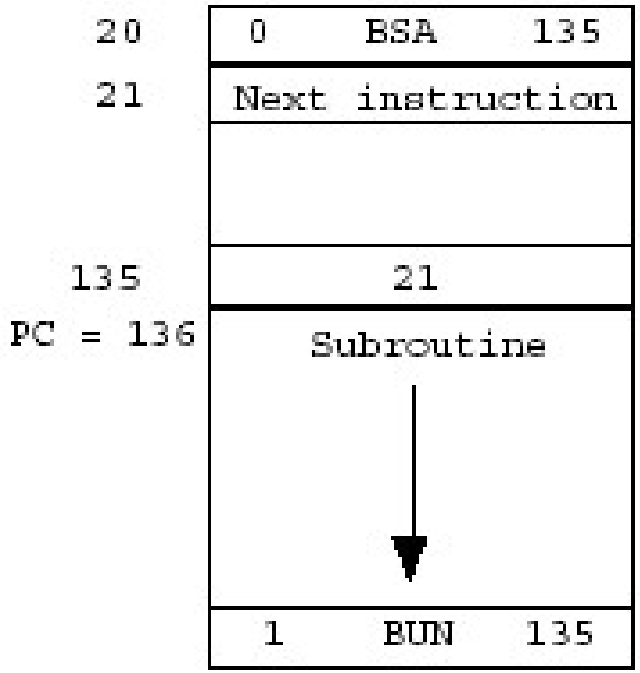
- **This instructioin is useful for branching to a position of the program called a subprogram**

- **BSA**: Branch and save return address. Branch to address AR and save PC address.
- • BSA is used to implement a subroutine call. The indirect BUN instruction at the end of the subroutine performs the subroutine return.

Branch and save return address

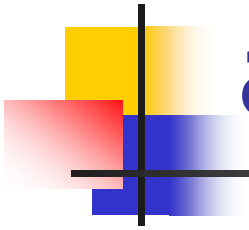


Memory, PC, and AR at time T_4

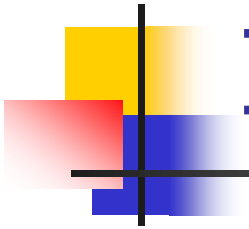


Memory and PC after BSA execution

Branch and save return address



- Note that the above microoperations require 2 cycles.
- **D5T4: M[AR] ← PC, AR ← AR + 1 (increment, INR AR)**
- **D5T5: PC ← AR, SC ← 0**



Increment and skip if zero

- **ISZ**: Increment and skip if zero.
- Programmer usually stores a negative number in the memory word (in two's complement form).
- As this negative number is repeatedly incremented by one, it eventually reaches zero. At that time PC is incremented by one in order to skip the next instruction.

Increment and skip if zero

- **increment: M[AR] ←M[AR] + 1, if (M[AR] + 1 = 0) then PC ← PC + 1**

- increment and skip if zero requires 3 cycles.
  - **D6T4: DR ← M[AR]**
  - **D6T5: DR ← DR + 1**
  - **D6T6: M[AR] ← DR, if DR=0 then PC ← PC + 1, SC ← 0**
- • The ISZ instructions is used to implement a loop.

Computer Instruction

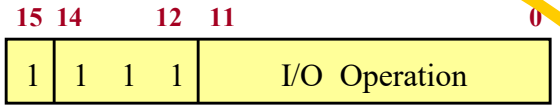
3 Instruction Code Formats :

1-Register-reference instruction

–7xxx (7800 ~ 7001) : CLA, CMA, ….

2-Input-Output instruction

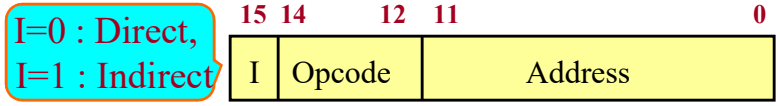
–Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ….



3-Memory-reference instruction

Opcode = 000 ~ 110

I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx



Symbol	Hexadecimal code I = 0	Hexadecimal code I = 1	Description
Register Reference			
CLA	0111 1000 0000 0000		Clear AC
CLE	0111 0100 0000 0000		Clear E
CMA	0111 0010 0000 0000		Complement AC
CME	0111 0001 0000 0000		Complement E
CIR	0111 0000 1000 0000		Circulate right AC and E
CIL	0111 0000 0100 0000		Circulate left AC and E
INC	0111 0000 0010 0000		Increment AC
SPA	0111 0000 0001 0000		Skip next instruction if AC positive
SNA	0111 0000 0000 1000		Skip next instruction if AC negative
SZA	0111 0000 0000 0100		Skip next instruction if AC zero
SZE	0111 0000 0000 0010		Skip next instruction if E is 0
HLT	0111 0000 0000 0001		Halt computer
			7800 (B_{11} bit)
Input-Output			
INP		1111 1000 0000 0000	Input character to AC
OUT		1111 0100 0000 0000	Output character from AC
SKI		1111 0010 0000 0000	Skip on input flag
SKO		1111 0001 0000 0000	Skip on output flag
ION		1111 0000 1000 0000	Interrupt on
IOF		1111 0000 0100 0000	Interrupt off
Memory Reference			
AND	0000 xxxx xxxx xxxx	1000 xxxx xxxx xxxx	AND memory word to AC
ADD	0001 xxxx xxxx xxxx	1001 xxxx xxxx xxxx	ADD memory word to AC
LDA	0010 xxxx xxxx xxxx	1010 xxxx xxxx xxxx	Load memory word to AC
STA	0011 xxxx xxxx xxxx	1011 xxxx xxxx xxxx	Store content of AC in memory
BUN	0100 xxxx xxxx xxxx	1100 xxxx xxxx xxxx	Branch unconditionally
BSA	0101 xxxx xxxx xxxx	1101 xxxx xxxx xxxx	Branch and save return address
ISZ	0110 xxxx xxxx xxxx	1110 xxxx xxxx xxxx	Increment and skip if zero

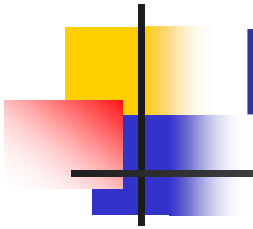
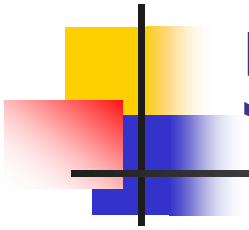


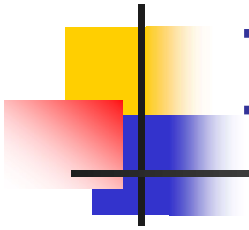
Figure 5-11

- Summary of memory-reference instructions



5.7 IO and Interrupt

- Input-Output Configuration :
  - Input Register(***INPR***), Output Register(***OUTR***)
    - These two registers communicate with a communication interface serially and with the AC in parallel
    - Each quantity of information has eight bits of an alphanumeric code



IO and Interrupt

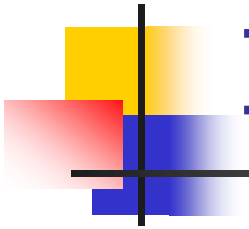
- Input Flag(***FGI***), Output Flag(***FGO***)
  - FGI : ***set*** when INPR has information, ***clear*** when INPR is empty
  - FGO : ***set*** when operation is completed, ***clear*** when output device is active (for example a printer is in the process of printing)

Input-output:

$D_7IT_3 = p$ (common to all input–output instructions)

$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$

	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$



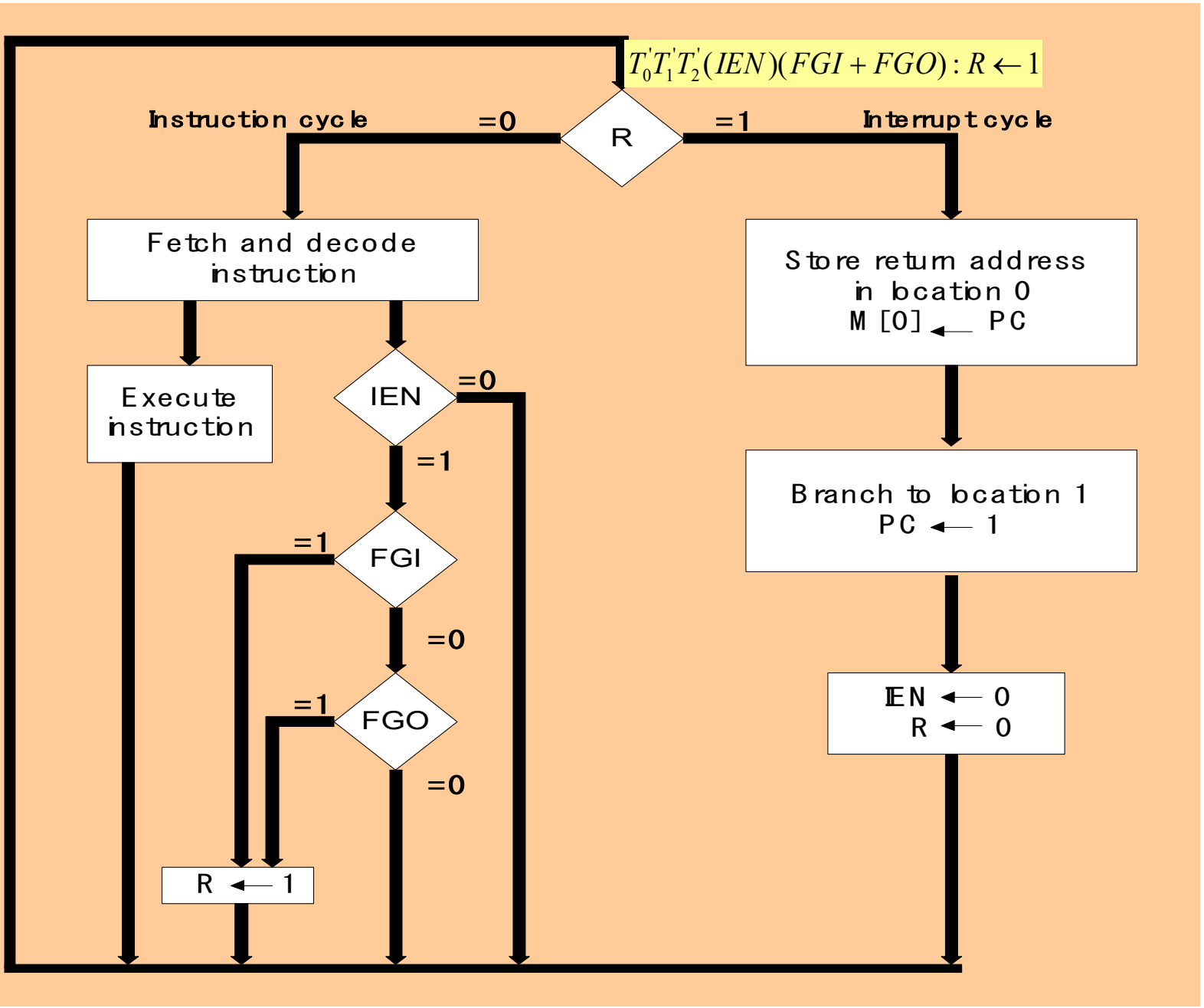
IO instructions

- These instructions are executed with the clock transition associated with timing signal T3
- For these instructions, D7=1 and I=1
- The control function is distinguished by one of the bits in IR(6-11)

Program Interrupt

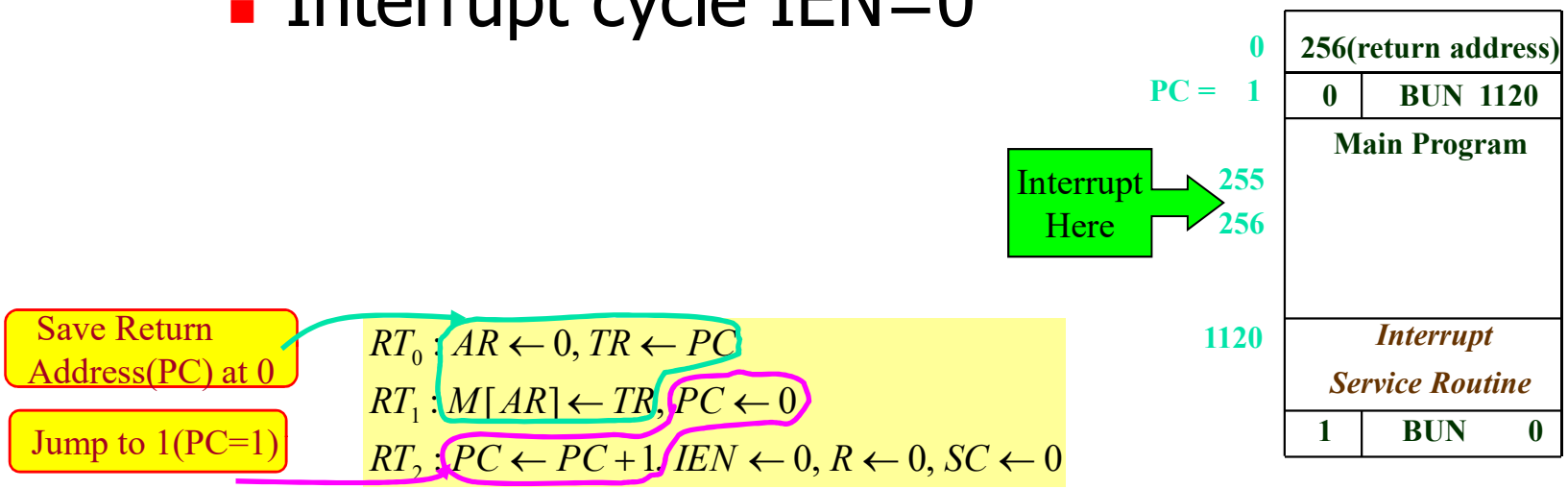
- Program Interrupt
  - Two I/O Transfer Modes
    - 1) Programmed I/O
    - 2) Interrupt-initiated I/O (FGI FGO)

- IEN: interrupt enable flip-flop
- R: interrupt flip-flop


$T_0'T_1'T_2'(IEN)(FGI + FGO): R \leftarrow 1$
Instruction cycle
=0
R
=1
Interrupt cycle
Fetch and decode instruction
Store return address in location 0
M [0] ← PC
Execute instruction
IEN
=0
=1
Branch to location 1
PC ← 1
=1
FGI
=0
IEN ← 0
R ← 0
=1
FGO
=0
R ← 1

Program Interrupt

- Demonstration of the interrupt cycle :
  - The memory location at address 0 is the place for storing the return address
  - Interrupt Branch to memory location 1
  - Interrupt cycle IEN=0



Mano's Computer: RTL

TABLE 5-6 Control Functions and Microoperations for the Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	D_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
	$T_0'T_1'T_2'(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
	$D_7I'T_3 = r$ (common to all register-reference instructions)	
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-output:		
	$D_7IT_3 = p$ (common to all input–output instructions)	
	$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$