Computer System Architecture(5장)

Hanbat National University Prof. Lee Jaeheung

기본 컴퓨터의 구성과 설계

- 명령어 코드
- 컴퓨터 레지스터
- 컴퓨터 명령어
- 타이밍과 제어
- 명령어 사이클
- 메모리 참조 명령어
- 입출력과 인터럽트
- 컴퓨터에 대한 완전한 기술
- 기본 컴퓨터 설계
- 누산기 논리의 설계

INTRODUCTION

- 다른 여러 프로세서 타입은 하나의 설계를 가진다.

(different registers, buses, microoperations, machine instructions, etc)

- 현대의 프로세서는 매우 복잡한 장치 이다.

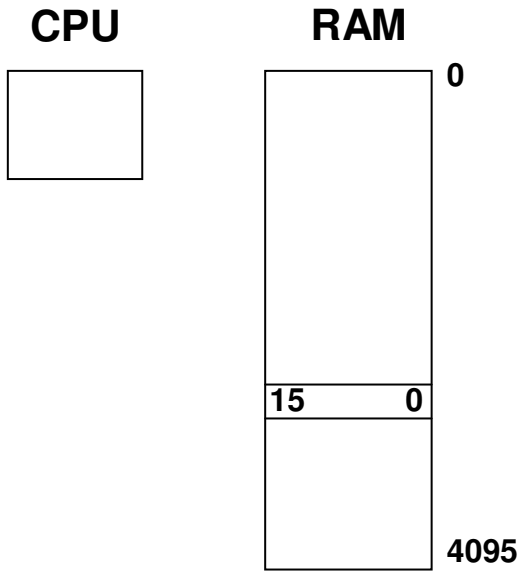
  - 구성 요소 -

  - 많은 레지스터.
  - 정수와 소수점 연산을 위해 많은 산술연산으로 구성되어 있다.
  - 연속적 명령을 빠르게 수행할수 있는 Pipeline 이점을 가진다.
  - Etc.

- 프로세서가 어떻게 작업을 처리하는지 이해하고 간략화된 프로세스 모델로 시작할 것 이다.
- 25년전부터 실제 프로세서들은 유사하다.
- M. Morris Mano는 간단한 프로세스 모델을 소개하며 Basic Computer 라 명했다.
- Processor의 구성 및 RTL Model과 higher level computer의 구성에 대해 소개 하겠다.

THE BASIC COMPUTER

- Basic Computer는 processor와 memorry 두개의 Components를 가진다.
- Memory는 4096 word를 가진다.
  - 4096 = 2^{12}, 그래서 메모리에서 12 bits에의해 word가 선택된다.
- 각각의 word는 16 bits 로 되어 있다.



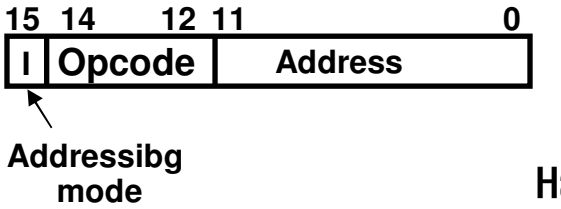
INSTRUCTIONS

- 프로그램
  - A sequence of (machine) instructions - 명령어 순서
- (Machine) Instruction
  - 컴퓨터에서 수행되는 Specific operation을 말하는 bit의 집합.
- 프로그램의 명령은 필요한 데이터와 함께 memory 에 저장된다.
- CPU 는 memory로부터 다음 명령어를 읽어온다.
- *Instruction Register* (IR)에 저장된다.
- Control unit의 Control circuitry는 microoperation의 sequence속의 명령을 해석하는데 필요한도구이다.

명령어 형식

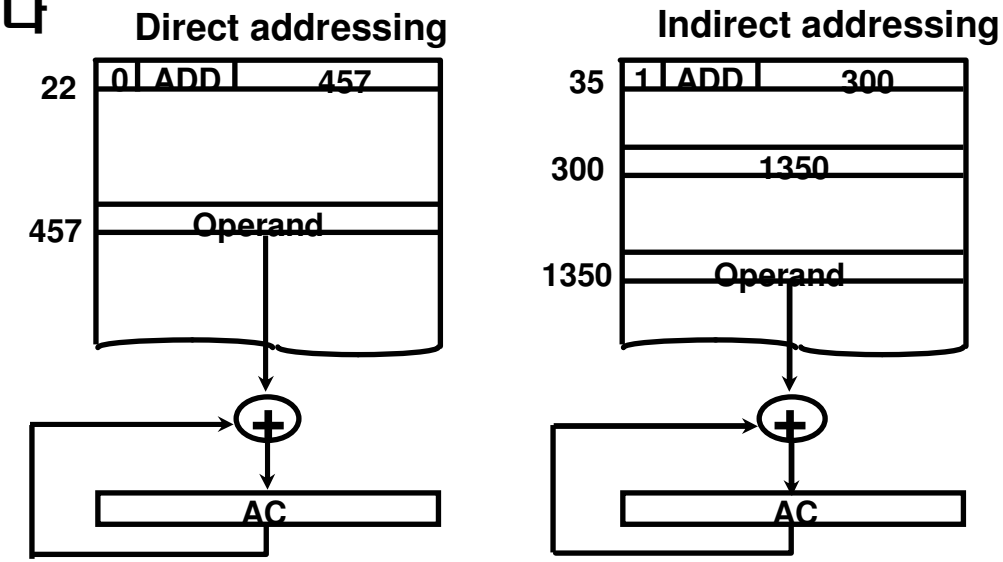
- 컴퓨터 명령어는 종종 두부분 으로 나뉜다.
  - Operation Code (opcode)는 연산에 대한 정의이다.
  - 연산하기 위해 사용되는 피연산(and/or)이 저장된 메모리 내의 주소.
- 4096 (= 2^{12}) 워드를 가진 기억장치에 대해서 12비트의 주소가 필요하다.
- 기본 컴퓨터에서는, 15번째 bit 명령은 주소 참조 (addressing mode)모드를 나타낸다.
  (0: 직접 주소 참조, 1: 간접 주소 참조)
- 메모리 워드단위 명령어 16bit 중의 3 bits 는 명령을 나타내는 opcode이다.

Instruction Format



ADDRESSING MODES

- 주소 필드는 서로 다르게 표현 된다.
  - 직접 주소 : 메모리 주소에 데이터 연산에 사용되는 내용이 들어 있다.
  - 간접 주소 : 메모리 주소에 데이터 연산에 사용되는 내용이 들어 있는 주소가 들어 있다



유효 주소 - Effective Address (EA)

- 계산형 명령어의 직접적인 피연산자의 주소.
- 혹은, 분기형 명령어에서 변경된 목적 주소.

PROCESSOR REGISTERS

- 프로세서는 명령어들을 포함한 레지스터들을 가지고 있다.

  (addresses, data, etc)

- 프로세서는 다음 연산의 주소를 가르키는 *Program Counter* (PC)레지스터를 가지고 있다.
  - 4096의 저장소를 가르키는 컴퓨터에서 PC는 12 bits가 필요하다.
- 간접주소에 접근하기 위해서는, 프로세서는 메모리 안에 저장되어 있는 주소를 통해 접근하여야 한다. : *Address Register* (AR)
  - 기본 컴퓨터에서는 AR은 12 bit 레지스터로 되어 있다.
- 명령어는 직접주소나 간접 주소의 내용을 참조하는것이 아니라 *Data Register* (DR)의 내용을 참조 한다. 명령어에 관련된 값이 저장되어 있다.
- 기본 컴퓨터는 일반적인 목적의 하나의 레지스터를 수행한다.

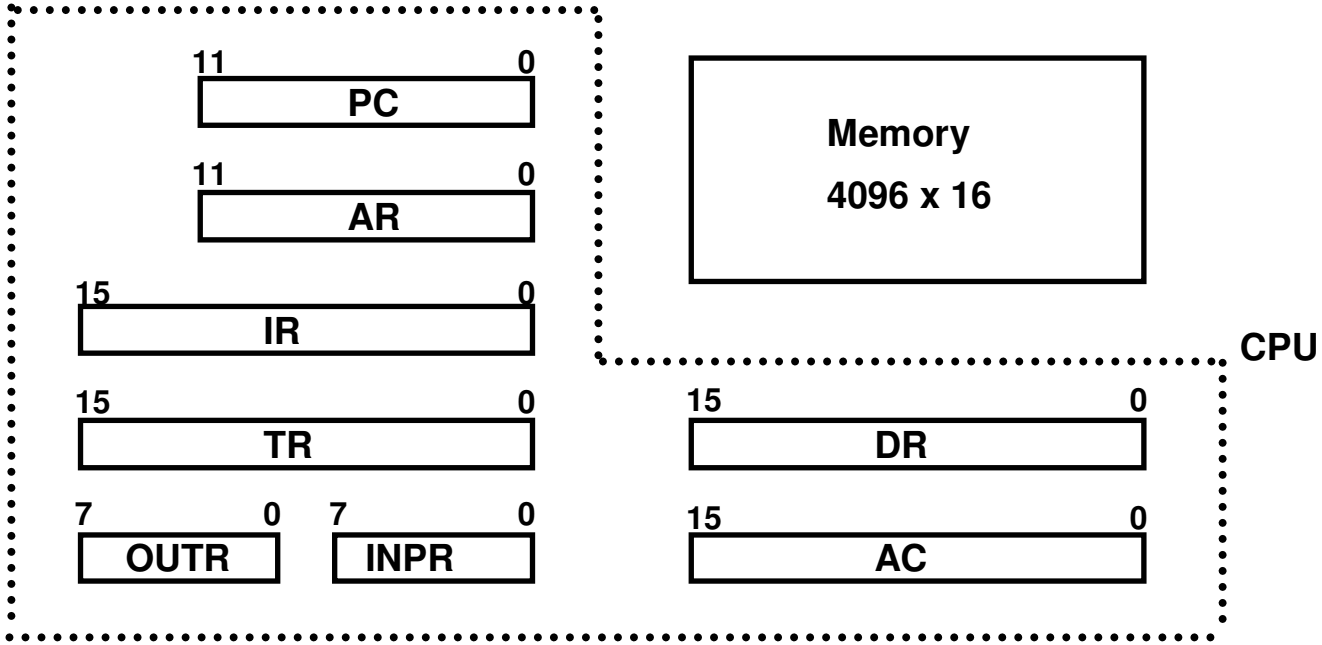
  – the *Accumulator* (AC)

PROCESSOR REGISTERS

- 일반적 목적 레지스터의 중요성은 명령어와 관련이 있다.
  - 예로 특수한 메모리 저장소 조건에 만족하면 AC가 로드된다.
    ; AC에 메모리저장소에 조건이 명시되어 저장된다.
- 종종 처리기는 즉시 중간결과를 저장하거나 일시적인 데이더를 저장해 놓을 수가 있다.
  ;기본 컴퓨터에서는 *Temporary Register* (TR)라 한다.
- 기본 컴퓨터는 매우 간단한 input/output (I/O) 연산을 한다.
  - 입력 장치는 character data처리를 위하여 8비트를 받아 들인다.
  - 처리기는 출력장치에 8비트 데이터를 보낼 수 있다.
- *Input Register* (INPR)에서는 입력 장치로부터 8 bit character를 받는다.
- *Output Register* (OUTR)에서는 출력 장치로 8 bit character를 보낸다.

BASIC COMPUTER REGISTERS

Registers in the Basic Computer



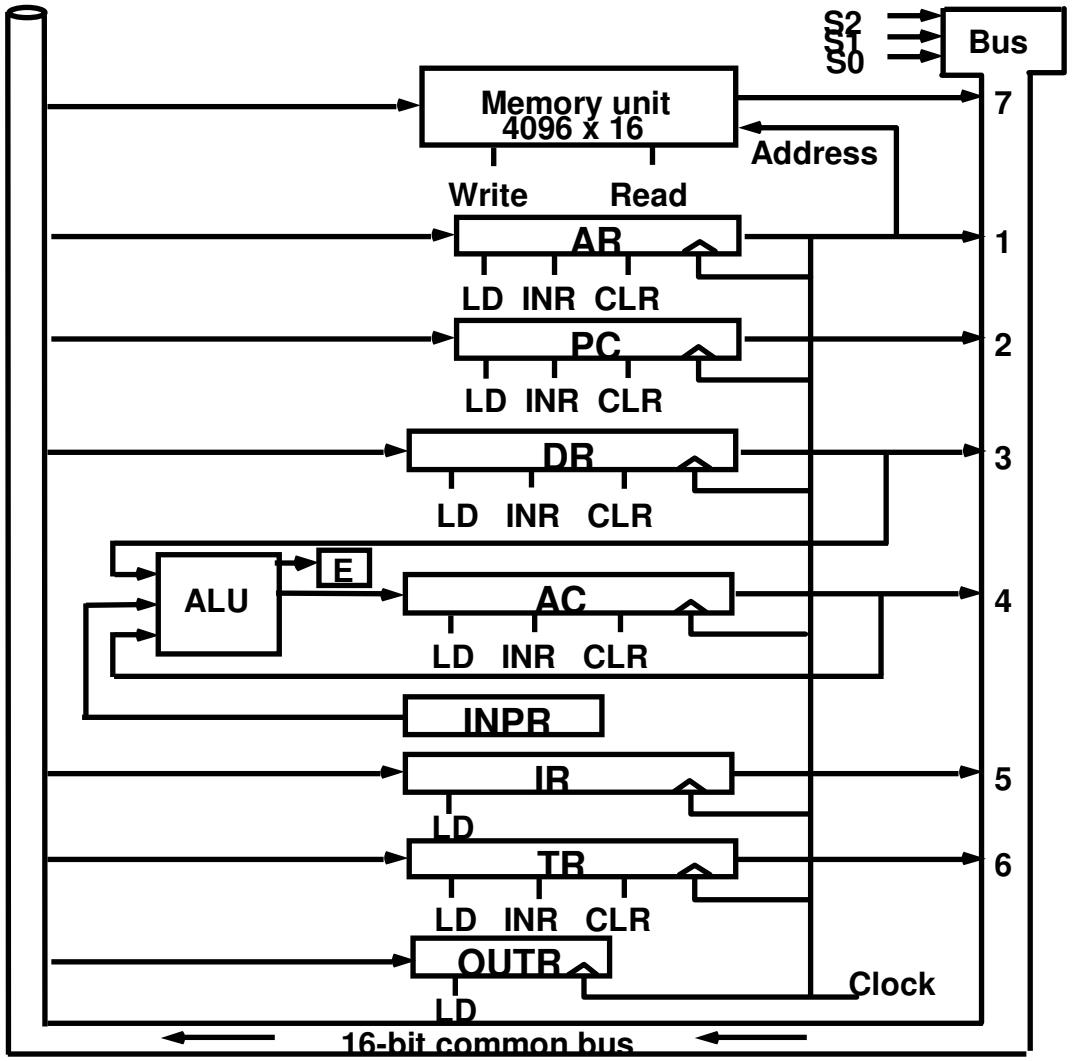
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

List of BC Registers

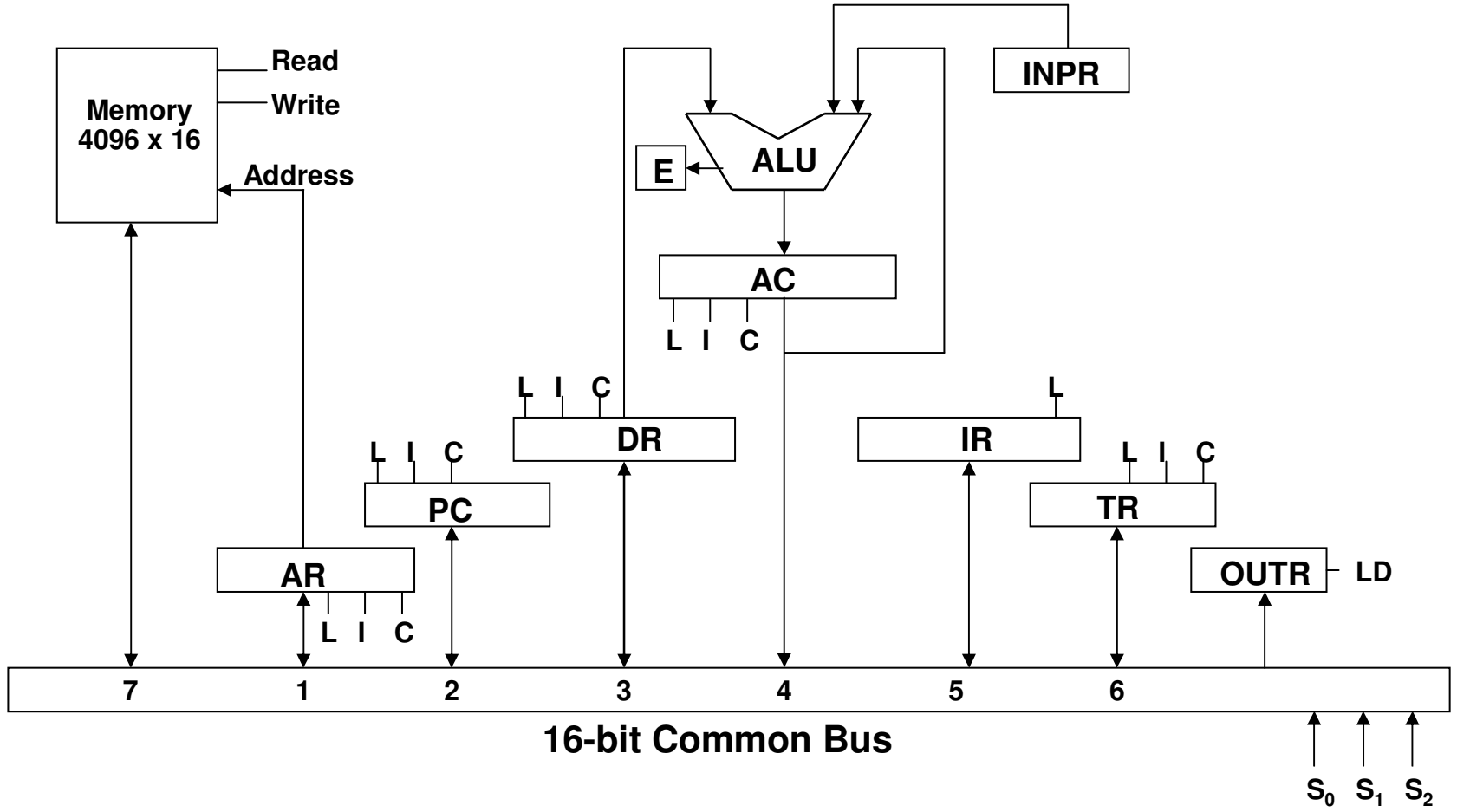
COMMON BUS SYSTEM

- 기본 컴퓨터에서는 레지스터들이 버스에 연결 되어 사용된다.
- 레지스터들끼리 연결된 회로소자 안에서 저장된다.

COMMON BUS SYSTEM



COMMON BUS SYSTEM



COMMON BUS SYSTEM

- 세개의 제어라인 S_2, S_1, S_0는 버스의 입력 신호를 제어한다.

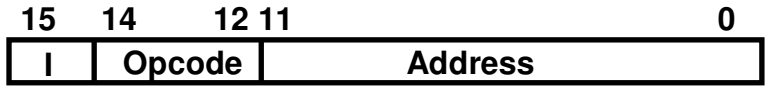
S_2 S_1 S_0	Register
0 0 0	x
0 0 1	AR
0 1 0	PC
0 1 1	DR
1 0 0	AC
1 0 1	IR
1 1 0	TR
1 1 1	Memory

- 다른 하나의 레지스터는 로드신호를 수행하거나 메모리에서 신호를 읽어온다.
  - 버스에 실어질 데이터를 정한다.
- 12비트 레지스터 AR, PC는 0을 로드 하였을 때 버스 위에 상위 4비트 읽어온다.
- 8비트레지스터 OUTR가 버스로부터 로드 될 때 데이터는 버스의 하위 8비트를 읽어다 쓴다.

BASIC COMPUTER INSTRUCTIONS

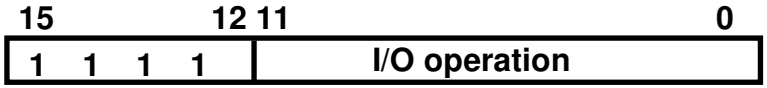
- 기본 컴퓨터 명령어 형식

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

❖ 컴퓨터는 사용자가 나름대로의 기계어로 된 프로그램을 만들 수 있도록 instruction set을 가지고 있다.

❖ 명령어 타입

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

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

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

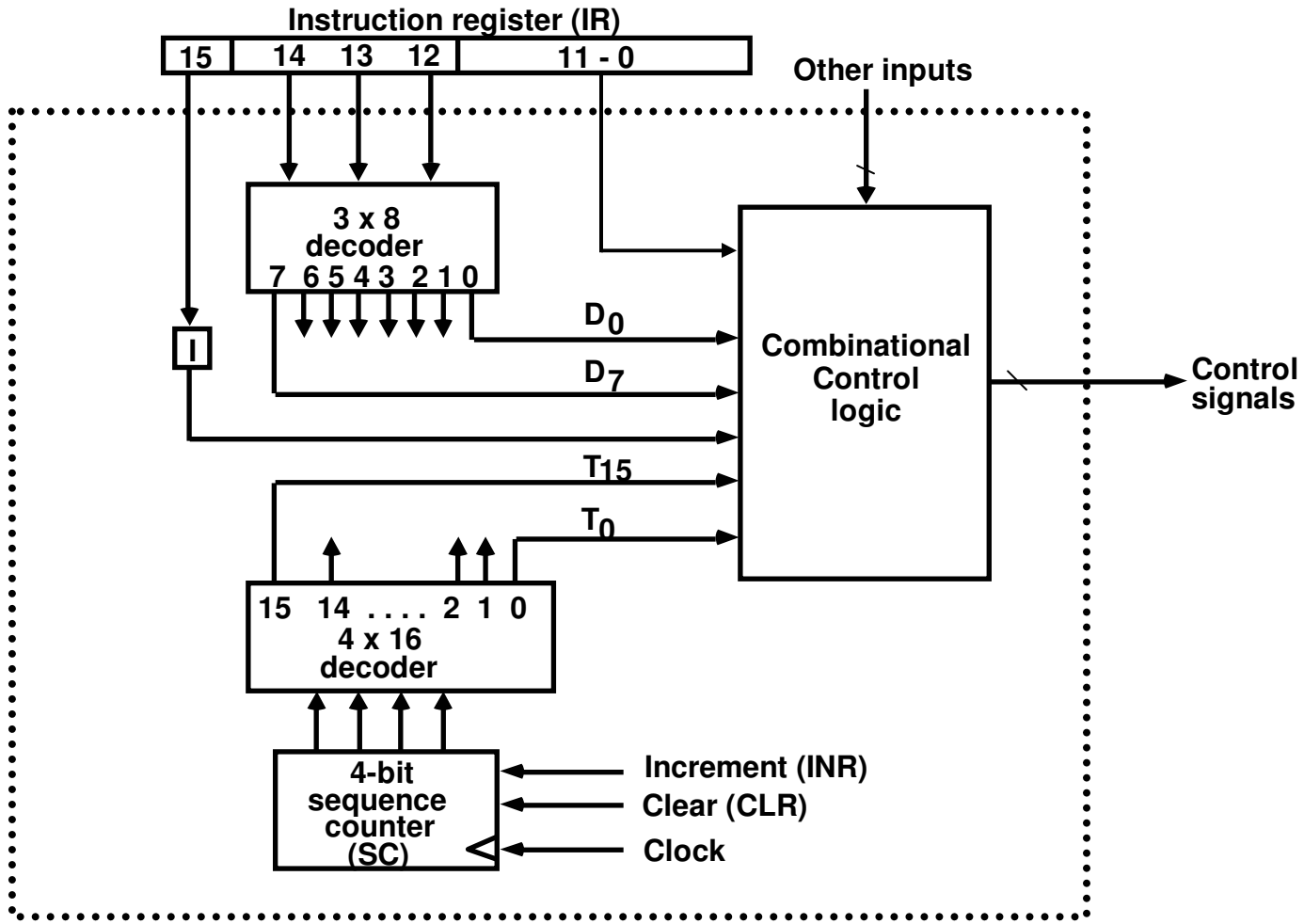
입출력 명령어
- **Input and output**
- **INP, OUT**

CONTROL UNIT

- 프로세서의 Control unit (CU)은 microoperations을 위해 머신 instruction을 컨트롤 신호로 바꾼다.
- Control unit은 두가지 방법으로 구현이 된다.
  - Hardwired Control
    - CU는 제어신호를 생성하기 위해서 sequential circuits과 combinational circuits으로 만들어진다.
  - Microprogrammed Control
    - 프로세서의 control memory는 꼭 필요한 제어신호를 활성화시키는 microprogram들을 포함한다.
- 우리는 기본 컴퓨터의 control unit의 hardwired적인 구현에 대하여 살펴볼 것이다.

TIMING AND CONTROL

Control unit of Basic Computer

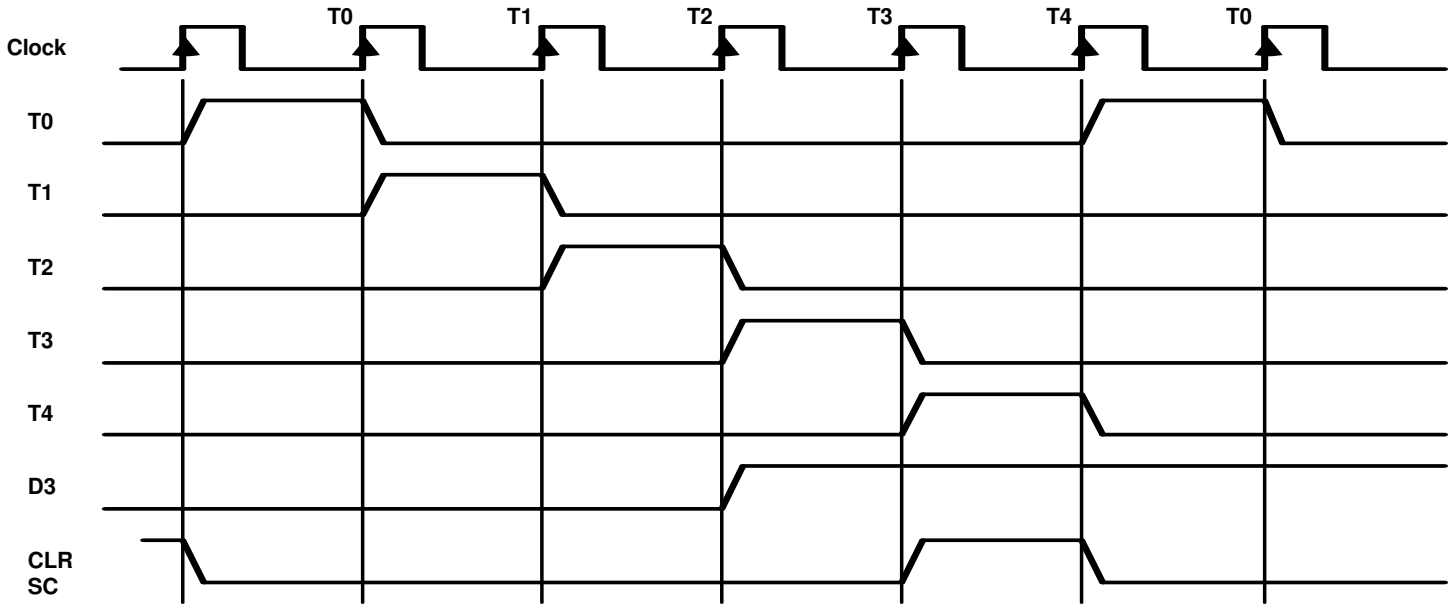


TIMING SIGNALS

- 4-bit 연속 카운터 와 4×16 decoder의 결과물
- SC가 증가 된다.

- 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

- **기본컴퓨터의 기계명령어는 다음의 사이클로써 수행된다.**
  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

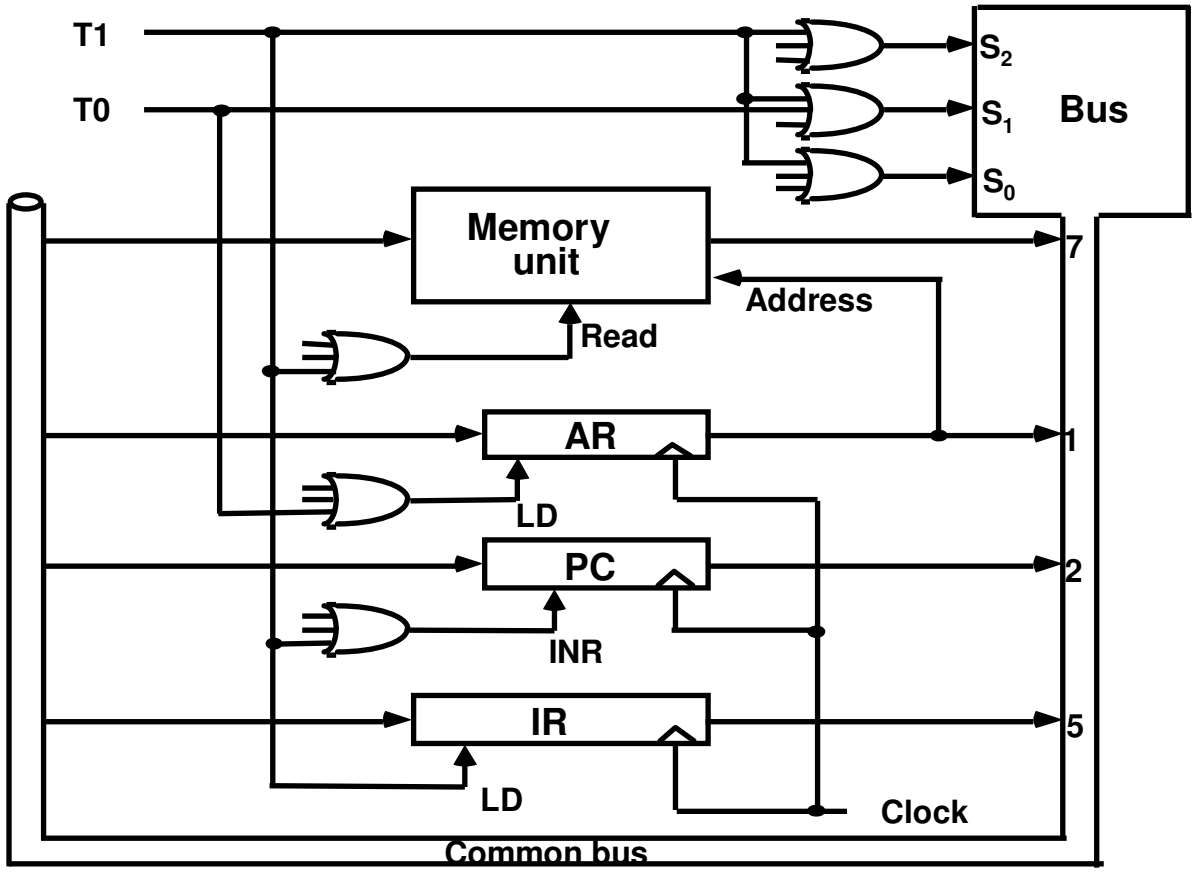
- **명령어 사이클은 1스텝씩 수행된다.**

- **주의 : 프로세서마다 다른 명령어 사이클을 가지고 있다.**

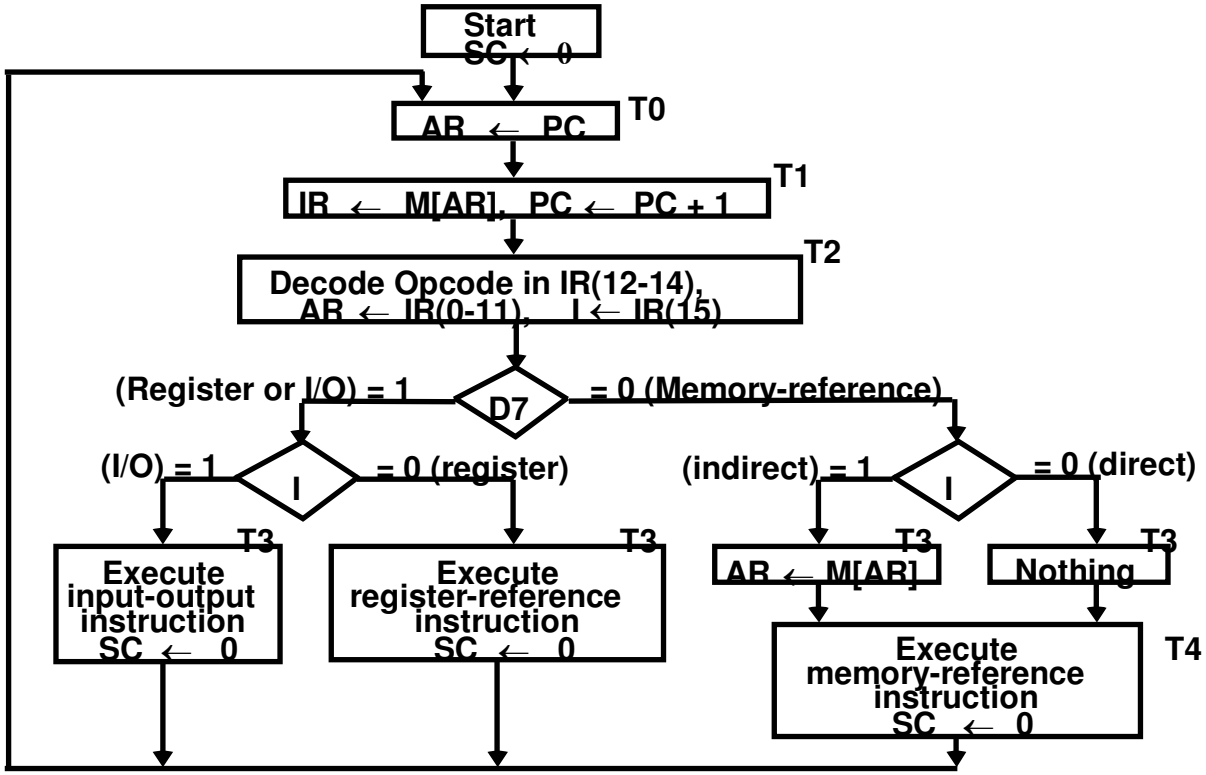
FETCH and DECODE

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=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: $AR \leftarrow M[AR]$
$D'_7I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instr.
D_7IT_3: Execute an input-output instr.

REGISTER REFERENCE INSTRUCTIONS

다음에서 레지스터 명령어를 확인할수 있다

- $D_7 = 1$, $I = 0$
- IR의 $b_0 \sim b_{11}$ 의 명령 조건을 참조한다.
- T_3 신호 에서 명령어가 수행된다.

$r = D_7 I' T_3$ => 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	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, if $M[AR] + 1 = 0$ then $PC \leftarrow PC+1$

- Instruction의 유효주소는 AR 를 통하여 이용된다.
  타임신호가 T_2 일때 I = 0, 타임신호가 T_3 일때 I = 1
- CPU 사이클 중에서 메모리 사이클은 간단하게 처리된다..
- MR Instruction 의 시작은 T와 같이 실행된다.$_4$

AND to AC

D_0T_4:	$DR \leftarrow M[AR]$	Read operand
D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$	AND with AC

ADD to AC

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

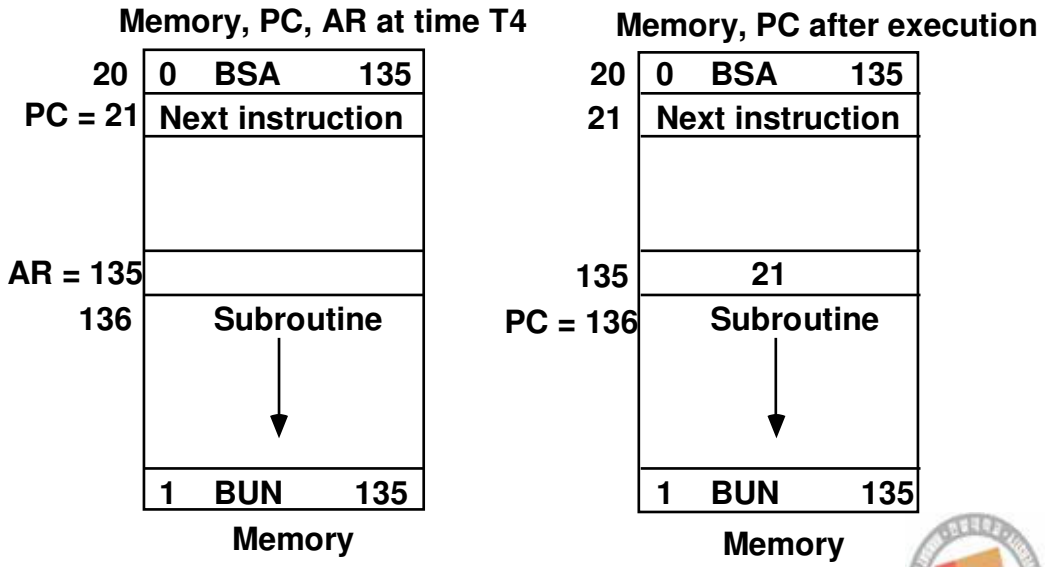
D_3T_4: M[AR] ← AC, SC ← 0

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

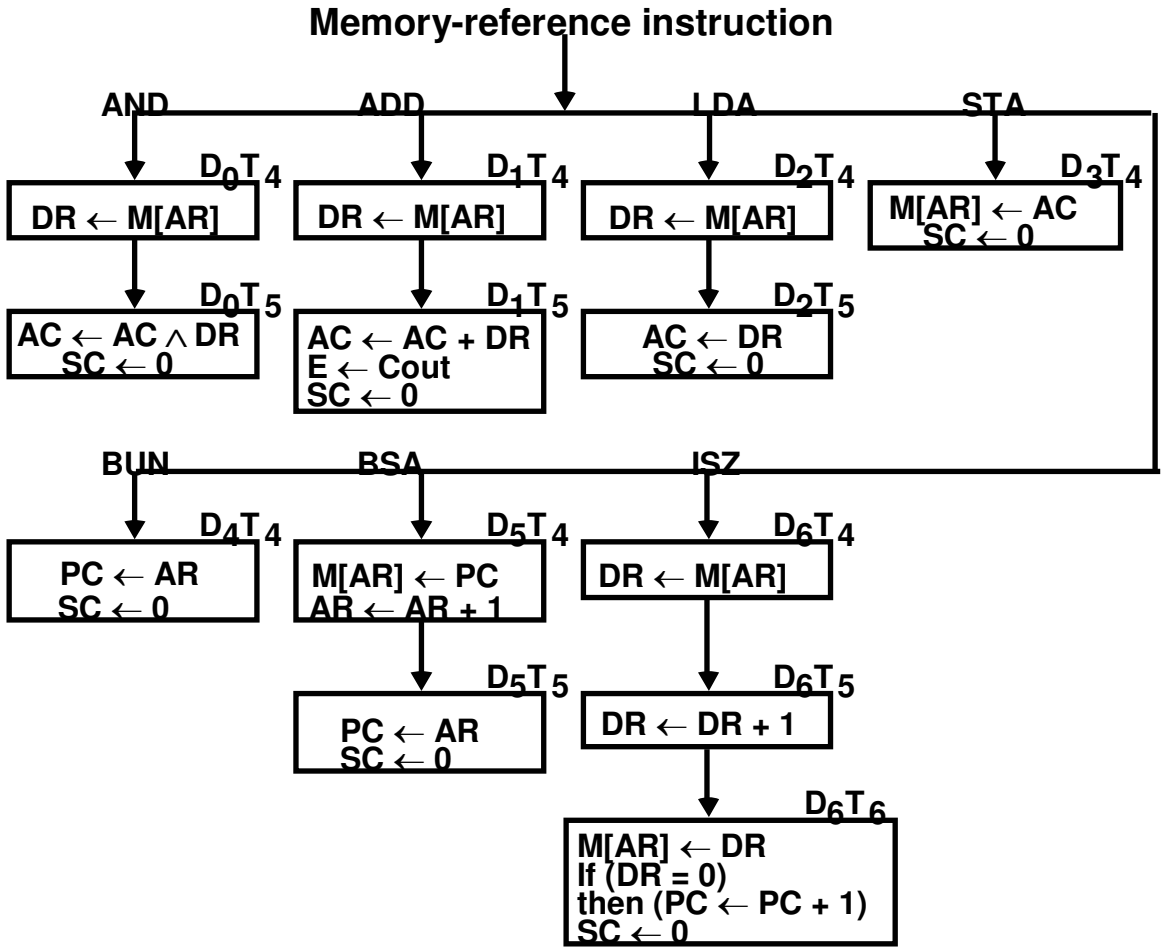
BSA:

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

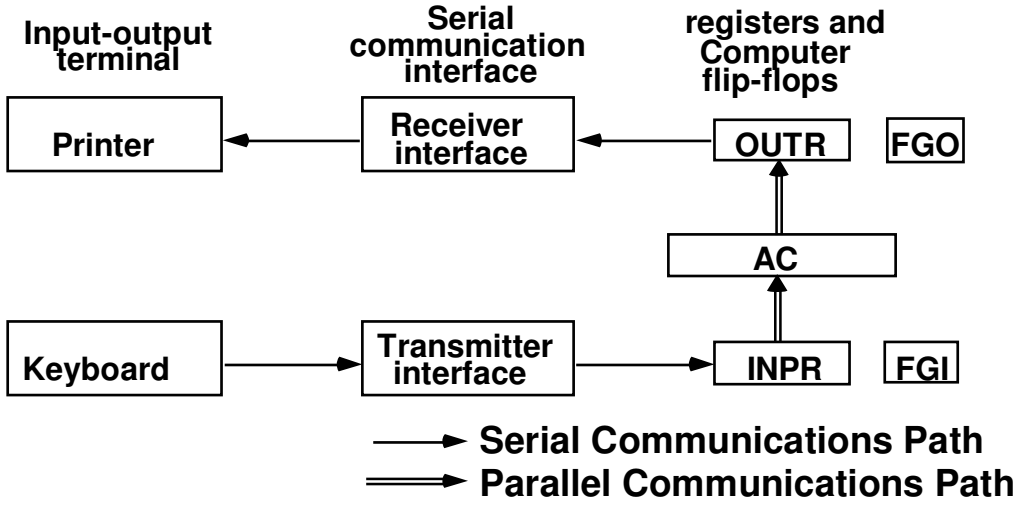
FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS



INPUT-OUTPUT AND INTERRUPT

A Terminal with a keyboard and a Printer

- Input-Output Configuration



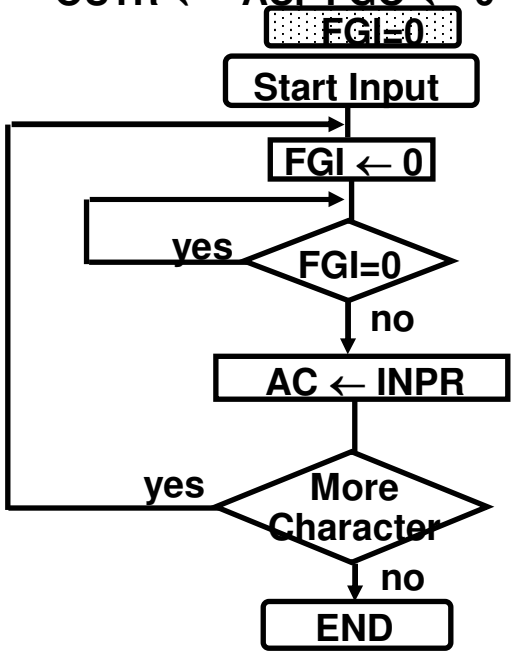
INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- 단말 장치는 연속 정보를 주고 받는다.
- The serial info. INPR안으로 키보드로 부터의 시프트된 정보.
- The serial info. OUTR안으로 프린터로 부터 저장된 정보.
- INPR 와 OUTR 는 병렬로 된AC 정보와 단말기의 직렬(연속)신호를 통신한다.
- Flags는 서로다른 I/O 장치와 컴퓨터 동기 신호를 필요로 한다.

PROGRAM CONTROLLED DATA TRANSFER

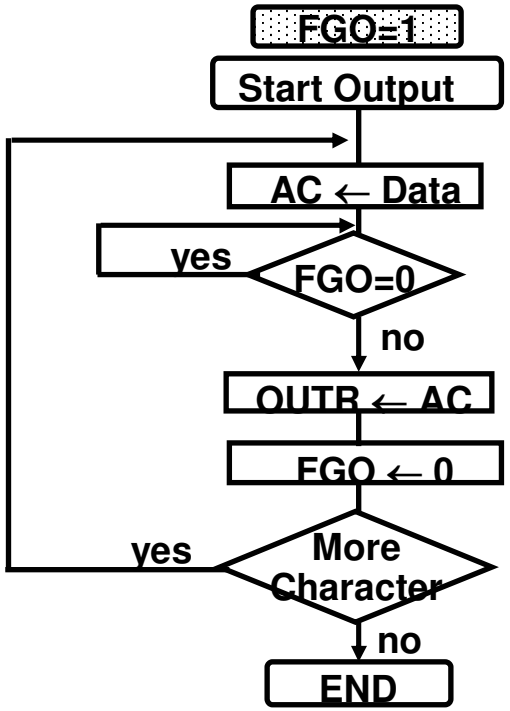
-- CPU --

```
/* 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

- 통신이 열려 있을때만, 데이터 입력이 허용된다. --> ***interrupt***.

- **I/O인터페이스가 아니라 CPU에서 I/O device를 모니터링한다.**

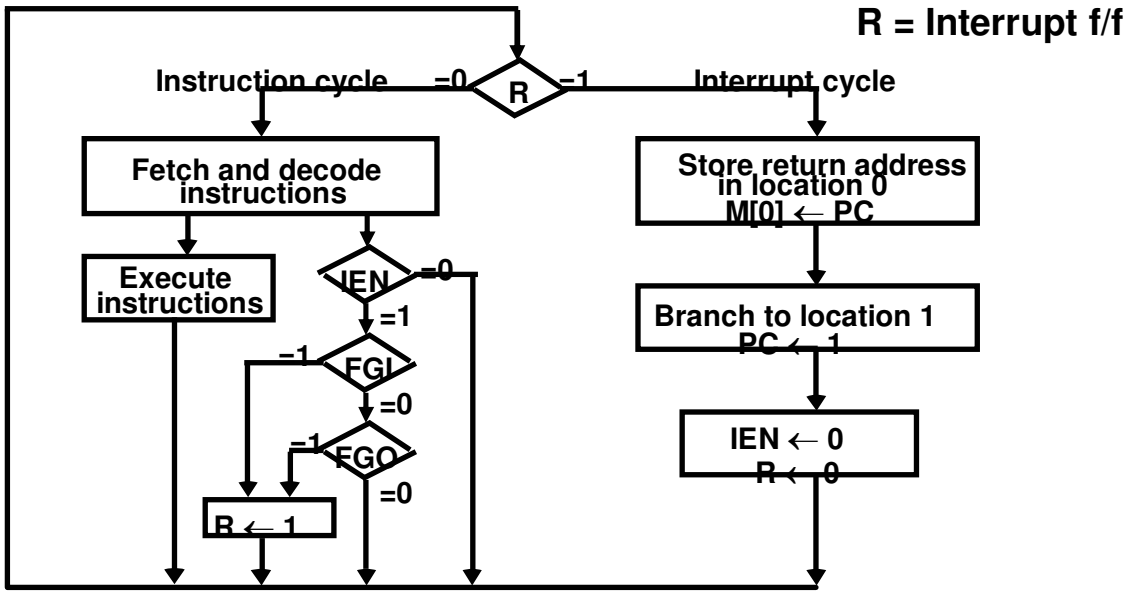
- 인터페이스를 찾을때 **I/O** 장치는 데이터 전송을 위한 준비로 **CPU**에게 **interrupt**요청 이있어야 한다.

-인터럽트 신호를 발견시 **CPU**는 하던일을 멈추고, **service routine**으로 데이터 전송을 위해 분기하고, 수행하던일을 리턴한다.

*** IEN (Interrupt-enable flip-flop)**

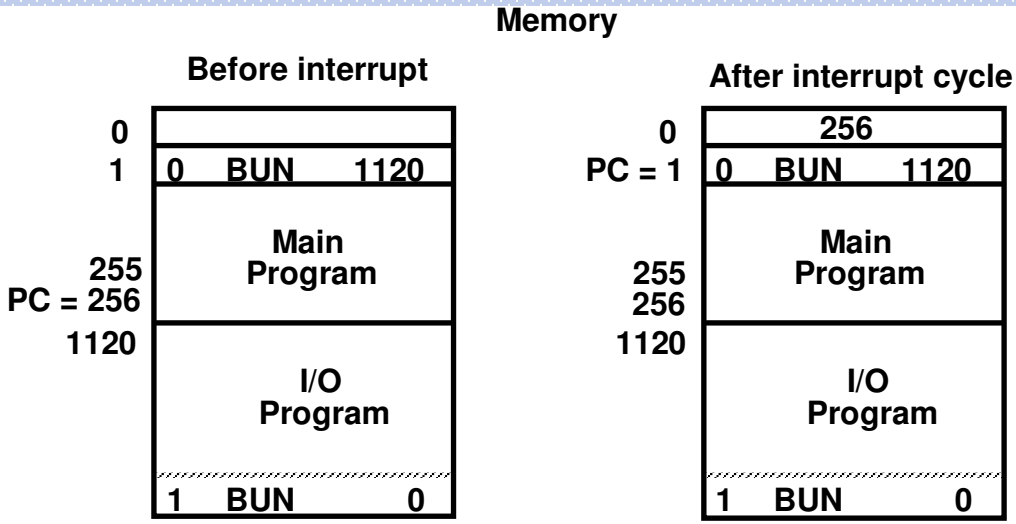
- **can be set and cleared by instructions**
- **when cleared, the computer cannot be interrupted**

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)$T_0'T_1'T_2'$

  ⇔ $T_0'T_1'T_2'$ (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

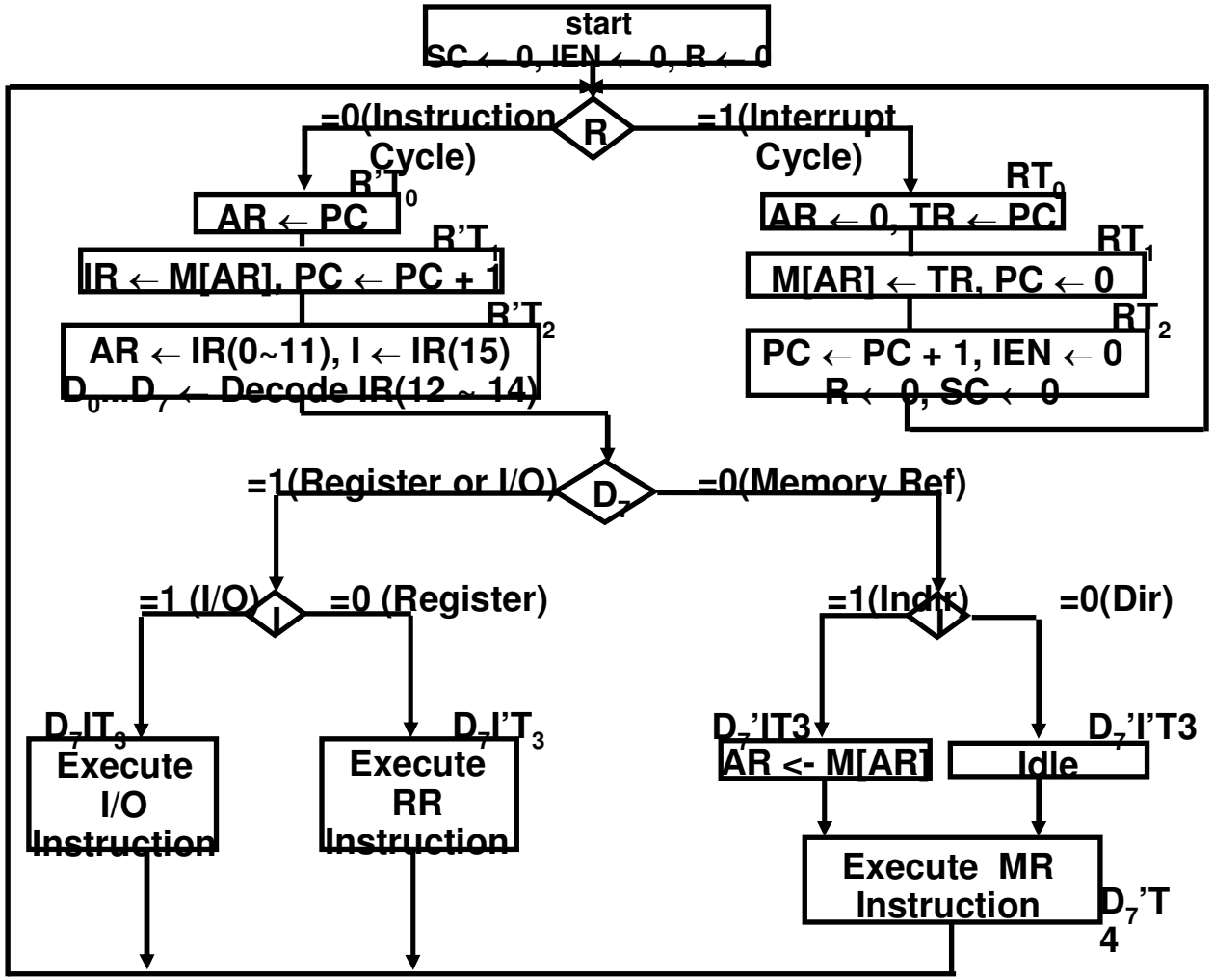
How can the CPU recognize the device requesting an interrupt ?

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**
- **S_2, S_1, S_0 Controls to select a register for the bus**
- **AC, and Adder and Logic circuit**

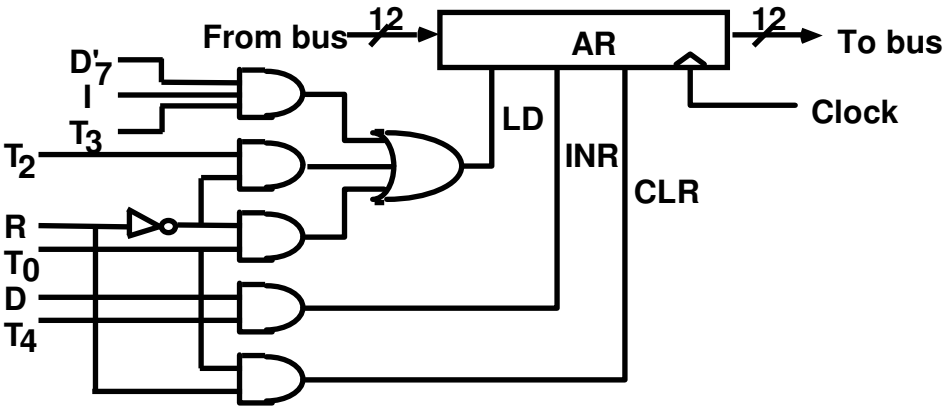
CONTROL OF REGISTERS AND MEMORY

Address Register; AR

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

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

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

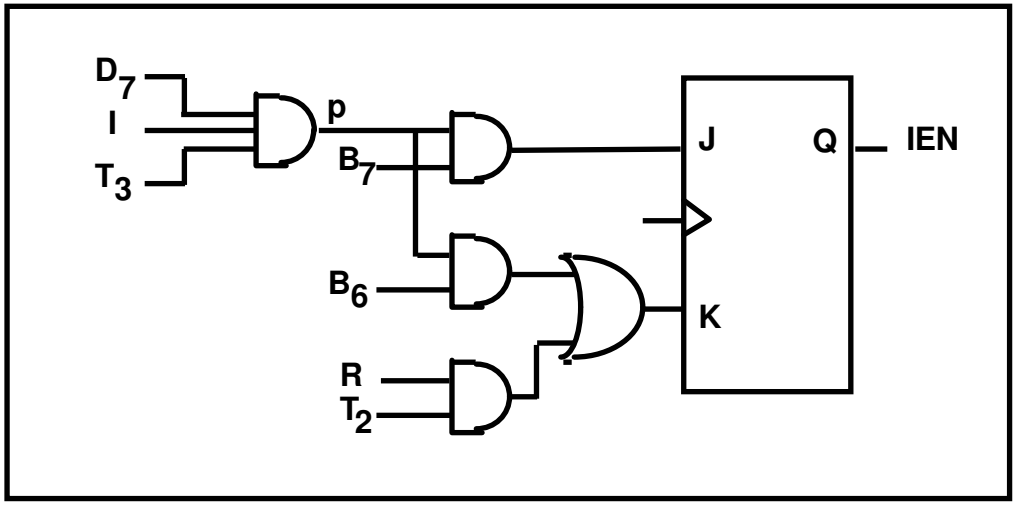


CONTROL OF FLAGS

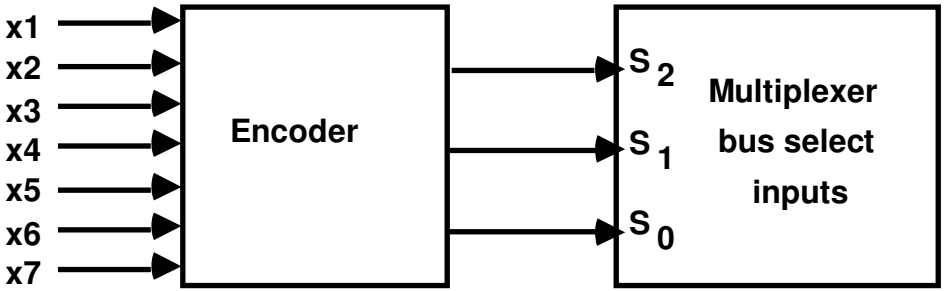
IEN: Interrupt Enable Flag

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

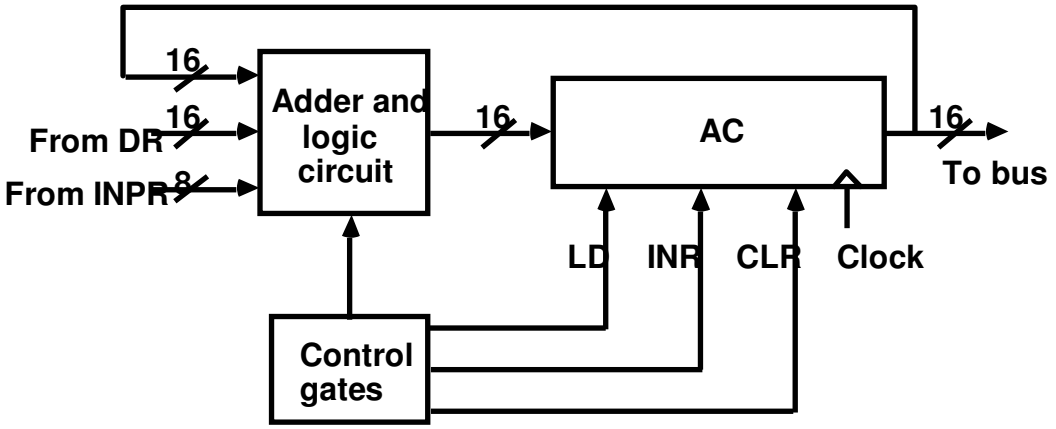
For AR

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

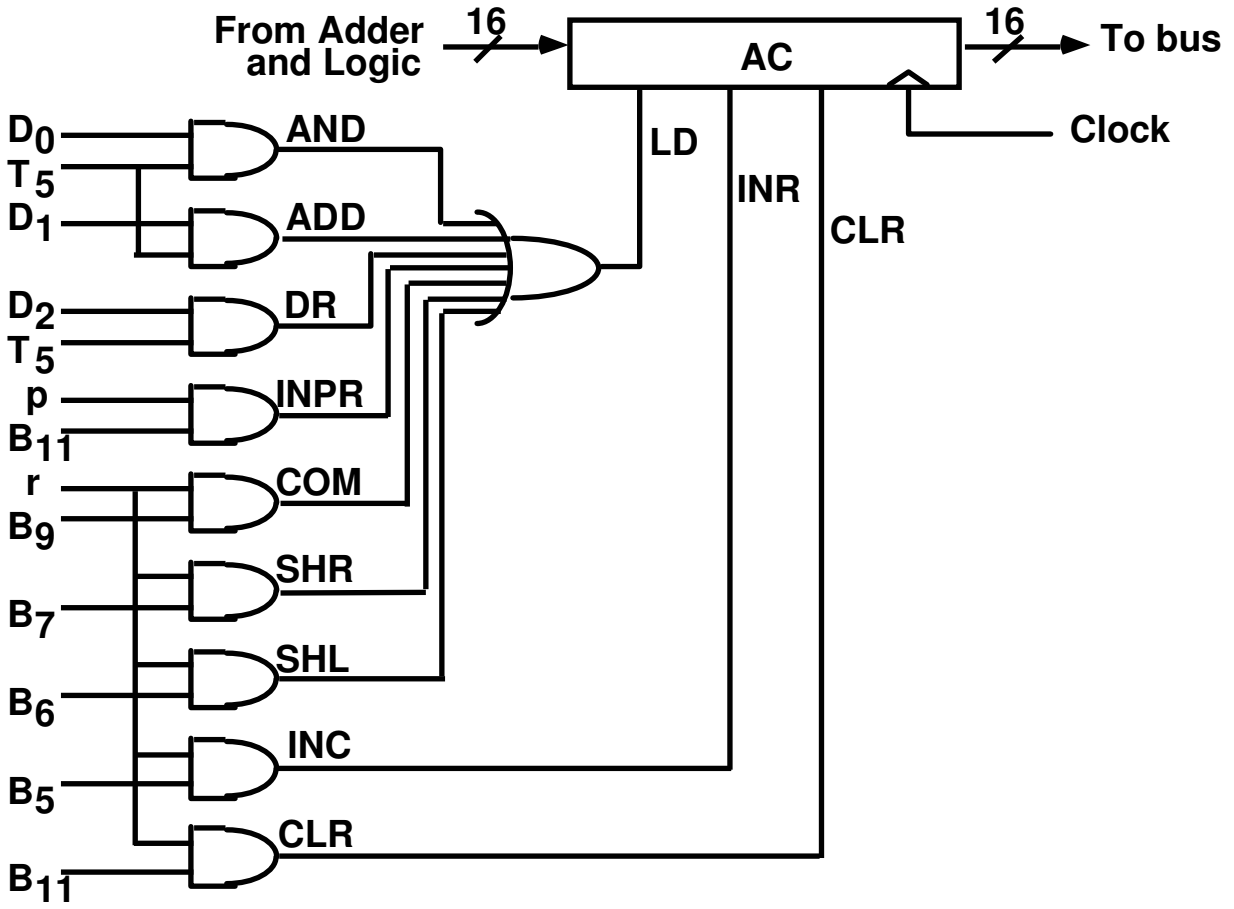


All the statements that change the content of AC

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

CONTROL OF AC REGISTER

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



ALU (ADDER AND LOGIC CIRCUIT)

One stage of Adder and Logic circuit