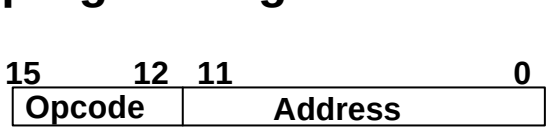
BASIC COMPUTER ORGANIZATION AND DESIGN

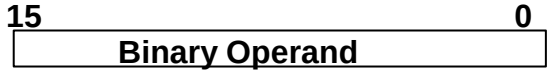
- **Instruction Codes**
- **Computer Registers**
- **Computer Instructions**
- **Timing and Control**
- **Instruction Cycle**
- **Memory Reference Instructions**
- **Input-Output and Interrupt**
- **Complete Computer Description**
- **Design of Basic Computer**
- **Design of Accumulator Logic**

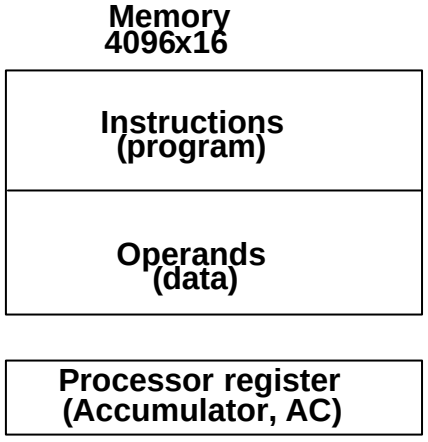
INSTRUCTION CODES

- **Program:**
  A set of instructions that specify the *operations*, *operands*, and the *sequence* by which processing has to occur.
- **Instruction Code:**
  A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)
  -->*macro-operation*
  - usually divided into *operation code*, *operand address*, *addressing mode*, etc.
  - basic addressing modes *Immediate*, *Direct*, *Indirect*
- **Simplest stored program organization**

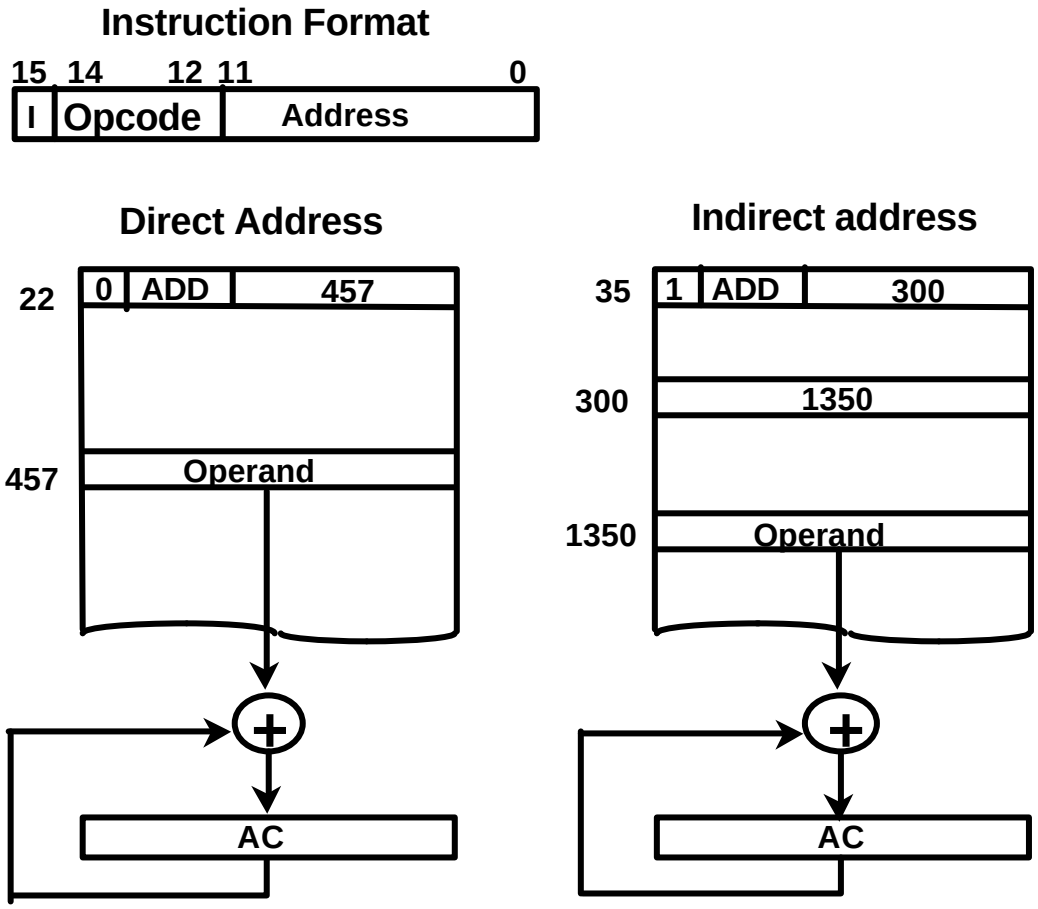


Instruction Format





INDIRECT ADDRESS

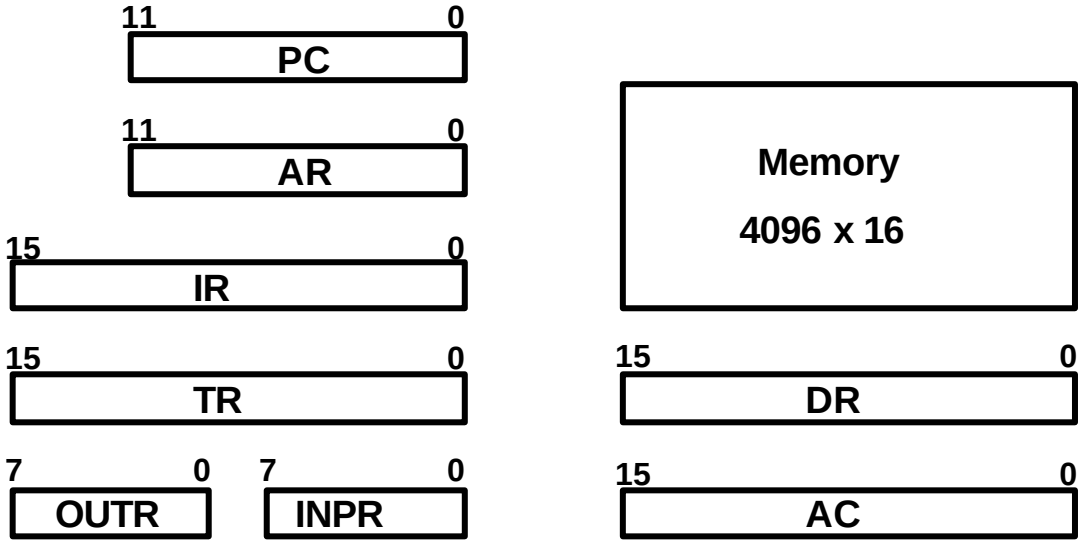


Effective Address(EFA, EA)

The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction

COMPUTER REGISTERS

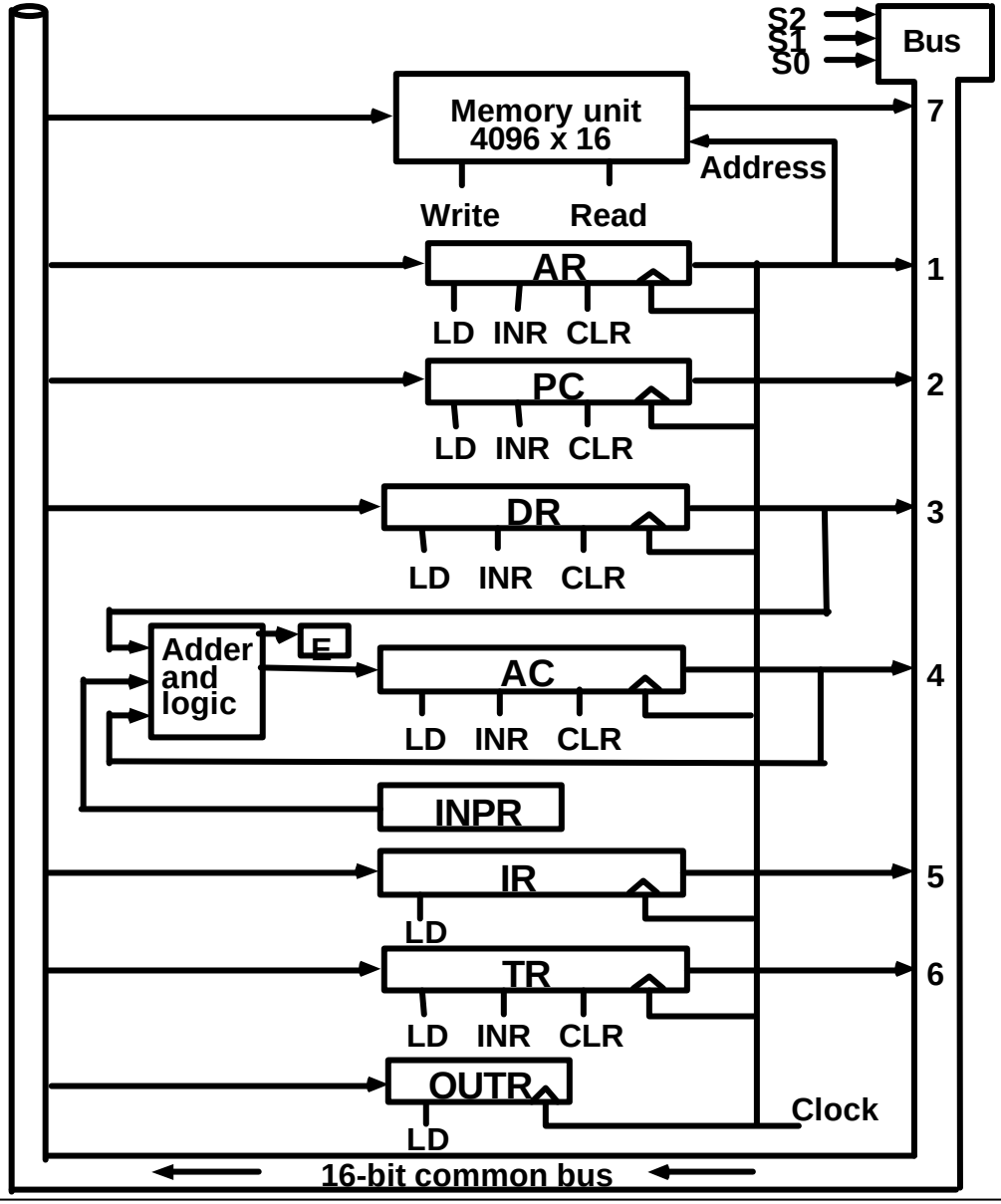
Registers in the Basic Computer



List of BC Registers

DR	16	Data Register	Holds memory operand
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

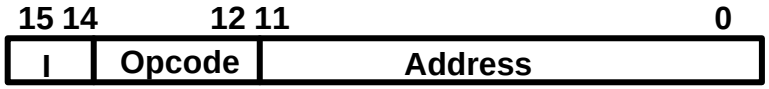
COMMON BUS SYSTEM



COMPUTER(BC) INSTRUCTIONS

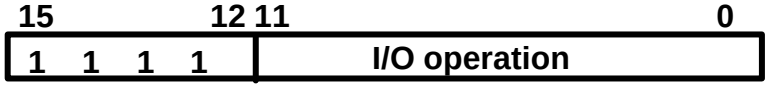
Basic Computer Instruction code format

Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)

Input-Output Instructions (OP-code =111, I = 1)



BASIC COMPUTER INSTRUCTIONS

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 AC from memory
STA	3xxx	Bxxx	Store content of AC into 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		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
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		Interrupt on
IOF	F040		Interrupt off

INSTRUCTION SET COMPLETENESS

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

Instruction Types

Functional Instructions

- **Arithmetic, logic, and shift instructions**
- **ADD, CMA, INC, CIR, CIL, AND, CLA**

Transfer Instructions

- **Data transfers between the main memory and the processor registers**
- **LDA, STA**

Control Instructions

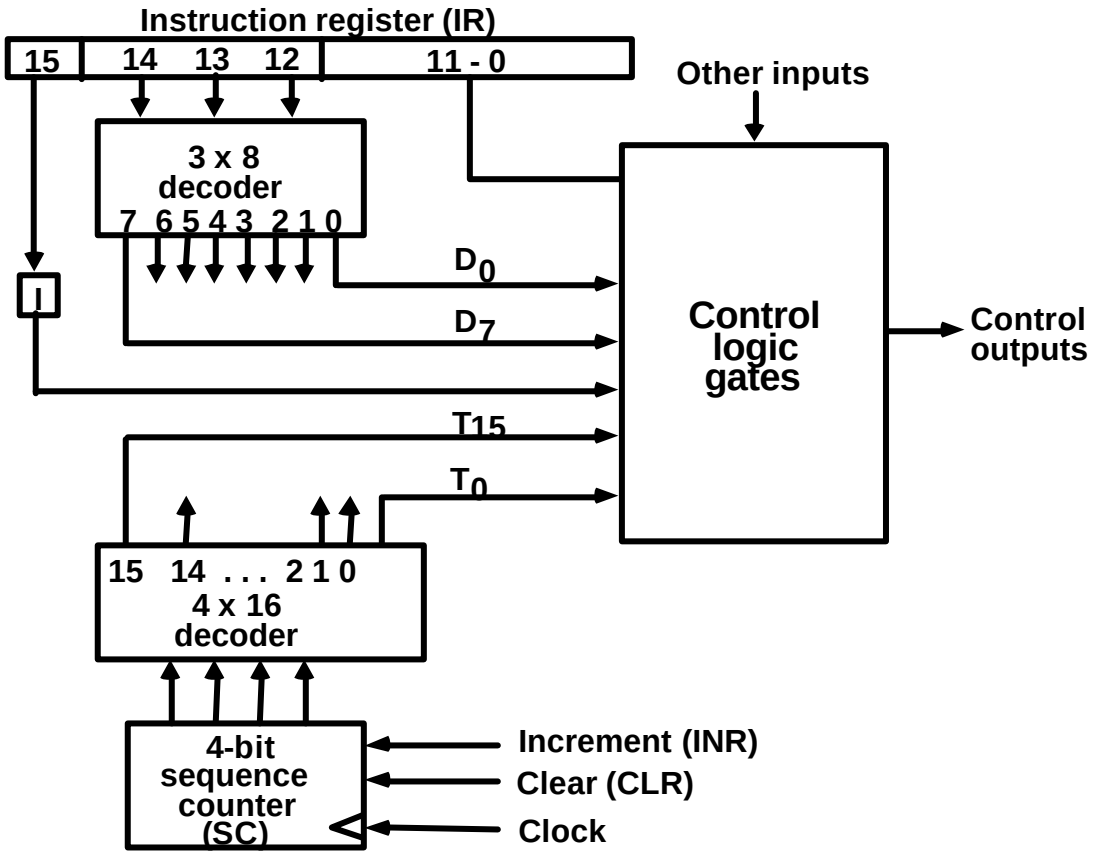
- **Program sequencing and control**
- **BUN, BSA, ISZ**

Input/Output Instructions

- **Input and output**
- **INP, OUT**

TIMING AND CONTROL

Control unit of basic computer



Control unit implementation

- Hardwired Implementation
- Microprogrammed Implementation

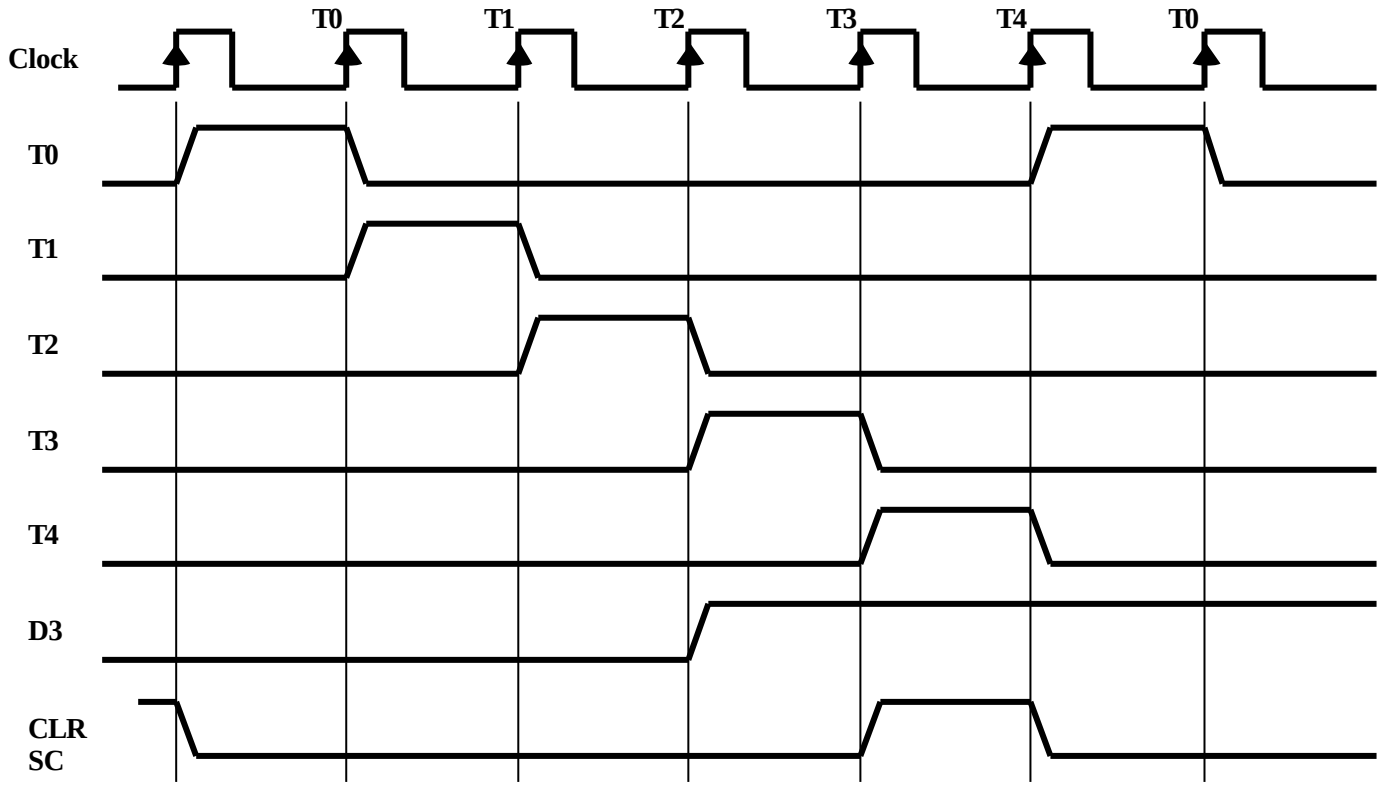
TIMING SIGNALS

- Generated by 4-bit sequence counter and 4x16 decoder
- The SC can be incremented or cleared.

- Example: $T_0, T_1, T_2, T_3, T_4, T_0, T_1, \ldots$

  Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.

D_3T_4: SC ← 0

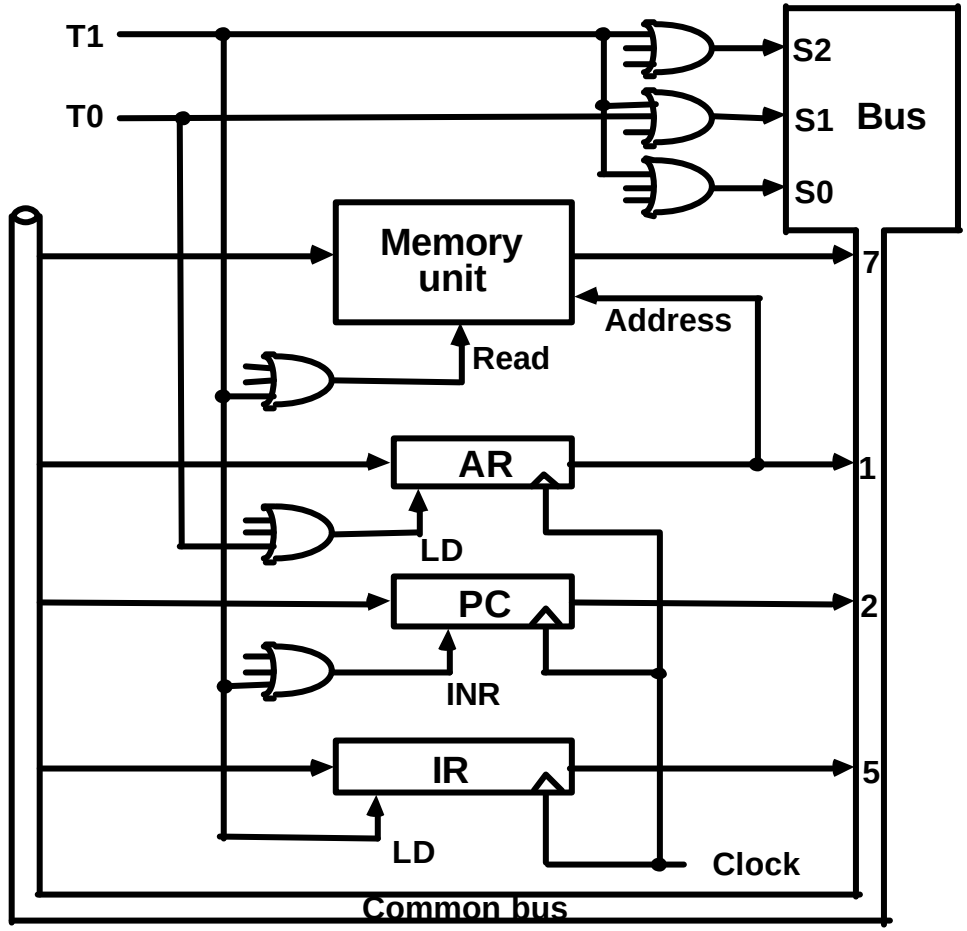


INSTRUCTION CYCLE

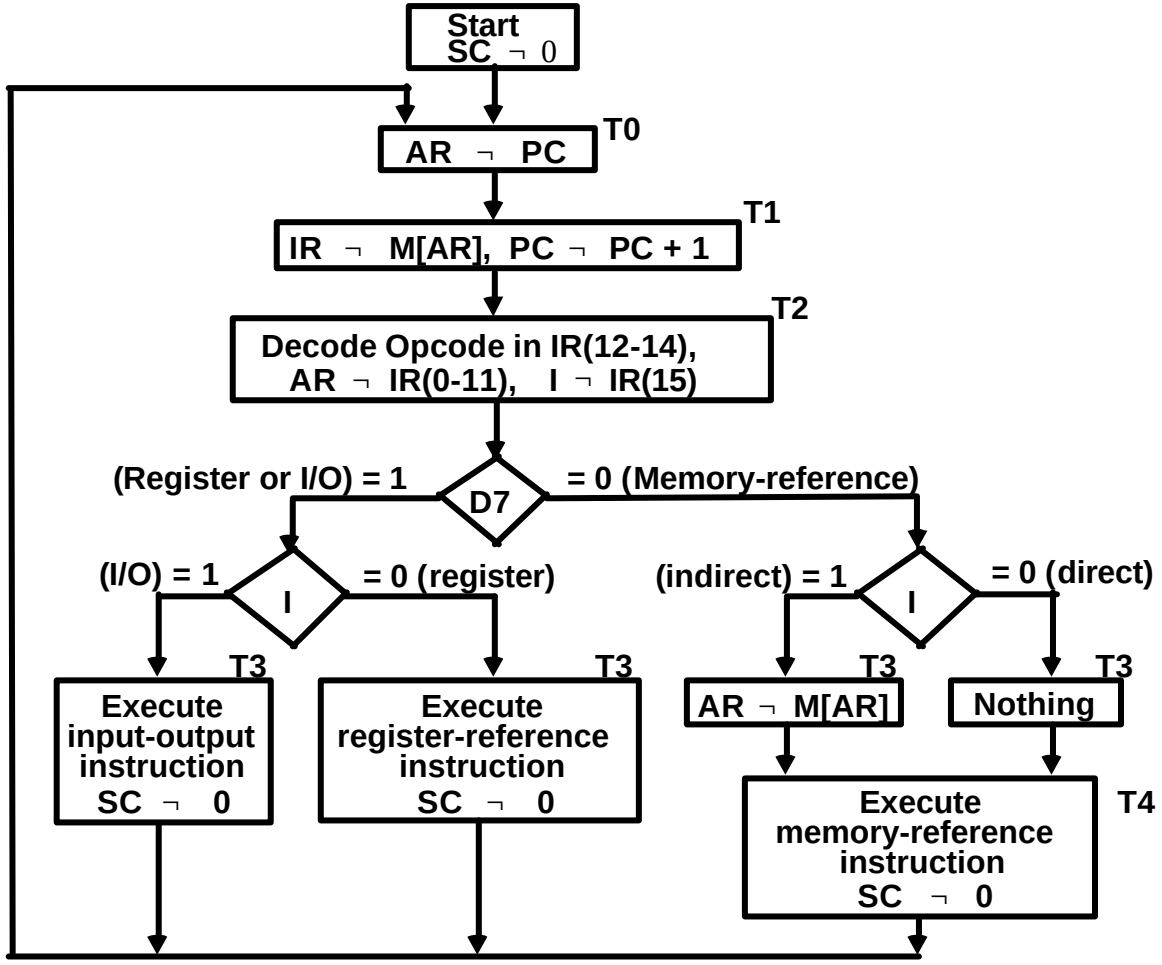
BC Instruction cycle: [Fetch Decode [Indirect] Execute]*

- **Fetch and Decode**

T0: AR ¬ PC (S0S1S2=010, T0=1)
T1: IR ¬ M [AR], PC ¬ PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ¬ Decode IR(12-14), AR ¬ IR(0-11), I ¬ IR(15)



DETERMINE THE TYPE OF INSTRUCTION



D'_7IT_3: AR ¬ M[AR]
$D'_7I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instr.
D_7IT_3: Execute an input-output instr.

REGISTER REFERENCE INSTRUCTIONS

Register Reference Instructions are identified when

- **$D_7 = 1$, $I = 0$**
- **Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR**
- **Execution starts with timing signal T_3**

$r = D_7\, I'\, T_3$ => Register Reference Instruction
$B_i = IR(i)$, i=0,1,2,...,11

	r:	SC ¬ 0
CLA	rB_{11}:	AC ¬ 0
CLE	rB_{10}:	E ¬ 0
CMA	rB_9:	AC ¬ AC'
CME	rB_8:	E ¬ E'
CIR	rB_7:	AC ¬ shr AC, AC(15) ¬ E, E ¬ AC(0)
CIL	rB_6:	AC ¬ shl AC, AC(0) ¬ E, E ¬ AC(15)
INC	rB_5:	AC ¬ AC + 1
SPA	rB_4:	if (AC(15) = 0) then (PC ¬ PC+1)
SNA	rB_3:	if (AC(15) = 1) then (PC ¬ PC+1)
SZA	rB_2:	if (AC = 0) then (PC ¬ PC+1)
SZE	rB_1:	if (E = 0) then (PC ¬ PC+1)
HLT	rB_0:	S ¬ 0 (S is a start-stop flip-flop)

MEMORY REFERENCE INSTRUCTIONS

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

- The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR Instruction starts with T_4

AND to AC

D_0T_4: DR ← M[AR] Read operand

D_0T_5: AC ← AC ∧ DR, SC ← 0 AND with AC

ADD to AC

D_1T_4: DR ← M[AR] Read operand

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0 Add to AC and store carry in E

MEMORY REFERENCE INSTRUCTIONS

LDA: Load to AC

D_2T_4: DR ¬ M[AR]

D_2T_5: AC ¬ DR, SC ¬ 0

STA: Store AC

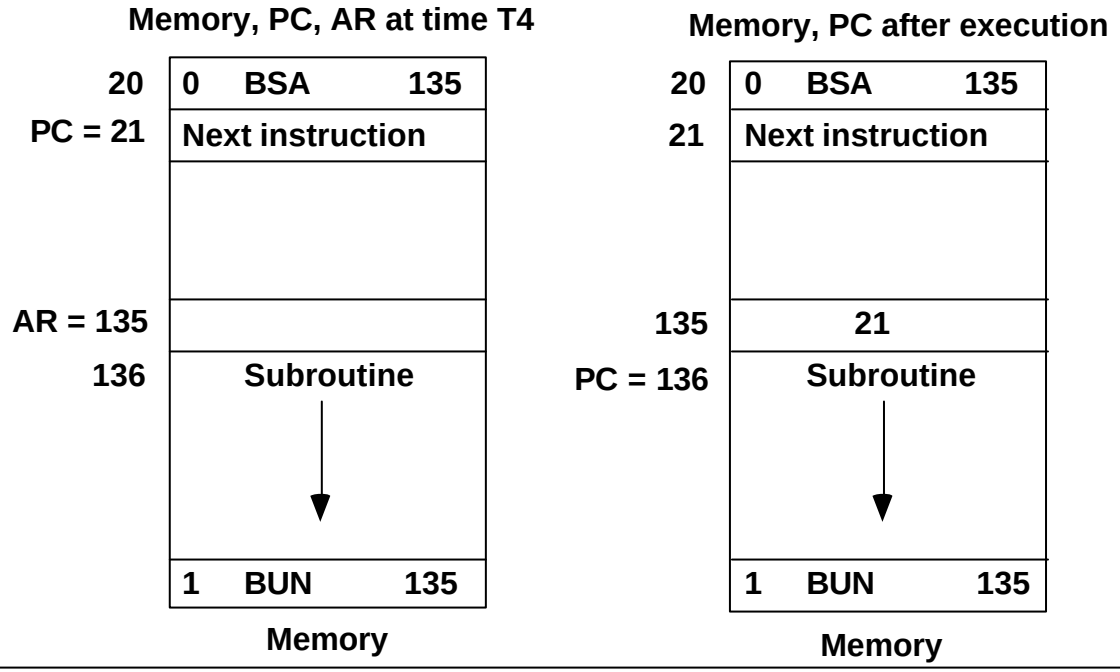
D_3T_4: M[AR] ¬ AC, SC ¬ 0

BUN: Branch Unconditionally

D_4T_4: PC ¬ AR, SC ¬ 0

BSA: Branch and Save Return Address

M[AR] ¬ PC, PC ¬ AR + 1



MEMORY REFERENCE INSTRUCTIONS

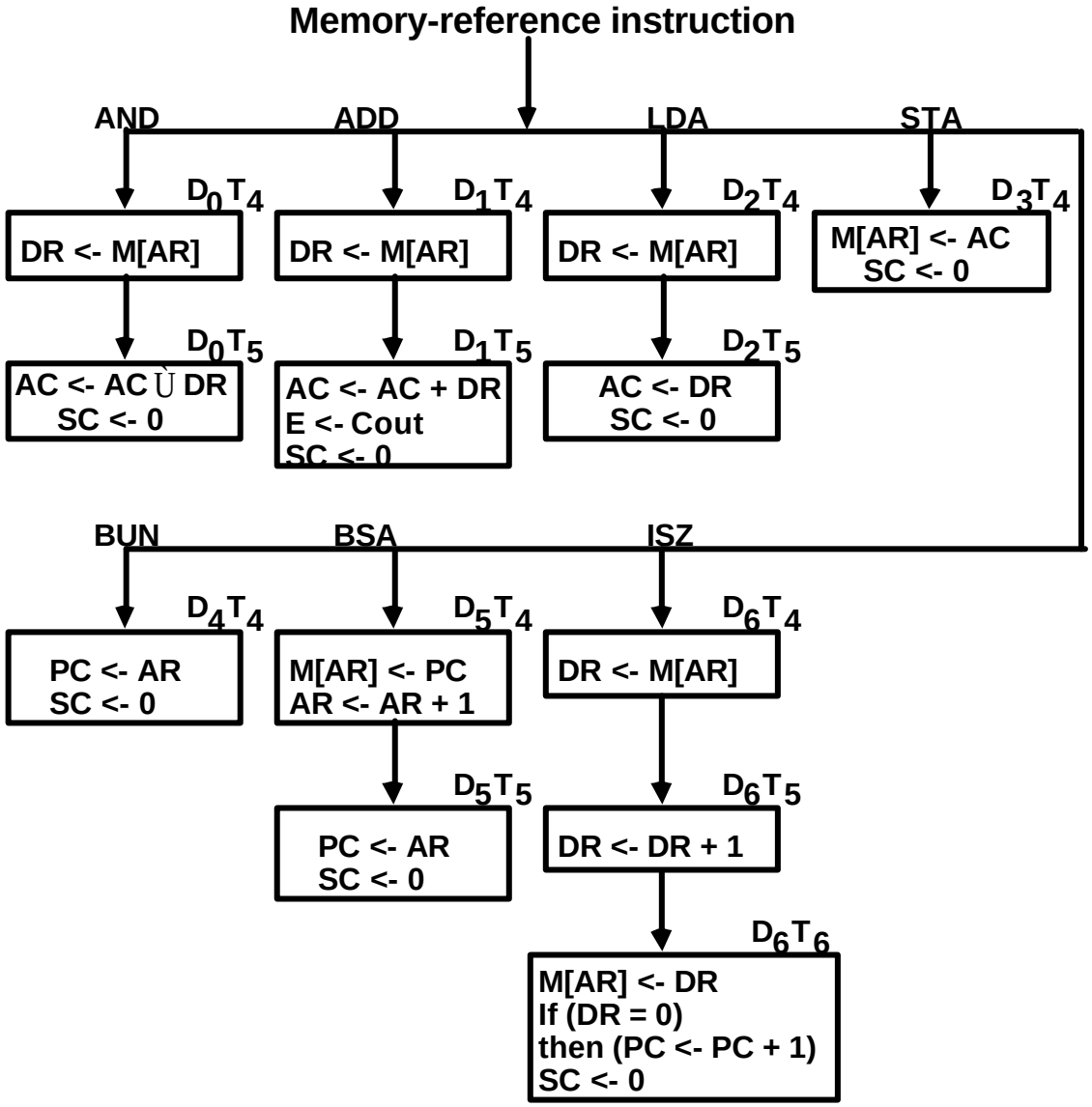
BSA:

D_5T_4: M[AR] ¬ PC, AR ¬ AR + 1
D_5T_5: PC ¬ AR, SC ¬ 0

ISZ: Increment and Skip-if-Zero

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

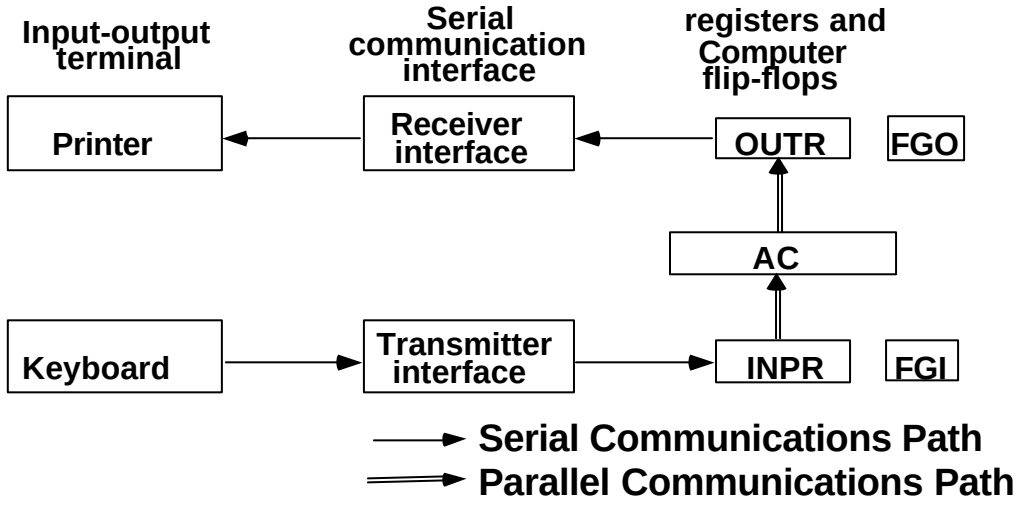
FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS



INPUT-OUTPUT AND INTERRUPT

A Terminal with a keyboard and a Printer

- **Input-Output Configuration**



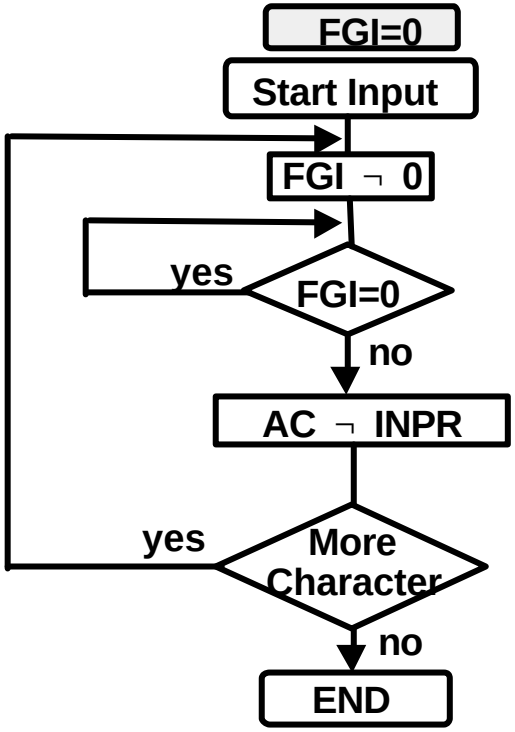
INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to synchronize the timing difference between I/O device and the computer

PROGRAM CONTROLLED DATA TRANSFER

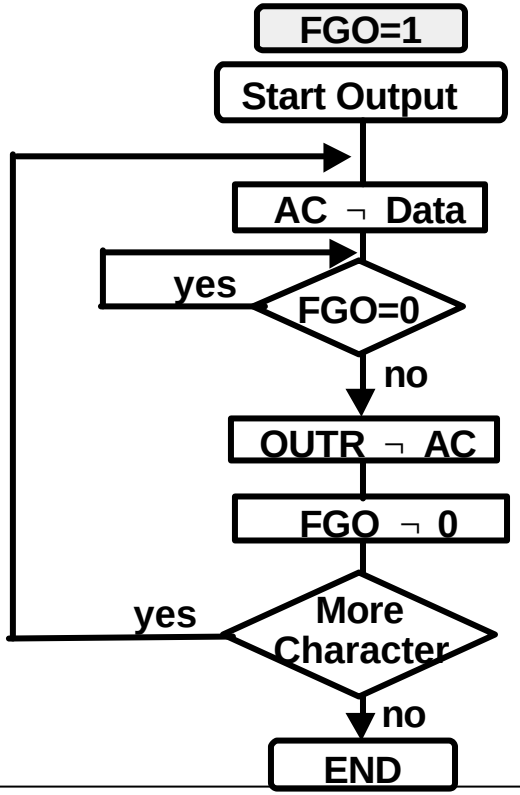
-- CPU --

```
/* Input */        /* Initially FGI = 0 */
  loop:  If FGI = 0 goto loop
         AC ¬ INPR, FGI ¬ 0

/* Output */       /* Initially FGO = 1 */
  loop:  If FGO = 0 goto loop
         OUTR ¬ AC, FGO ¬ 0
```



-- I/O Device --

```
loop: If FGI = 1 goto loop
      INPR ¬ new data, FGI ¬ 1

loop: If FGO = 1 goto loop
      consume OUTR, FGO ¬ 1
```



INPUT-OUTPUT INSTRUCTIONS

$D_7IT_3 = p$
$IR(i) = B_i$, i = 6, …, 11

INP	pB_{11}: AC(0-7) ¬ INPR, FGI ¬ 0	Input char. to AC
OUT	pB_{10}: OUTR ¬ AC(0-7), FGO ¬ 0	Output char. from AC
SKI	pB_9: if(FGI = 1) then (PC ¬ PC + 1)	Skip on input flag
SKO	pB_8: if(FGO = 1) then (PC ¬ PC + 1)	Skip on output flag
ION	pB_7: IEN ¬ 1	Interrupt enable on
IOF	pB_6: IEN ¬ 0	Interrupt enable off

PROGRAM-CONTROLLED INPUT/OUTPUT

- **Program-controlled I/O**
  - **Continuous CPU involvement**
    I/O takes valuable CPU time
  - **CPU slowed down to I/O speed**
  - **Simple**
  - **Least hardware**

Input

```
LOOP,   SKI   DEV
        BUN   LOOP
        INP   DEV
```

Output

```
LOOP,   LD    DATA
LOP,    SKO   DEV
        BUN   LOP
        OUT   DEV
```

INTERRUPT INITIATED INPUT/OUTPUT

- Open communication only when some data has to be passed --> *interrupt*.

- The I/O interface, instead of the CPU, monitors the I/O device.

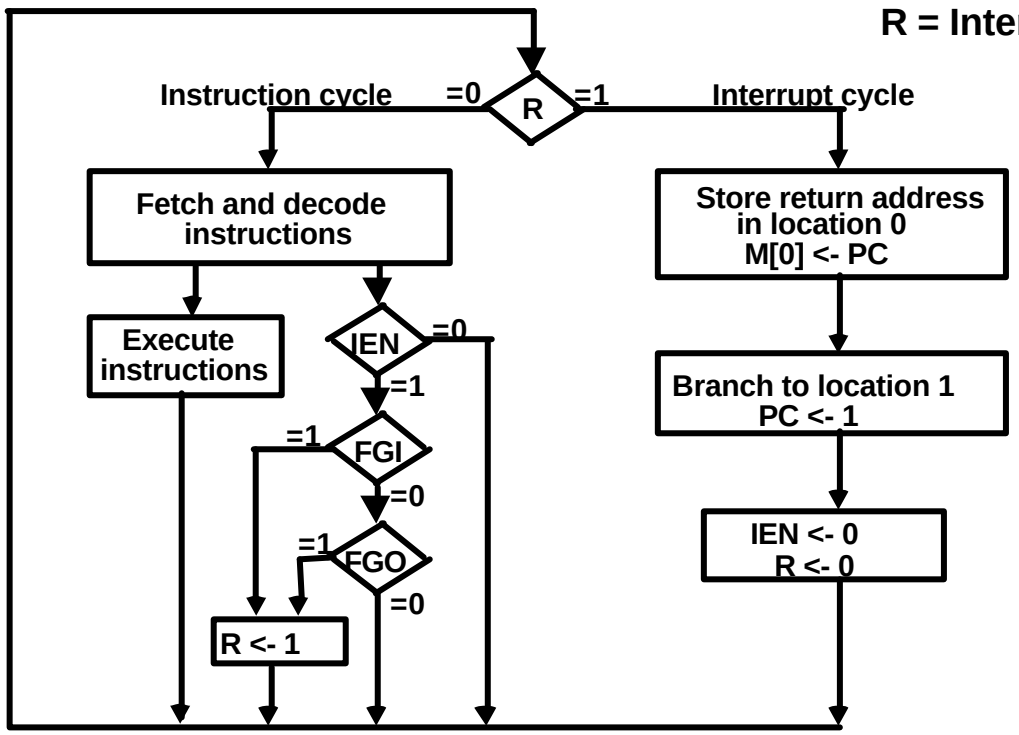
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

* IEN (Interrupt-enable flip-flop)

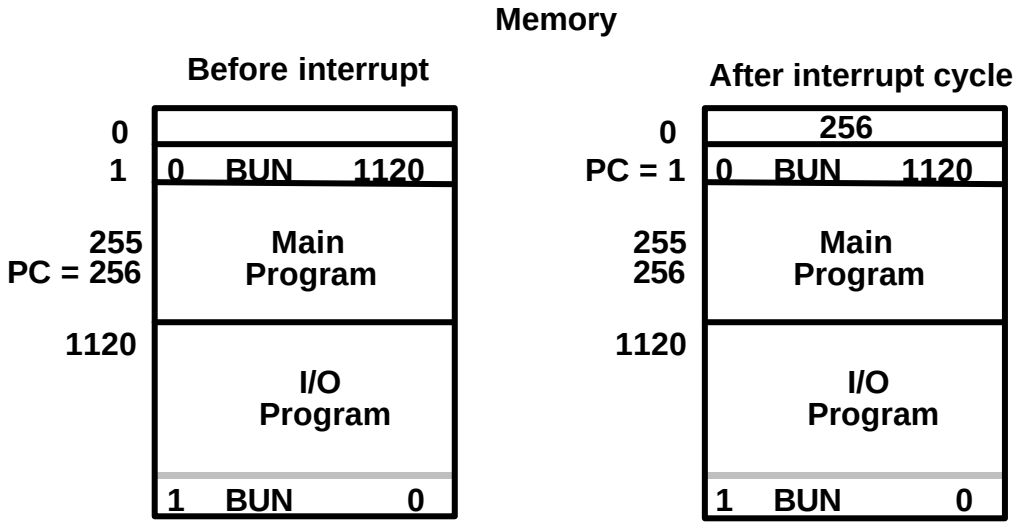
  - can be set and cleared by instructions
  - when cleared, the computer cannot be interrupted

FLOWCHART FOR INTERRUPT CYCLE



- The interrupt cycle is a HW implementation of a branch and save return address operation.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

REGISTER TRANSFER OPERATIONS IN INTERRUPT CYCLE



Register Transfer Statements for Interrupt Cycle

- R F/F ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$
  Û $T_0'T_1'T_2'$ (IEN)(FGI + FGO): R ← 1

- The fetch and decode phases of the instruction cycle must be modified:Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle :

RT_0: AR ← 0, TR ← PC

RT_1: M[AR] ← TR, PC ← 0

RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

FURTHER QUESTIONS ON INTERRUPT

Questions on Interrupt

How can the CPU recognize the device requesting an interrupt ?

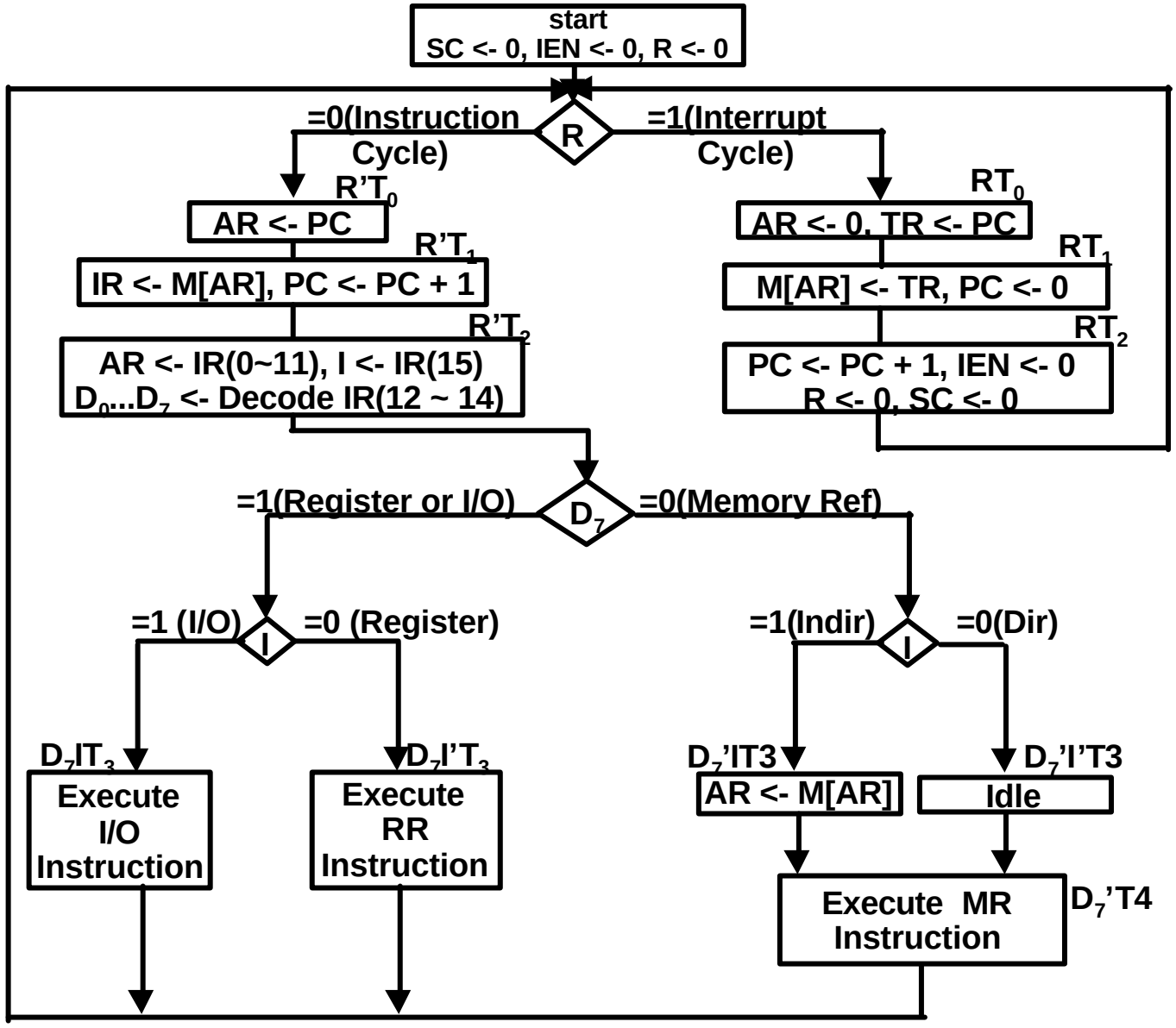
Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case ?

Should any device be allowed to interrupt the CPU while another interrupt is being serviced ?

How can the situation be handled when two or more interrupt requests occur simultaneously ?

COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations



COMPLETE COMPUTER DESCRIPTION

Microoperations

Fetch	**R'T0:**	**AR <- PC**
	R'T1:	**IR <- M[AR], PC <- PC + 1**
Decode	**R'T2:**	**D0, ..., D7 <- Decode IR(12 ~ 14),**
		AR <- IR(0 ~ 11), I <- IR(15)
Indirect	**D7'IT3:**	**AR <- M[AR]**
Interrupt		
T0'T1'T2'(IEN)(FGI + FGO):		**R <- 1**
	RT0:	**AR <- 0, TR <- PC**
	RT1:	**M[AR] <- TR, PC <- 0**
	RT2:	**PC <- PC + 1, IEN <- 0, R <- 0, SC <- 0**
Memory-Reference		
AND	**D0T4:**	**DR <- M[AR]**
	D0T5:	**AC <- AC . DR, SC <- 0**
ADD	**D1T4:**	**DR <- M[AR]**
	D1T5:	**AC <- AC + DR, E <- Cout, SC <- 0**
LDA	**D2T4:**	**DR <- M[AR]**
	D2T5:	**AC <- DR, SC <- 0**
STA	**D3T4:**	**M[AR] <- AC, SC <- 0**
BUN	**D4T4:**	**PC <- AR, SC <- 0**
BSA	**D5T4:**	**M[AR] <- PC, AR <- AR + 1**
	D5T5:	**PC <- AR, SC <- 0**
ISZ	**D6T4:**	**DR <- M[AR]**
	D6T5:	**DR <- DR + 1**
	D6T6:	**M[AR] <- DR, if(DR=0) then (PC <- PC + 1),**
		SC <- 0

COMPLETE COMPUTER DESCRIPTION

Microoperations

Register-Reference		
	D7I'T3 = r	**(Common to all register-reference instr)**
	IR(i) = Bi	**(i = 0,1,2, ..., 11)**
	r:	**SC <- 0**
CLA	**rB11:**	**AC <- 0**
CLE	**rB10:**	**E <- 0**
CMA	**rB9:**	**AC <- AC'**
CME	**rB8:**	**E <- E'**
CIR	**rB7:**	**AC <- shr AC, AC(15) <- E, E <- AC(0)**
CIL	**rB6:**	**AC <- shl AC, AC(0) <- E, E <- AC(15)**
INC	**rB5:**	**AC <- AC + 1**
SPA	**rB4:**	**If(AC(15) =0) then (PC <- PC + 1)**
SNA	**rB3:**	**If(AC(15) =1) then (PC <- PC + 1)**
SZA	**rB2:**	**If(AC = 0) then (PC <- PC + 1)**
SZE	**rB1:**	**If(E=0) then (PC <- PC + 1)**
HLT	**rB0:**	**S <- 0**
Input-Output	**D7IT3 = p**	**(Common to all input-output instructions)**
	IR(i) = Bi	**(i = 6,7,8,9,10,11)**
	p:	**SC <- 0**
INP	**pB11:**	**AC(0-7) <- INPR, FGI <- 0**
OUT	**pB10:**	**OUTR <- AC(0-7), FGO <- 0**
SKI	**pB9:**	**If(FGI=1) then (PC <- PC + 1)**
SKO	**pB8:**	**If(FGO=1) then (PC <- PC + 1)**
ION	**pB7:**	**IEN <- 1**
IOF	**pB6:**	**IEN <- 0**

DESIGN OF BASIC COMPUTER(BC)

Hardware Components of BC

A memory unit: 4096 x 16.
Registers:
AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
Flip-Flops(Status):
I, S, E, R, IEN, FGI, and FGO
Decoders: a 3x8 Opcode decoder
a 4x16 timing decoder
Common bus: 16 bits
Control logic gates
Adder and Logic circuit: Connected to AC

Control Logic Gates

- **Input Controls of the nine registers**
- **Read and Write Controls of memory**
- **Set, Clear, or Complement Controls of the flip-flops**
- **S2, S1, S0 Controls to select a register for the bus**
- **AC, and Adder and Logic circuit**

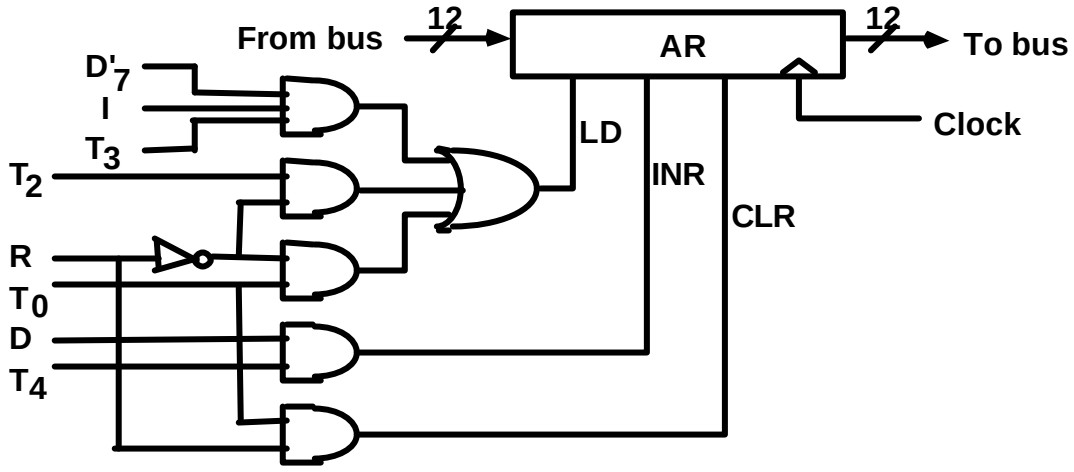
CONTROL OF REGISTERS AND MEMORY

Address Register; AR

Scan all of the register transfer statements that change the content of AR:

$R'T_0$:	AR ← PC	LD(AR)
$R'T_2$:	AR ← IR(0-11)	LD(AR)
D'_7IT_3:	AR ← M[AR]	LD(AR)
RT_0:	AR ← 0	CLR(AR)
D_5T_4:	AR ← AR + 1	INR(AR)

$$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$

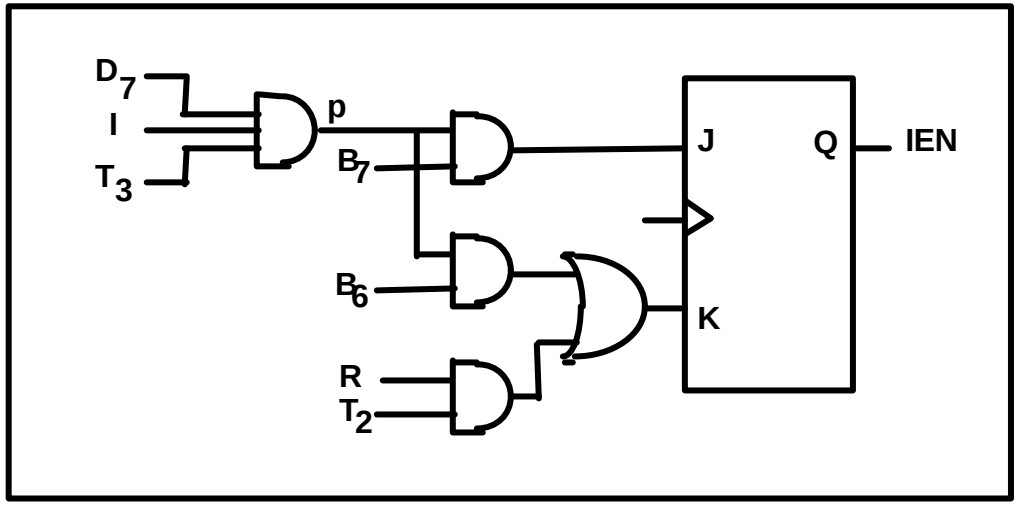


CONTROL OF FLAGS

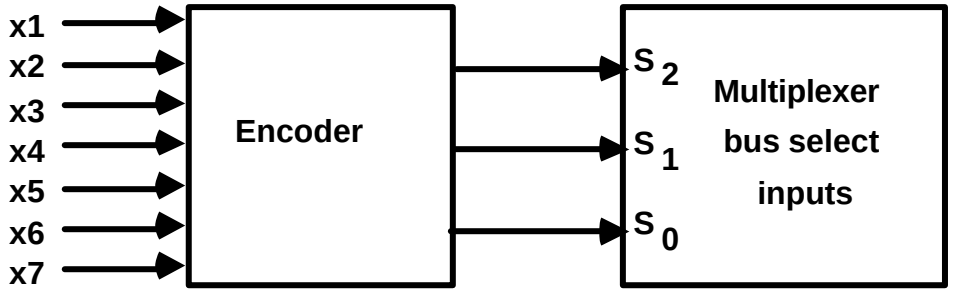
IEN: Interrupt Enable Flag

pB7: IEN ¬ 1 (I/O Instruction)
pB6: IEN ¬ 0 (I/O Instruction)
RT_2: IEN ¬ 0 (Interrupt)

$p = D_7IT_3$ (Input/Output Instruction)



CONTROL OF COMMON BUS



x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	selected register
0	0	0	0	0	0	0	0	0	0	none
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

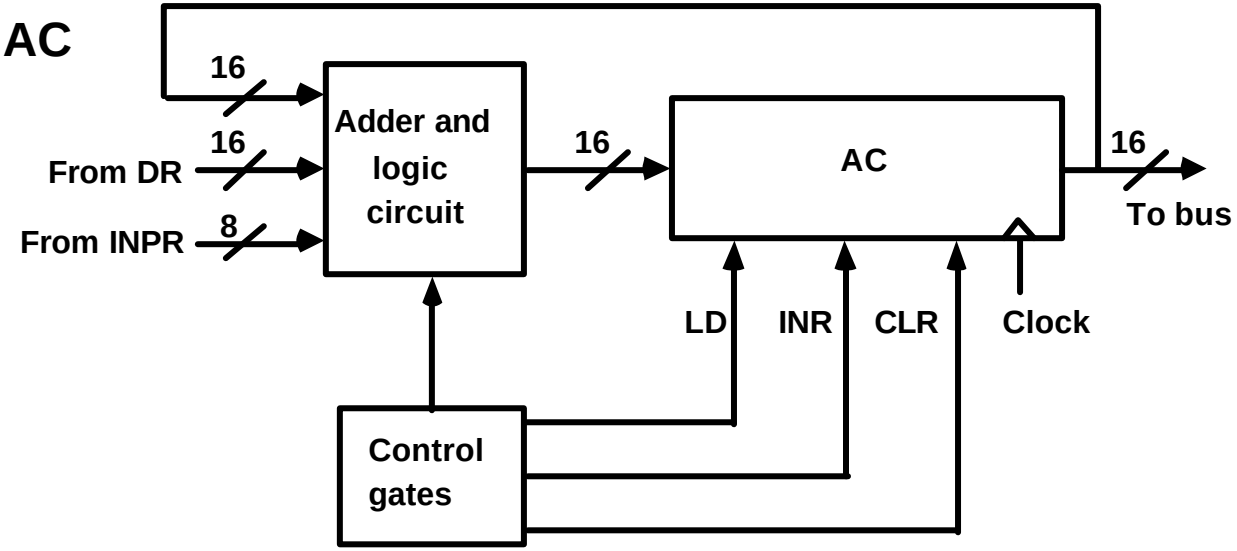
For AR

D_4T_4: PC ¬ AR
D_5T_5: PC ¬ AR

$x1 = D_4T_4 + D_5T_5$

DESIGN OF ACCUMULATOR LOGIC

Circuits associated with AC

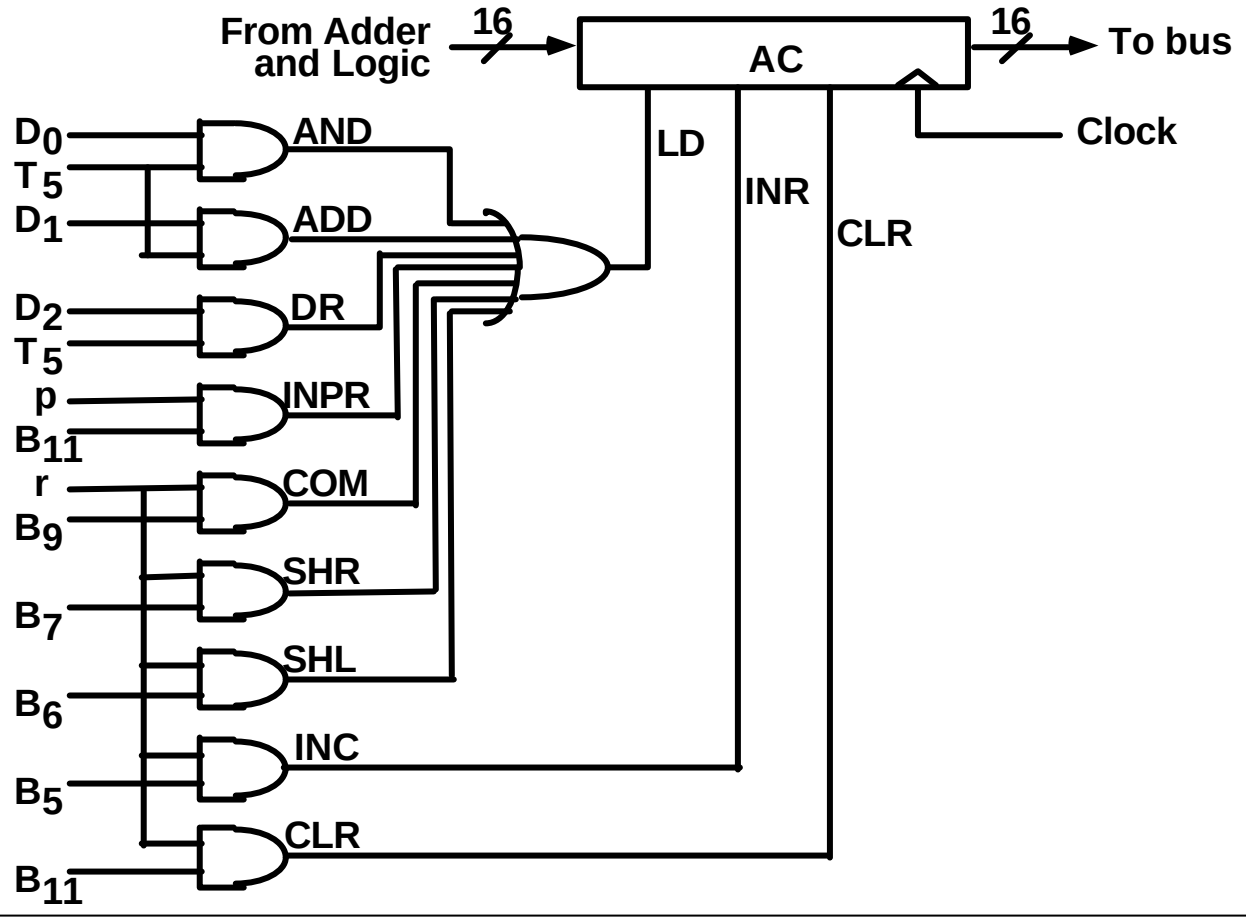


All the statements that change the content of AC

D_0T_5:	AC ¬ AC Ù DR	AND with DR
D_1T_5:	AC ¬ AC + DR	Add with DR
D_2T_5:	AC ¬ DR	Transfer from DR
pB_{11}:	AC(0-7) ¬ INPR	Transfer from INPR
rB_9:	AC ¬ AC'	Complement
rB_7 :	AC ¬ shr AC, AC(15) ¬ E	Shift right
rB_6 :	AC ¬ shl AC, AC(0) ¬ E	Shift left
rB_{11} :	AC ¬ 0	Clear
rB_5 :	AC ¬ AC + 1	Increment

CONTROL OF AC REGISTER

Gate structures for controlling the LD, INR, and CLR of AC



ADDER AND LOGIC CIRCUIT

One stage of Adder and Logic circuit