기본 컴퓨터의 구조와 설계

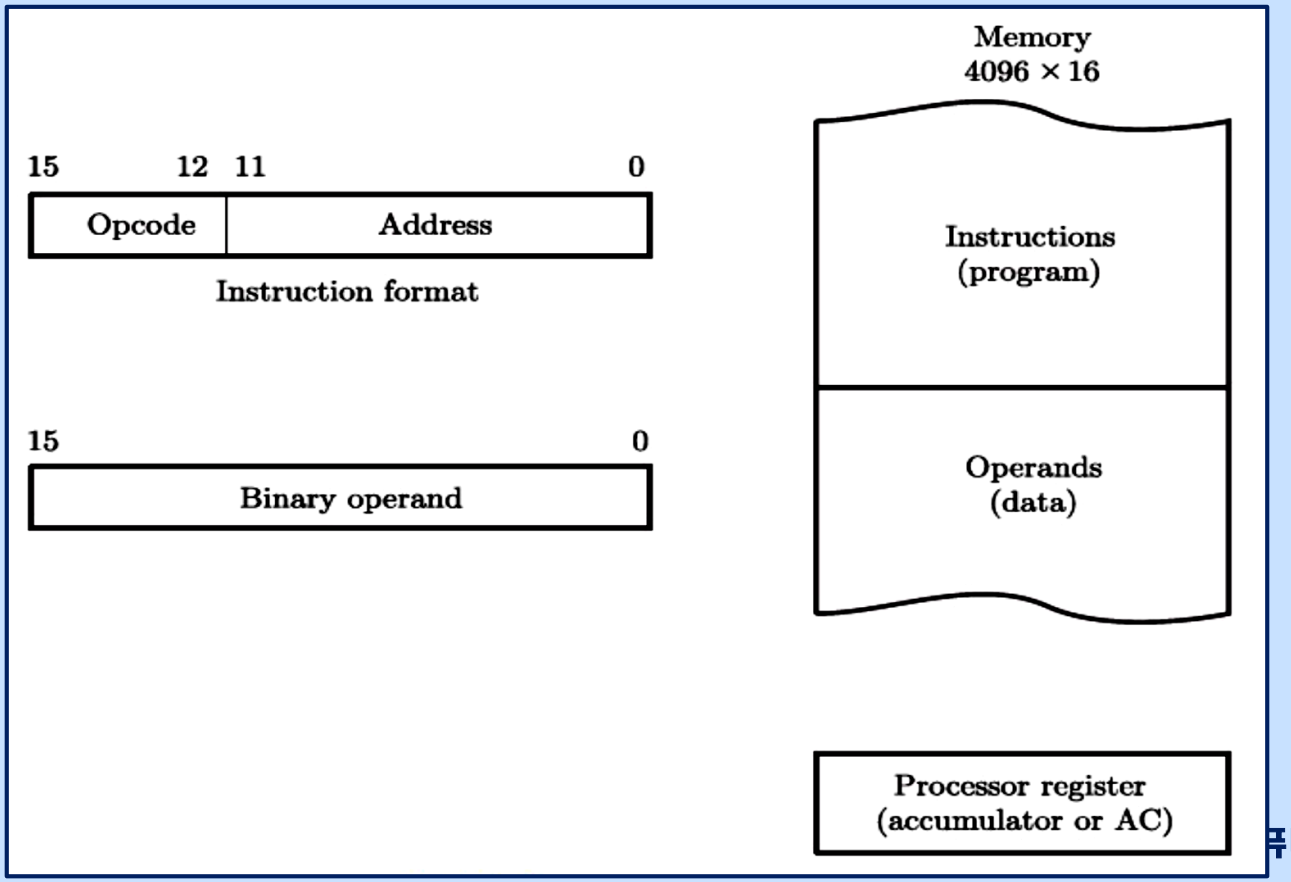
5.1 명령어 코드

◈ 디지털 시스템의 내부 조직은 레지스터 안에 저장된 데이터를 가지고 수행되는 마이크로 연산의 시퀀스에 의해서 정의됨

- 디지털 컴퓨터는 다양한 마이크로 연산을 실행할 수 있으며 수행할 연산의 특수한 시퀀스를 명령할 수 있음
- 컴퓨터 명령어는 컴퓨터에 대한 일련의 마이크로 연산을 기술한 이진 코드
- 명령어 코드의 연산 코드 부분은 컴퓨터가 실행하여야 하는 연산을 나타내며 제어 장치는 연산 코드의 비트 부분을 해석하여 일련의 마이크로 연산을 실행하는 제어 함수를 발생시킴
- 명령어 코드는 연산 뿐만 아니라 피연산자가 저장된 레지스터나 메모리 워드, 또한 연산 결과가 저장될 장소를 기술해야 함

◈ 저장 프로그램의 구조

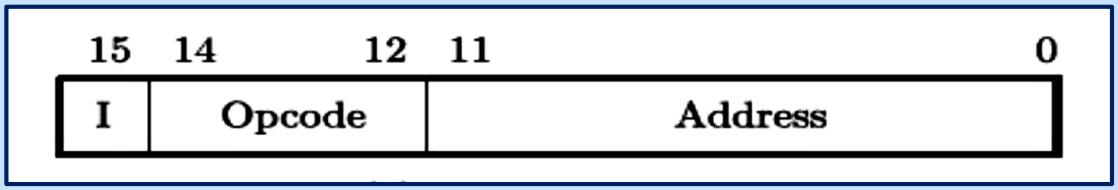
- 컴퓨터의 가장 간단한 구성은 하나의 프로세서 레지스터(AC)를 가지고 연산 부분과 메모리 주소 부분으로 구성된 명령어 코드를 사용하는 것
- 4096워드의 기억 장치는 12비트의 주소 비트가 필요, 연산 코드 4비트



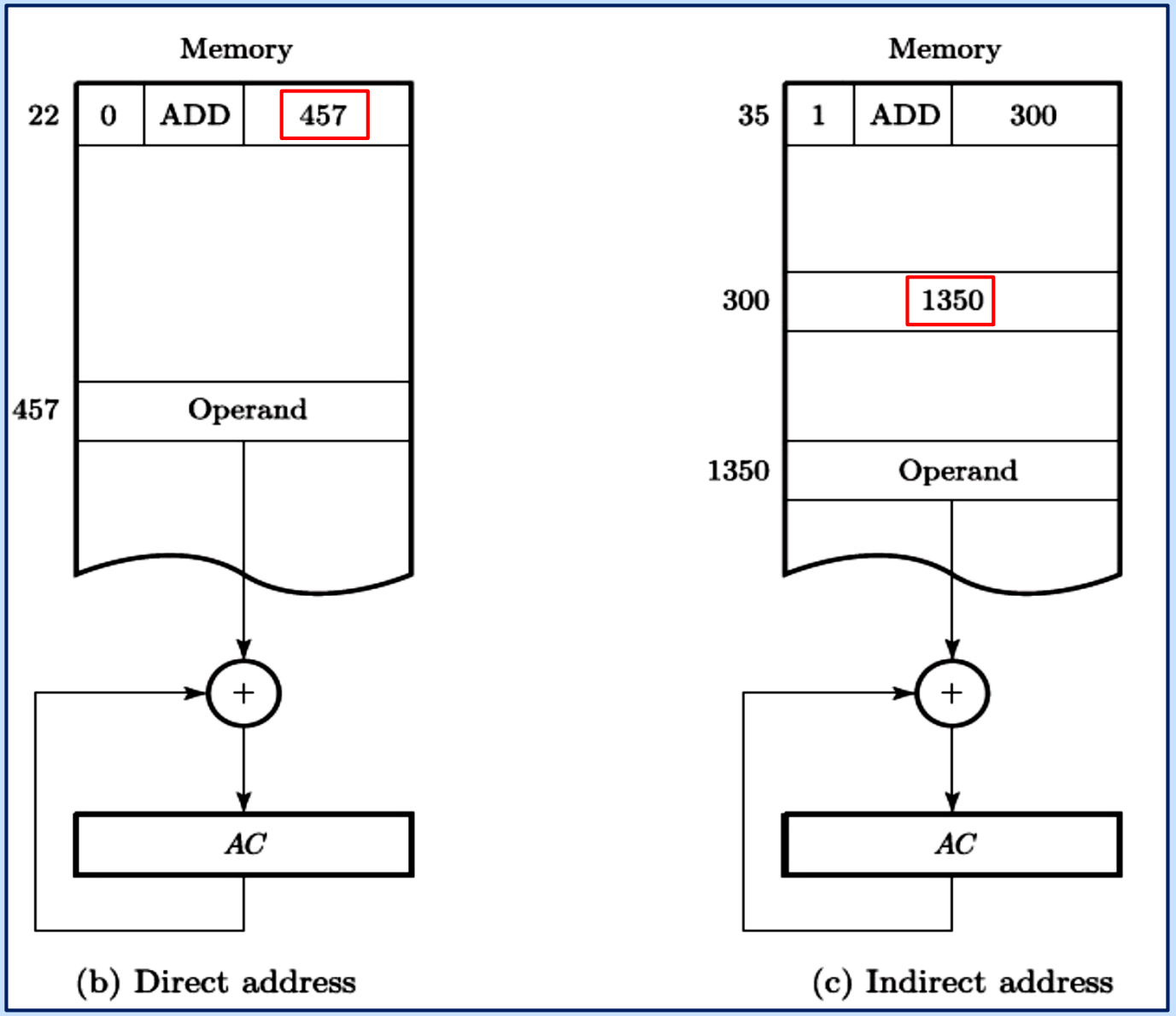
- AC 클리어, 보수, 인크리멘트는 피연산자 부분을 다른 목적으로 사용 가능

◈ 간접 주소

- 직접 주소 : 주소 부분이 피연산자(데이터)가 있는 메모리 주소를 나타냄
- 간접 주소 : 주소 부분이 피연산자(데이터)가 있는 주소가 저장되어 있는 메모리 주소를 나타냄
- 명령어 형식은 3비트의 연산 코드, 12비트의 주소, 간접 주소 모드 I 비트로 구성되며 I = 0이면 직접 주소, I = 1이면 간접 주소를 나타냄

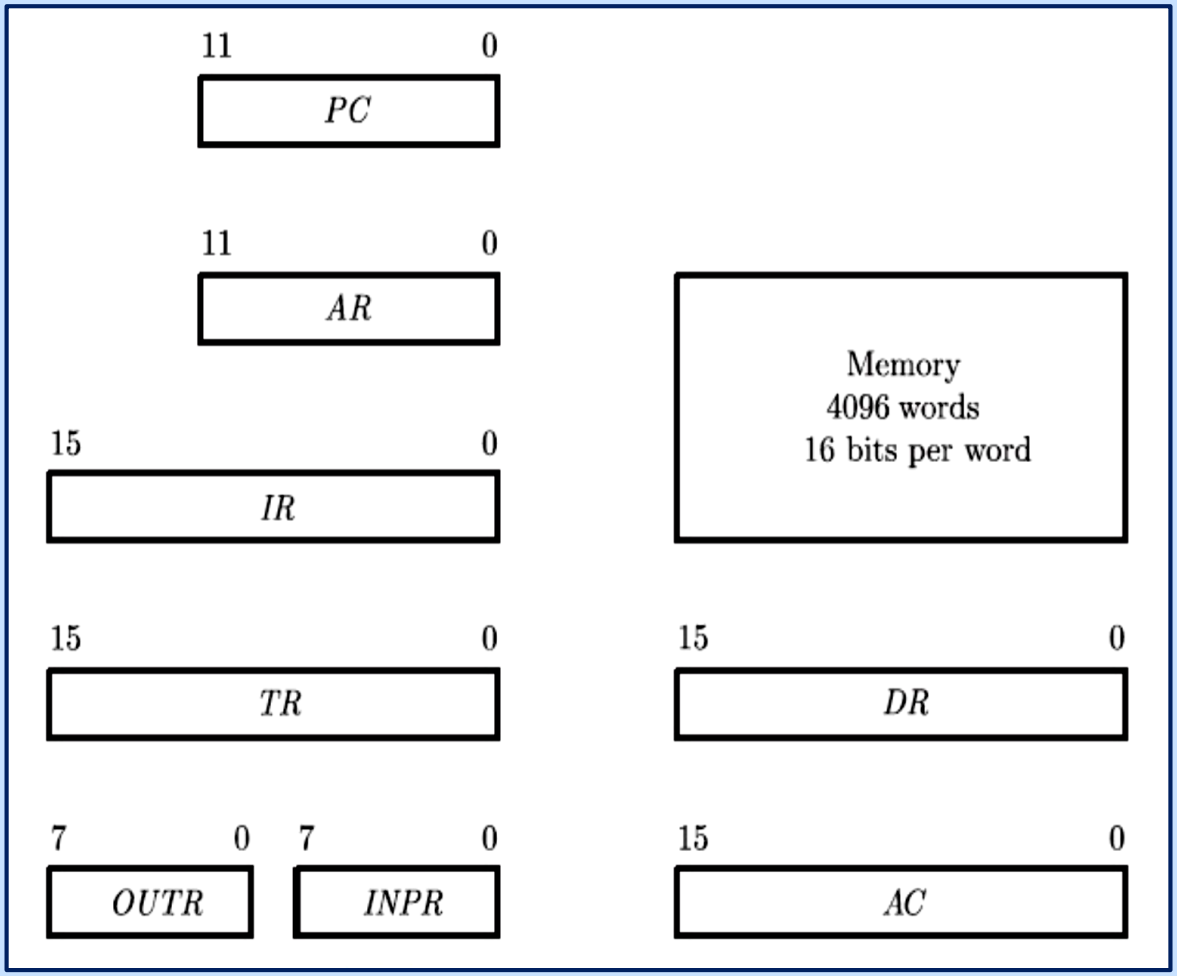


- 간접 주소는 피연산자를 얻기 위해 두 번의 메모리 참조가 필요
- 계산형 명령어에서 피연산자의 주소와 분기형 명령어에서의 목적 주소를 유효 주소라고 함


Memory
22
0
ADD
457
457
Operand
AC
(b) Direct address
Memory
35
1
ADD
300
300
1350
1350
Operand
AC
(c) Indirect address

5.2 컴퓨터 레지스터

◈ 기본 컴퓨터의 레지스터와 메모리

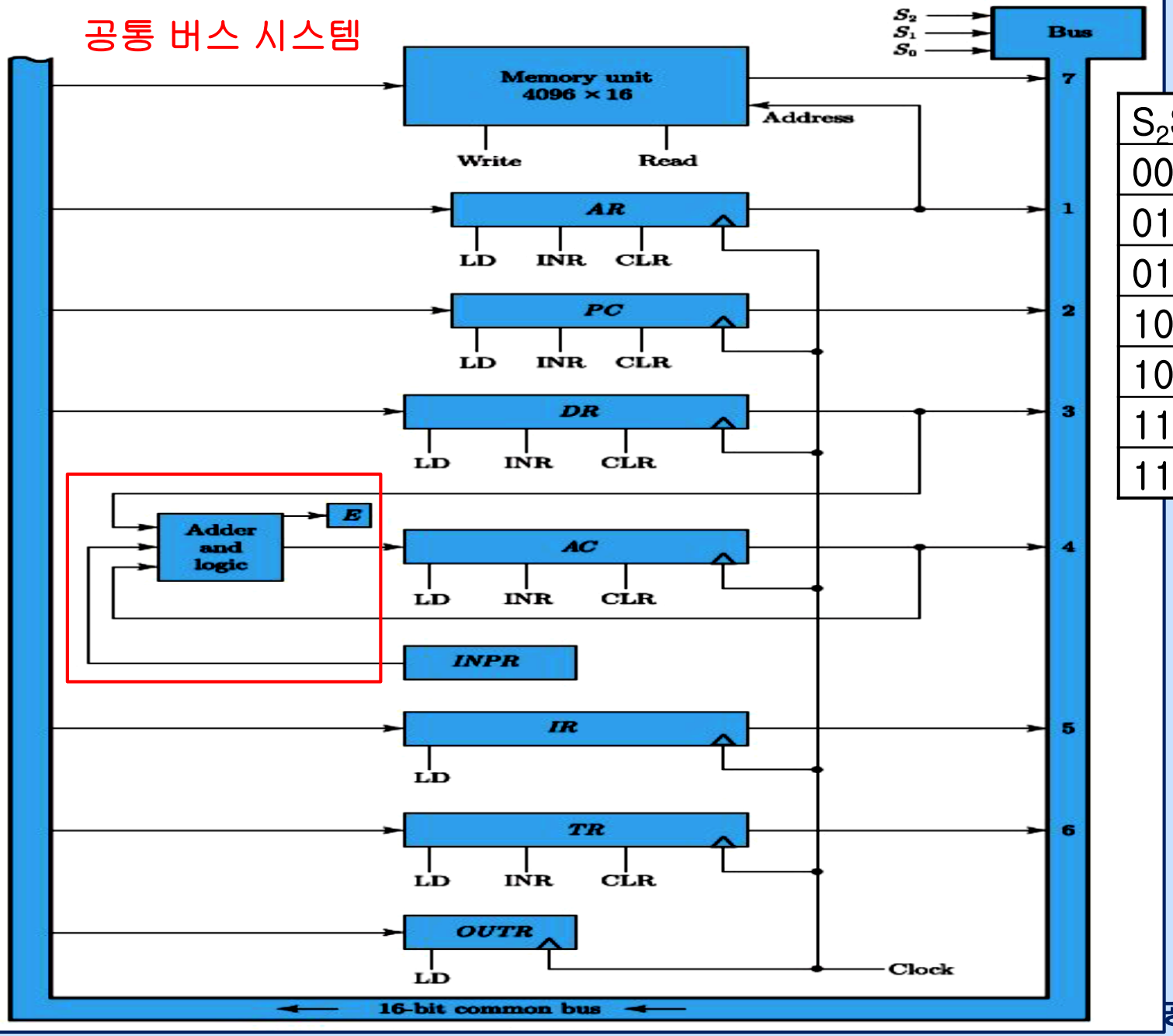


- 프로그램 카운터 (PC) : 다음 수행될 명령어의 주소를 저장 (12비트)
- 메모리 주소 레지스터 (AR) : 메모리 주소를 저장 (12비트)

- 누산기 레지스터 (AC) : 범용 처리 레지스터로 사용
- 명령어 레지스터 (IR) : 메모리에서 읽어온 명령어 코드를 저장
- 데이터 레지스터 (DR) : 메모리에서 읽어온 피연산자를 저장
- 임시 레지스터 (TR) : 계산 도중의 임시 데이터를 저장
- 입력 레지스터 (INPR) : 입력 장치로부터 받은 8비트 입력 문자를 저장
- 출력 레지스터 (OUTR) : 출력 장치로 전송할 8비트 출력 문자를 저장

Register symbol	Number of bits	Register name	Function
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

공통 버스 시스템



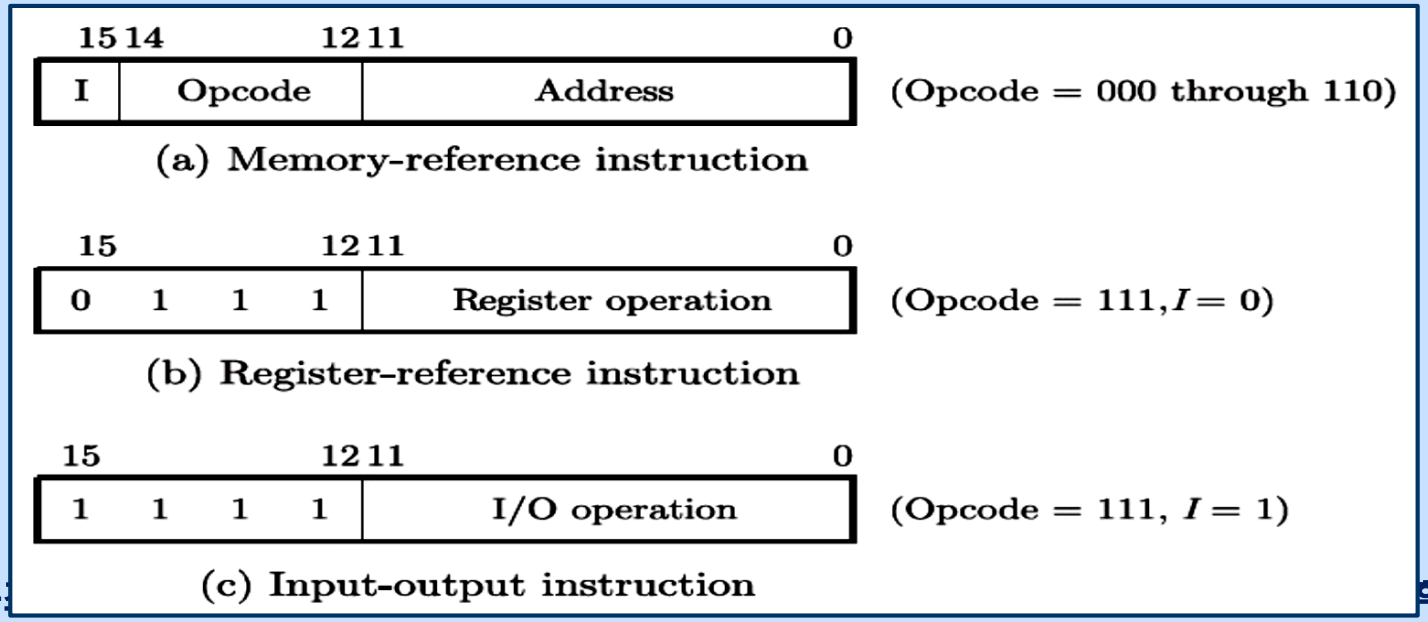
$S_2S_1S_0$	Bus
001(1)	AR
010(2)	PC
011(3)	DR
100(4)	AC
101(5)	IR
110(6)	TR
111(7)	Mem

- 일곱 레지스터와 메모리의 출력이 공통 버스에 연결되고 선택 입력 $S_2S_1S_0$를 통해 버스 위에 놓이게 될 출력 선택
- 공통 버스는 각 레지스터의 입력과 메모리의 입력에 연결되어 있고 로드 입력이 활성화되어 있는 레지스터가 다음 클럭에서 버스의 데이터를 저장
- 12비트의 AR과 PC의 내용이 버스에 전송될 때에는 상위 4비트가 0으로 채워지고 버스의 내용이 AR과 PC에 전송될 때에는 하위 12비트만이 전송
- 8비트의 INPR은 입력 장치로부터 한 문자를 읽어와 AC로 전송하고 OUTR은 AC로부터 한 문자를 읽어와 출력 장치로 전송
- AR, PC, DR, AC, TR은 LD, INR, CLR 등의 세 제어 입력을 가지고 있음
- AC의 16비트 입력은 가산 논리 회로에 연결되고 가산 논리 회로의 입력에는 AC, DR, INPR이 연결되고 end 캐리 출력은 E 플립플롭에 연결
- 레지스터의 내용이 버스에 출력되는 것과 동일한 클럭에 가산 논리 회로의 연산이 수행 (DR ← AC, AC ← DR)

5.3 컴퓨터 명령어

◈ 3가지 종류의 명령어와 명령어 형식

- 메모리 참조 명령어 : 12비트를 주소 지정에 사용하고 주소 모드 I를 사용
- 레지스터 참조 명령어 : 연산 코드가 111, 최상위 비트가 0이며 나머지 12비트는 AC 레지스터에 대한 연산을 나타냄
- 입출력 명령어 : 연산 코드가 111, 최상위 비트가 1이며 나머지 12비트는 입출력 연산의 종류를 나타냄



Symbol	Hexadecimal code $I = 0$	Hexadecimal 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		Complement *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		Interrupt on
IOF	F040		Interrupt off

◈ 명령어 집합의 완전성

- 요구하는 모든 데이터 처리를 위해 충분한 명령어들을 갖고 있어야 함
- 명령어 종류 : 산술, 논리, 시프트 명령어, 메모리와 프로세서 레지스터 사이에 정보를 이동시킬 수 있는 명령어, 상태 조건을 검사하는 명령과 프로그램 제어 명령어, 입력과 출력 명령어
- ADD, CMA, INC 명령만으로도 가감산을 할 수 있으며 순환 명령어 CIR, CIL을 함께 사용해서 곱셈, 나눗셈 등을 수행할 수 있음
- AND, CMA, CLA 명령으로 논리 연산을 실행할 수 있고 메모리와 AC 간의 정보 이동은 LDA와 STA 명령을 사용
- 분기 명령어 BUN, BSA, ISZ와 4개의 Skip 명령어는 상태 조건을 검사하고 프로그램을 제어하는 기능을 제공
- 입력과 출력 명령어는 컴퓨터와 외부 장치 사이의 정보 전송을 수행

5.4 타이밍과 제어

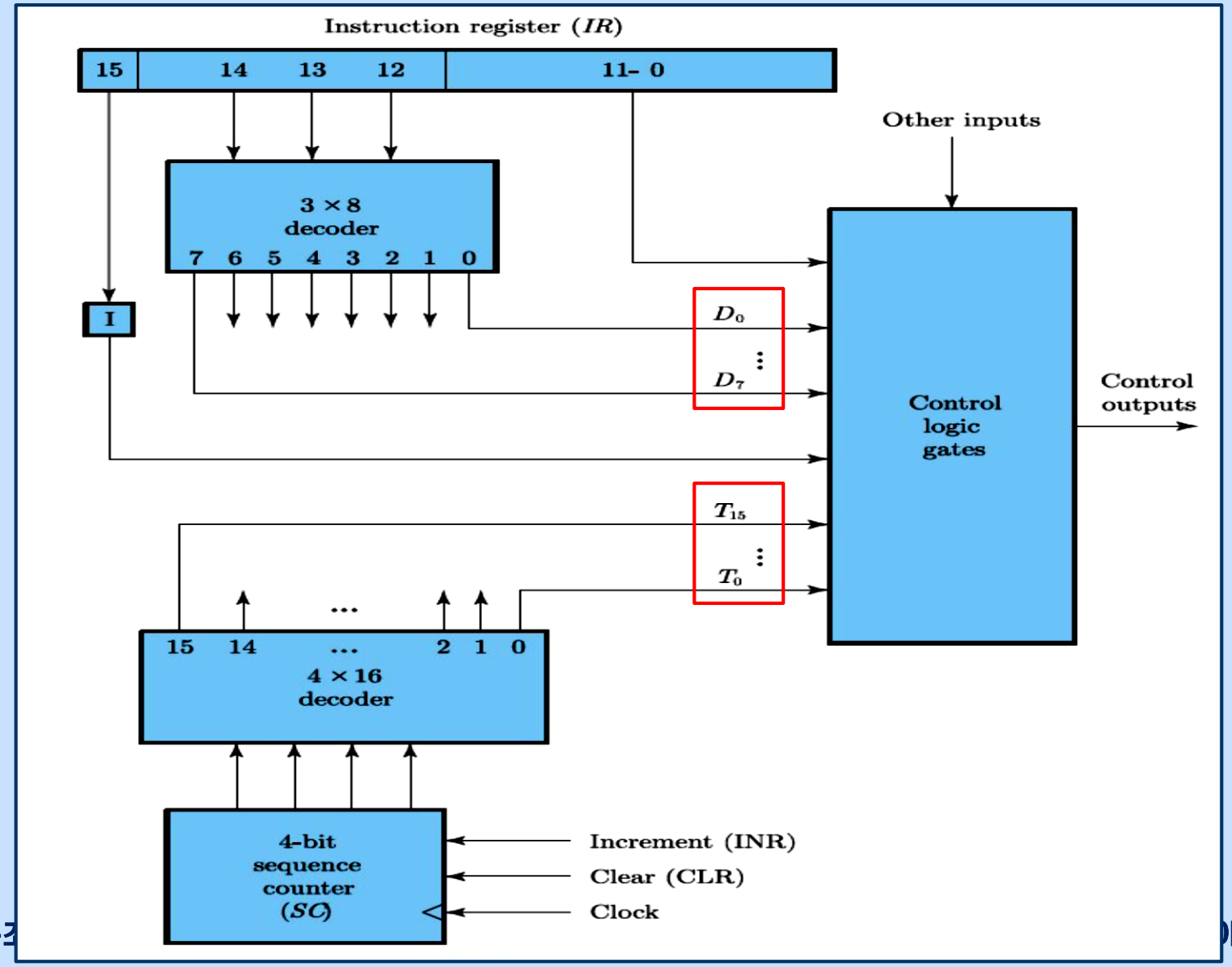
◈ 모든 플립플롭과 레지스터는 클럭 펄스가 인가되고 제어 장치에서 생성된 제어신호가 활성화가 되어야만 상태를 변경시킬 수 있음

◈ 제어 장치는 하드와이어 방식과 마이크로 프로그램 방식의 두 종류가 있음

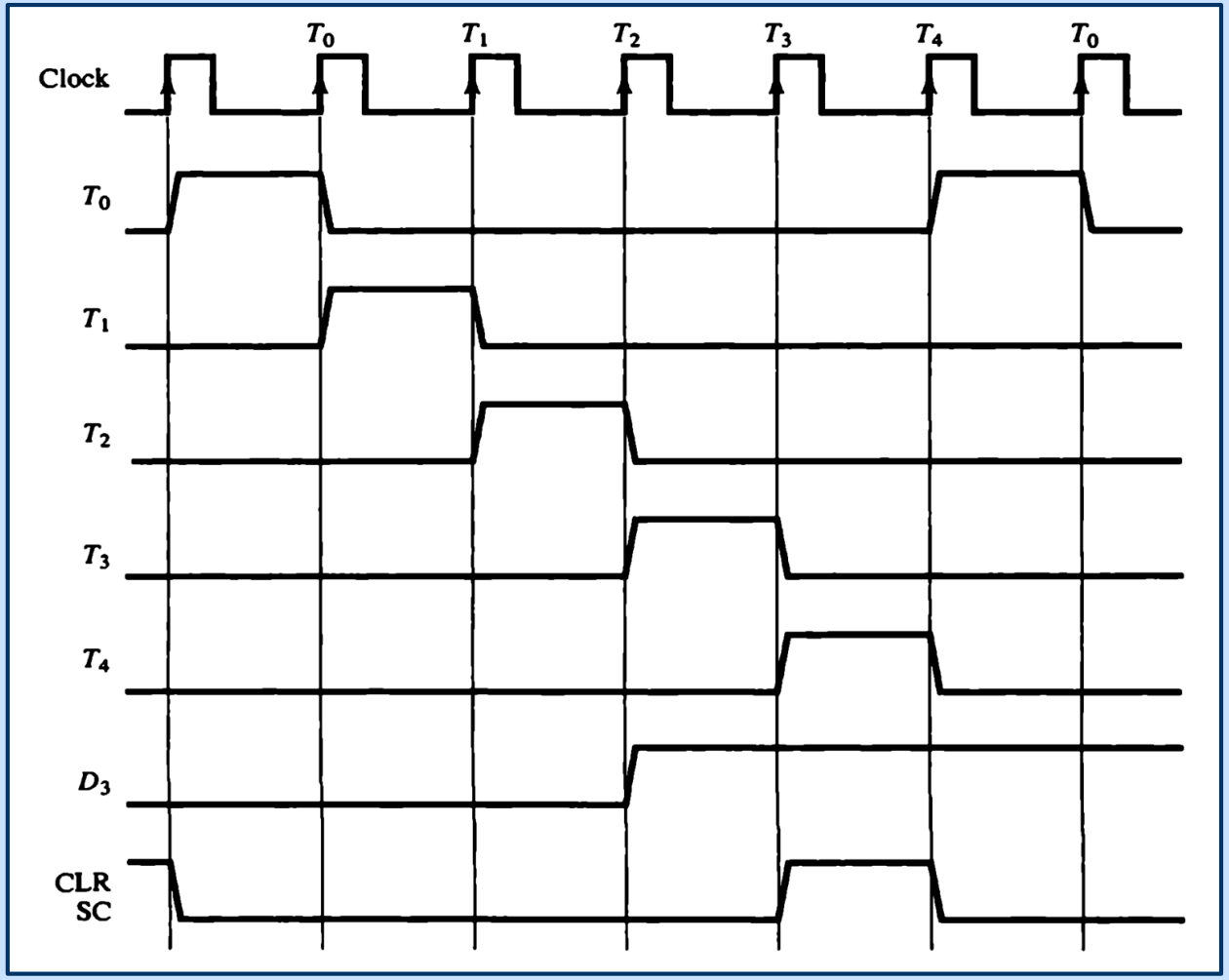
- 하드와이어 방식 : 게이트, 플립플롭, 디코더 등의 디지털 회로를 이용하여 제어 논리를 구현하기 때문에 속도면에서는 유리하지만 컴퓨터 구조가 변경되었을 때 여러 부품들 사이의 배선을 바꾸어 주어야 함
- 마이크로 프로그램 방식 : 제어 메모리에 저장된 제어 정보를 이용하여 마이크로 연산을 순차적으로 수행하기 때문에 설계가 변경되더라도 제어 메모리의 프로그램만 갱신해주면 됨

◈ 기본 컴퓨터의 제어 장치는 두 개의 디코더, 하나의 순차 카운터, 여러 개의 제어 논리 게이트로 구성할 수 있음

– 연산 코드는 디코더에 의해 해석되고 순차 카운터(SC)는 타이밍 신호 생성



- 제어 타이밍 D_3T_4 : SC ← 0 (시간 T_4에서 D_3가 활성화될 때 SC를 클리어함)



- T_0 : AR ← PC (T_0 = 1일 때 PC의 값을 AR로 전송하기 위해 $S_2S_1S_0$ = 010, AR의 LD 입력을 HIGH로 설정함, 다음 클럭의 상승 에지에서 데이터 전송)

5.5 명령어 사이클

◈ 기본 컴퓨터의 명령어 사이클의 단계

- 명령어를 메모리에서 가져옴 (fetch)
- 명령어를 디코딩
- 간접 주소 방식의 명령어일 경우에 메모리로부터 유효주소를 읽어옴
- 명령어를 실행

◈ Fetch와 디코드

- T_0 : AR ← PC (Fetch 단계)

  $S_2S_1S_0$ = 010로 하여 PC 내용이 공통 버스에 놓이게 함

  AR의 LD 입력을 인에이블시켜 버스의 내용을 AR로 전송

- T_1 : IR ← M[AR], PC ← PC + 1 (Fetch 단계)

  메모리의 읽기 입력을 인에이블시킴

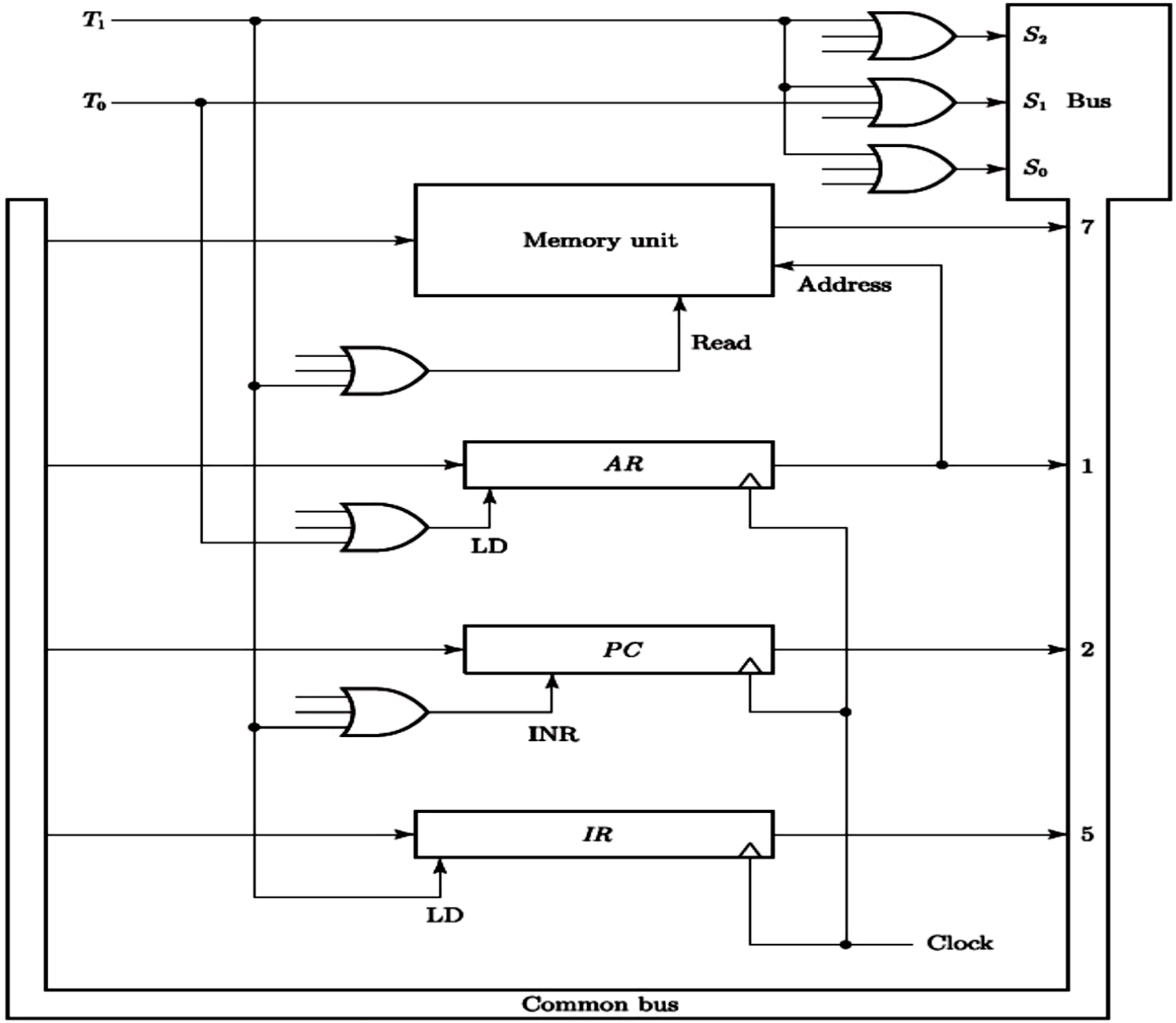
  $S_2S_1S_0$ = 111로 하여 메모리의 내용이 공통 버스에 놓이게 함

  IR의 LD 입력을 인에이블시켜 공통 버스의 내용을 IR로 전송

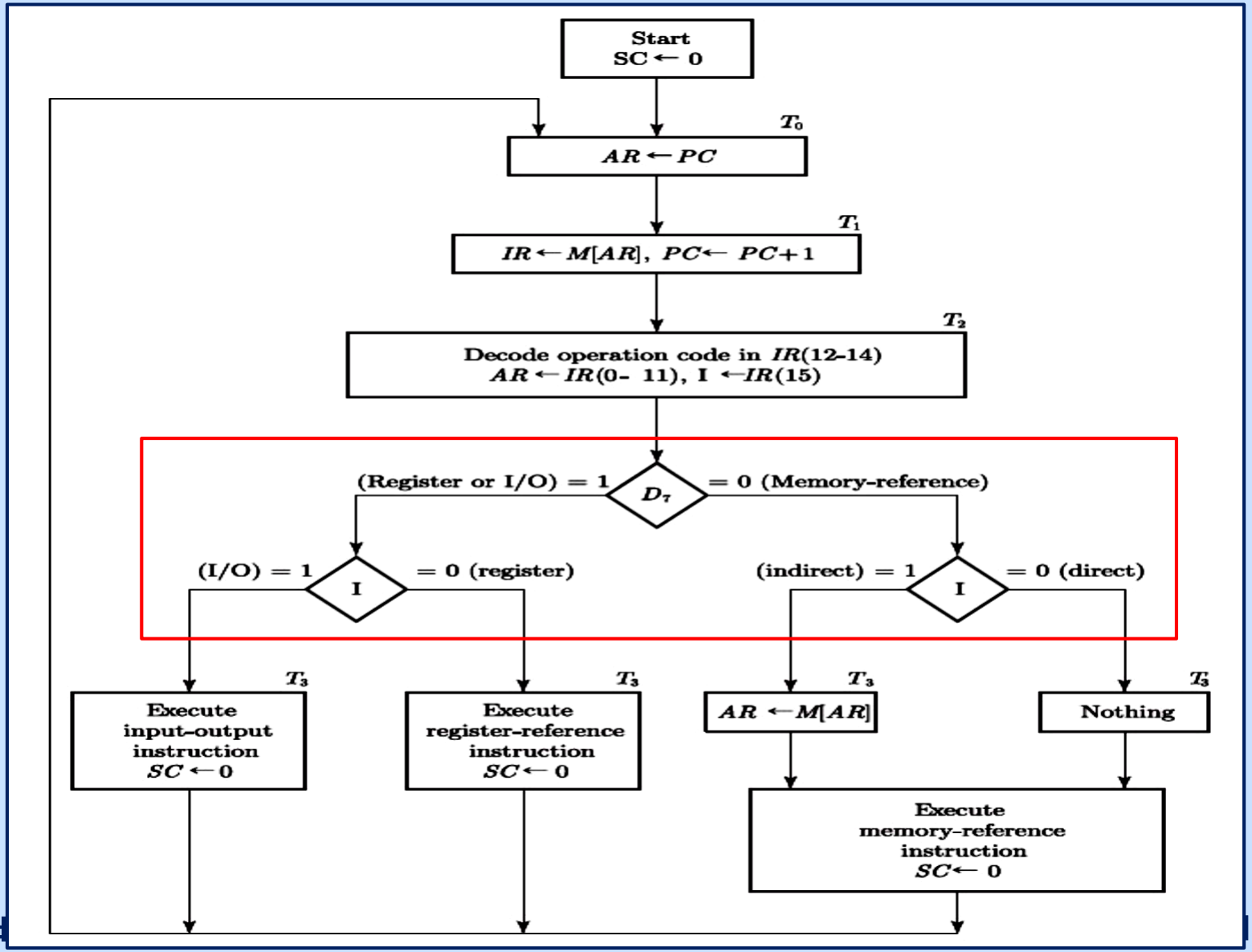
  PC의 INR 입력을 인에이블시켜 PC의 값을 하나 증가시킴

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

- Fetch 단계를 위한 레지스터 전송


T_1
T_0
S_2
S_1
S_0
Bus
Memory unit
Address
Read
AR
LD
PC
INR
IR
LD
Clock
7
1
2
5
Common bus

◈ 명령어 종류의 결정 (시간 T_3 동안에 제어 장치는 명령어의 종류를 결정)



- T_3 동안 3가지 종류의 명령어는 4개의 서로 다른 동작을 수행

  D'_7IT_3 : AR ← M[AR] (메모리 참조 명령어, 간접주소)

  $D'_7I'T_3$: 아무런 일을 하지 않음 (메모리 참조 명령어, 직접주소)

  $D_7I'T_3$: 레지스터 참조 명령어를 수행, D_7IT_3 : 입출력 명령어를 수행

- T_3 동안 아무런 일을 하지 않더라도 다음의 타이밍 변수를 얻기 위해 순차 카운터 증가 동작은 수행하여야 함

- 한 명령어가 수행된 다음에는 SC의 값이 0이 되어 다음 명령어에 대한 fetch 동작으로 제어가 돌아감

◈ 레지스터 참조 명령어 ($D_7 = 1, I = 0$)

- IR(0-11)에 있는 나머지 12비트로 12가지 명령어를 나타냄

- $D_7I'T_3 = r$, $IR(i) = B_i$로 표시하여 각 명령어에 대한 제어 함수를 구별함

- 시간 T_3 동안에 수행이 완료되고 SC는 0으로 돌아감

- 레지스터 참조 명령어의 종류

$D_7I'T_3 = r$ (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

5.6 메모리 참조 명령어

◈ 각 명령어는 디코더의 출력 D_i에 의해 구별되고 피연산자에 대한 유효 주소는 T_2 나 T_3 시간에 AR 레지스터로 전송되어 T_4 시간에 명령어가 수행됨

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

- AND 명령어 : AC와 유효 주소로 지정된 메모리 워드의 각 비트쌍에 대해 AND 연산을 수행하고 결과를 AC로 전송

D_0T_4 : DR ← M[AR], D_0T_5 : AC ← AC ∧ DR, SC ← 0

- ADD 명령어 : 유효 주소에 지정된 메모리 워드의 내용을 AC에 더한 다음 그 합을 AC에 저장하고 출력 캐리는 E 플립플롭에 전송

  D_1T_4 : DR ← M[AR], D_1T_5 : AC ← AC + DR, E ← C_{out}, SC ← 0

- LDA 명령어 : 유효 주소로 지정된 메모리 워드의 내용을 AC에 전송

  D_2T_4 : DR ← M[AR], D_2T_5 : AC ← DR, SC ← 0

- STA 명령어 : AC의 내용을 유효 주소가 지정된 메모리 워드에 전송

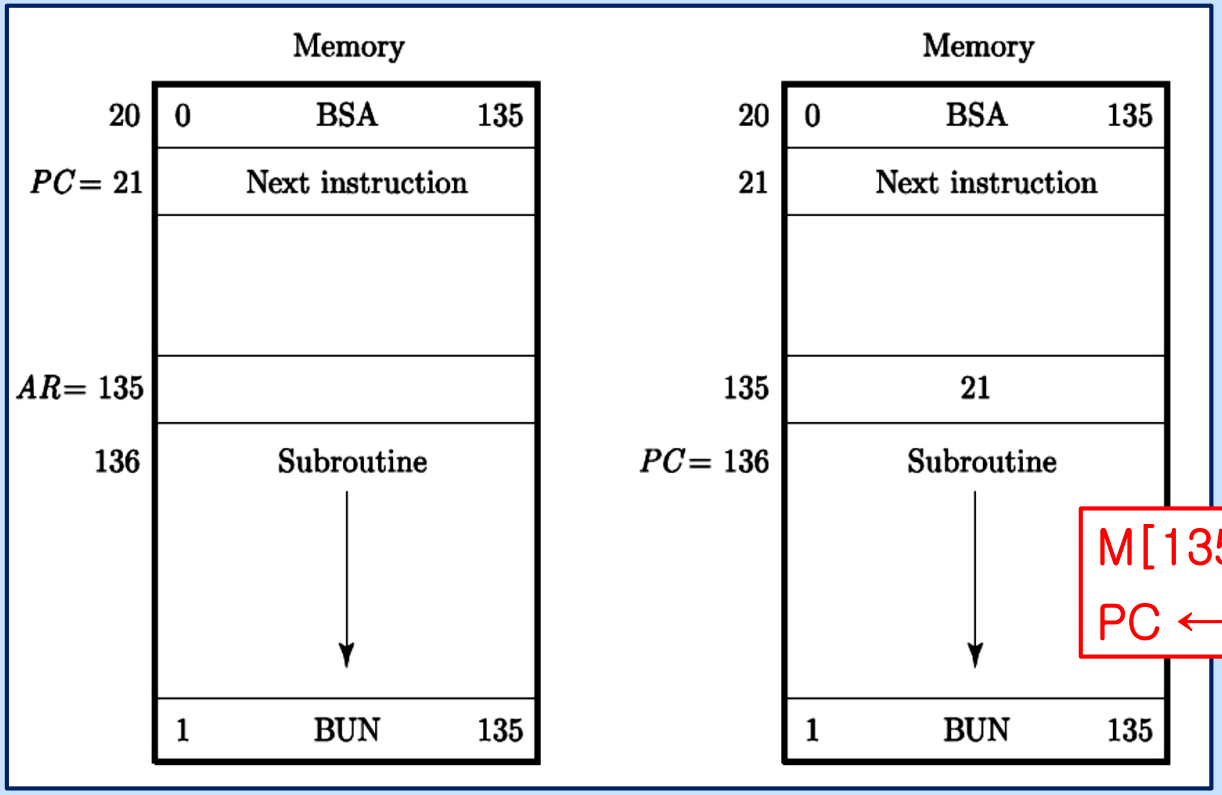
  D_3T_4 : M[AR] ← AC, SC ← 0

- BUN 명령어 : 프로그램의 수행을 유효 주소가 지정된 명령어로 이동

  D_4T_4 : PC ← AR, SC ← 0

- BSA 명령어 : PC의 다음 명령어 주소가 유효 주소로 지정된 메모리에 저장되고 (유효 주소+1)이 PC로 전송되어 서브루틴의 첫 명령어를 가리킴

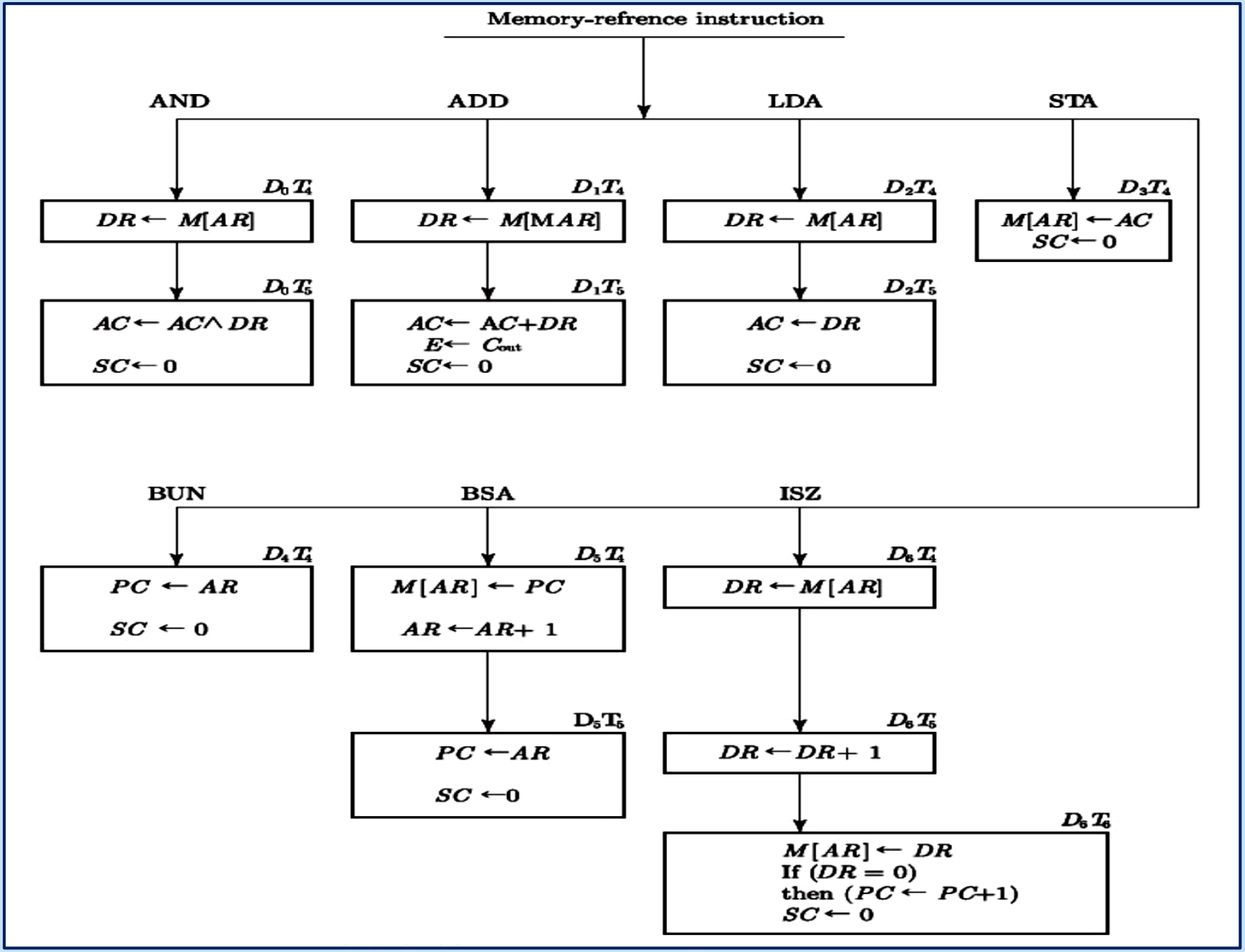
  D_5T_4 : M[AR] ← PC, AR ← AR + 1, D_5T_5 : PC ← AR, SC ← 0



- BSA 명령어는 서브루틴의 호출, BUN 명령어는 서브루틴 복귀에 이용됨
- ISZ 명령어 : 유효 주소로 지정된 워드의 값을 하나 증가시키고 증가된 값이 0이면 PC도 하나 증가시켜 다음 명령어를 수행하게 함

D_6T_4 : DR ← M[AR], D_6T_5 : DR ← DR + 1

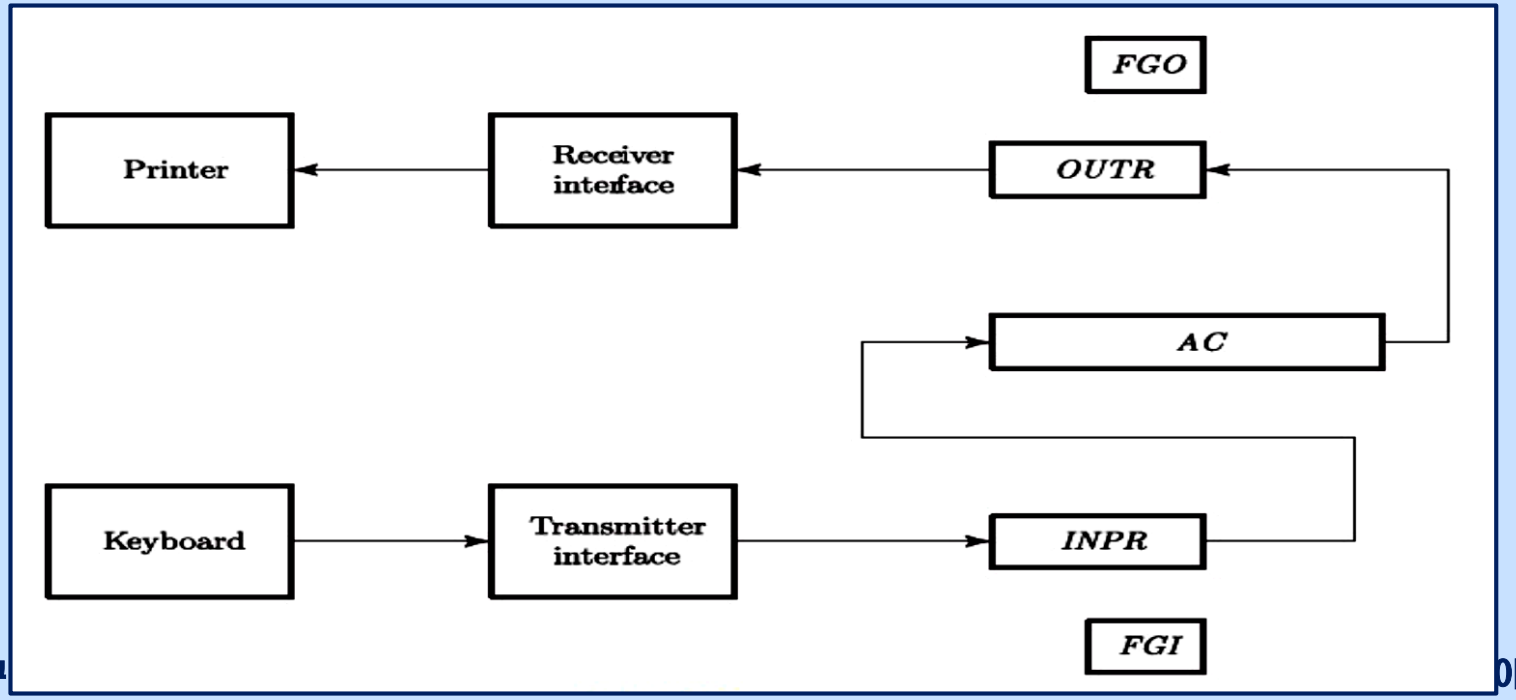
D_6T_6 : M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0


Memory-refrence instruction
AND
ADD
LDA
STA
D_0T_4
$DR \leftarrow M[AR]$
D_0T_5
$AC \leftarrow AC \wedge DR$
$SC \leftarrow 0$
D_1T_4
$DR \leftarrow M[MAR]$
D_1T_5
$AC \leftarrow AC + DR$
$E \leftarrow C_{out}$
$SC \leftarrow 0$
D_2T_4
$DR \leftarrow M[AR]$
D_2T_5
$AC \leftarrow DR$
$SC \leftarrow 0$
D_3T_4
$M[AR] \leftarrow AC$
$SC \leftarrow 0$
BUN
BSA
ISZ
D_4T_4
$PC \leftarrow AR$
$SC \leftarrow 0$
D_5T_4
$M[AR] \leftarrow PC$
$AR \leftarrow AR + 1$
D_5T_5
$PC \leftarrow AR$
$SC \leftarrow 0$
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 $(PC \leftarrow PC + 1)$
$SC \leftarrow 0$

5.7 입출력과 인터럽트

◈ 입출력 구성

- 컴퓨터는 사용자와 통신을 하기 위해 입력장치와 출력장치를 갖추어야 함
- 입출력 장치의 정보는 직렬로 주고 받으며 영문숫자 코드당 8비트 할당
- 키보드로부터의 직렬정보는 입력 레지스터에 시프트되고 프린터에 대한 직렬 정보는 출력 레지스터에 저장



- 제어 플립플롭인 FGI는 키보드에서의 입력정보가 INPR에 저장된 후 세트되고 플래그가 세트되어 있는 동안은 다른 입력정보가 INPR에 저장 안됨
- 컴퓨터는 FGI가 1이 되면 INPR로부터 AC로 입력 정보를 병렬 전송하고 FGI를 클리어시켜 다시 새로운 정보를 받아드릴 수 있도록 함
- 컴퓨터는 FGO가 1이 되면 AC로부터 OUTR로 출력 정보를 병렬 전송한 후 FGO를 클리어시키고 출력 장치는 OUTR로부터 정보를 가져가 문자를 프린트한 후 FGO를 1로 세트시킴
- FGO가 0인 상태에서는 컴퓨터는 OUTR에 새로운 정보를 넣지 못함

◈ 입출력 명령어

- AC 레지스터로 정보를 전송하고 플래그 비트를 검사하여 인터럽트를 제어하는 명령어로 연산 코드가 1111이고 6개의 명령어로 구성

- SKI, SKO 명령어는 플래그가 0이면 다음의 분기 명령에 의해 플래그 검사를 계속하고 플래그가 1이면 다음 명령어를 건너뛰어 다음의 입출력 명령을 수행

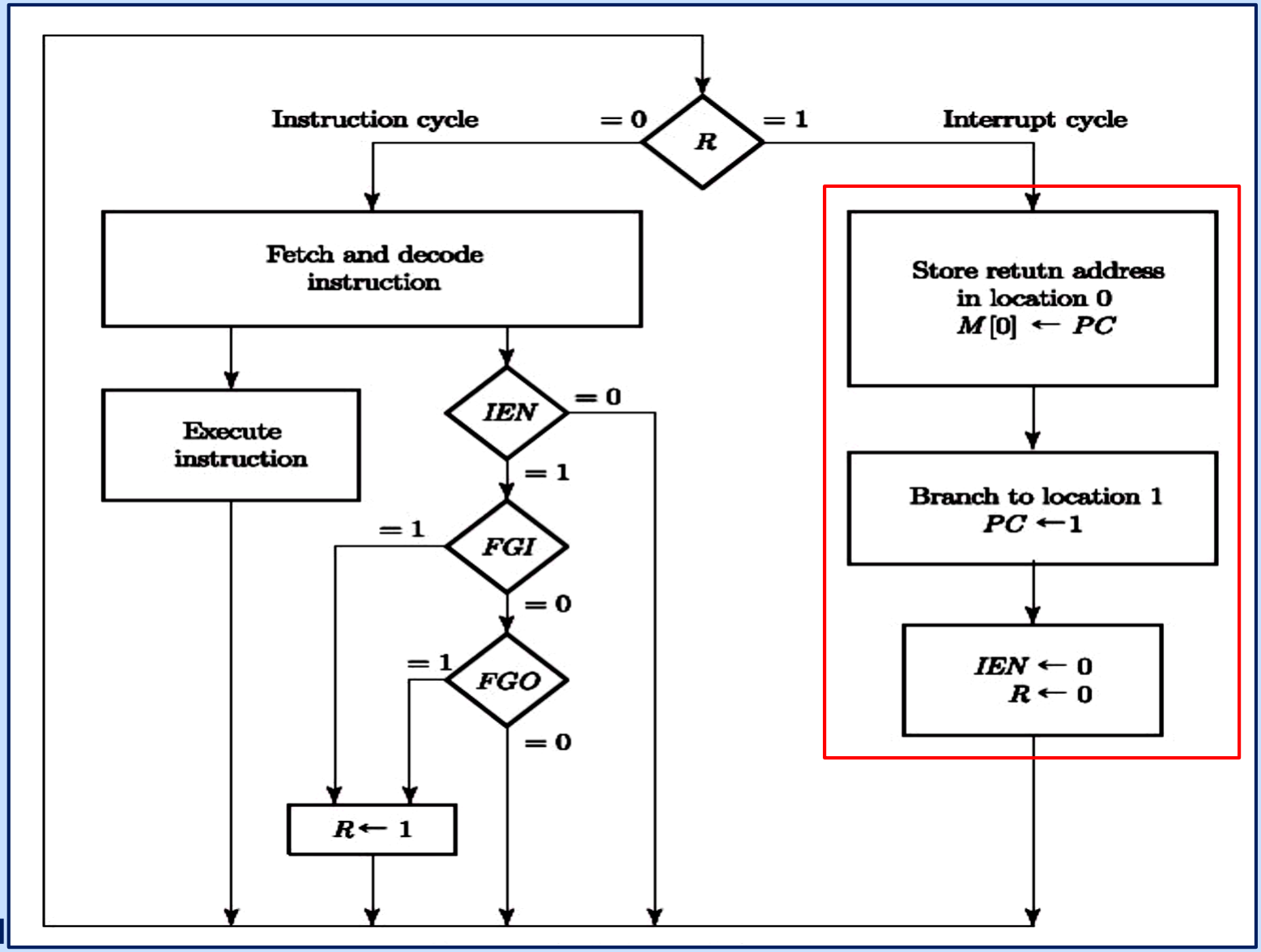
$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If ($FGI = 1$) then ($PC \leftarrow PC + 1$)	Skip on input flag
SKO	pB_8:	If ($FGO = 1$) then ($PC \leftarrow PC + 1$)	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

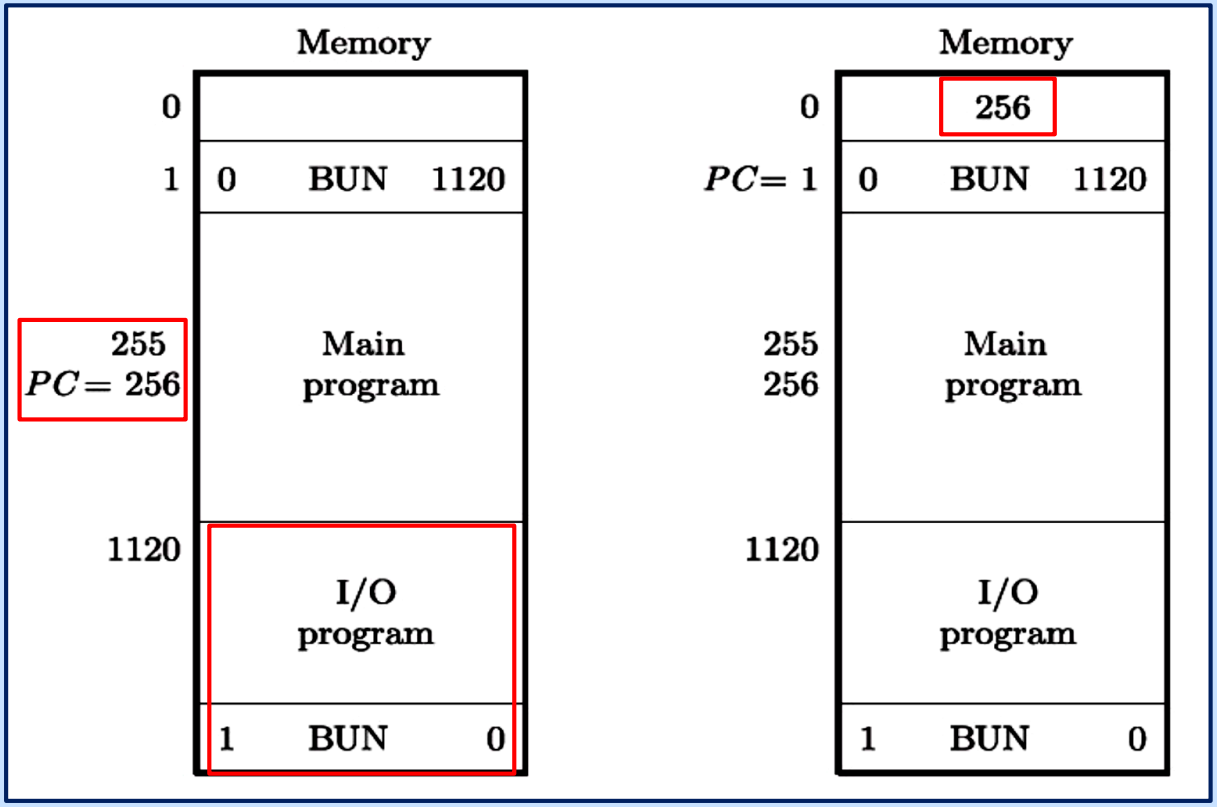
◈ 프로그램 인터럽트

- 입출력 장치의 속도가 매우 느려 플래그를 검사하여 문자를 처리하는 방법보다 외부 장치가 전송 준비가 되었을 때 컴퓨터에게 알리는 방법을 사용
- 컴퓨터는 플래그를 검사하지 않고 플래그가 세트되면 현재 프로그램을 중지하고 입출력을 실행한 후 다시 인터럽트 이전에 실행하던 프로그램을 계속 수행
- 인터럽트 인에이블 플립플롭 IEN가 1인 경우에만 인터럽트를 걸 수가 있음
- 인터럽트 플립플롭 R이 0이면 명령어 사이클을 수행하고 명령어 사이클의 실행 단계에서 IEN과 두 플래그 비트를 검사하여 R값을 결정
- R이 1이면 인터럽트 사이클이 수행되어 복귀 주소는 0번지에 저장되고 PC에 주소 1을 넣고 IEN, R을 클리어하여 다른 인터럽트가 발생하지 않도록 함

- 인터럽트 사이클의 흐름도



- 인터럽트 사이클의 수행 과정



◈ 인터럽트 사이클

- 인터럽트 플립플롭 R은 T_0,T_1,T_2 외의 어떤 시간에서도 IEN = 1이고

  FGO나 FGI가 1일 때 1로 세트됨 ($T_0'T_1'T_2'$(IEN)(FGI + FGO) : R ← 1)

- R=0일 때만 명령어 사이클이 수행되도록 Fetch와 디코드 단계에서의 제어 함수를 $R'T_0$, $R'T_1$, $R'T_2$와 같이 수정
- 인터럽트 사이클에서 수행되는 마이크로 연산

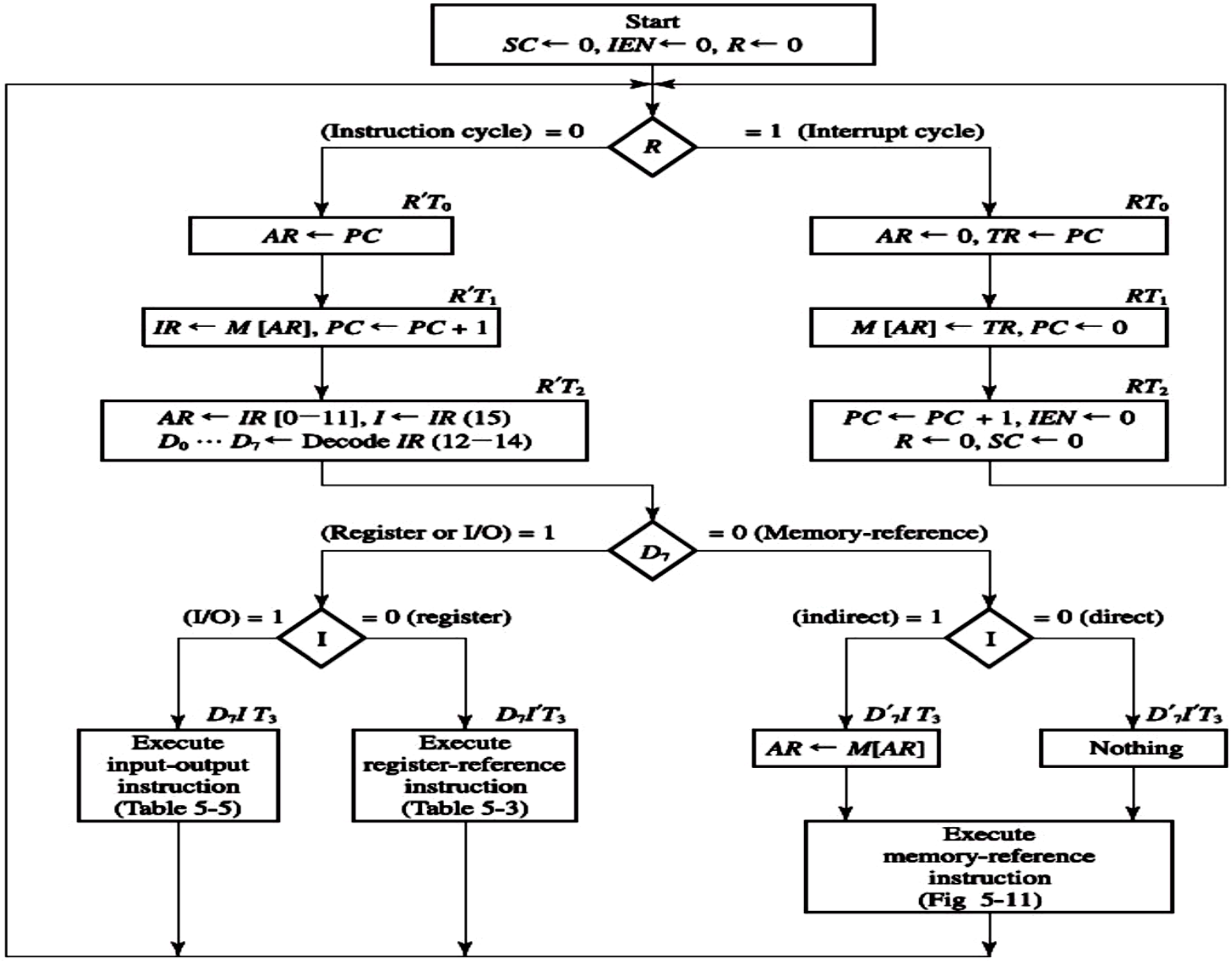
  RT_0 : AR ← 0, TR ← PC

  RT_1 : M[AR] ← TR, PC ← 0

  RT_2 : PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0
- 복귀 주소를 저장하기 위해 TR 레지스터를 사용하고 BUN 명령어를 수행하여 인터럽트 서브루틴으로 분기가 일어남

5.8 컴퓨터에 대한 완전한 기술

◈ 컴퓨터의 동작 기술은 명령어 실행을 나타내는 흐름도와 레지스터 전송문을 이용하여 나타낼 수 있음


Start
SC ← 0, IEN ← 0, R ← 0
R
(Instruction cycle) = 0
= 1 (Interrupt cycle)
R′T0
AR ← PC
R′T1
IR ← M [AR], PC ← PC + 1
R′T2
AR ← IR [0—11], I ← IR (15)
D0 ··· D7 ← Decode IR (12—14)
RT0
AR ← 0, TR ← PC
RT1
M [AR] ← TR, PC ← 0
RT2
PC ← PC + 1, IEN ← 0
R ← 0, SC ← 0
D7
(Register or I/O) = 1
= 0 (Memory-reference)
I
(I/O) = 1
= 0 (register)
I
(indirect) = 1
= 0 (direct)
D7I T3
Execute input-output instruction (Table 5-5)
D7I′T3
Execute register-reference instruction (Table 5-3)
D′7I T3
AR ← M[AR]
D′7I′T3
Nothing
Execute memory-reference instruction (Fig 5-11)

5.9 기본 컴퓨터의 설계

◈ 하드웨어 구성 요소

- 16비트의 4096워드를 가진 메모리 장치
- 9개의 레지스터 : AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
- 7개의 플립플롭 : I, S, E, R, IEN, FGI, FGO
- 2개의 디코더 : 3×8 동작 디코더와 4×16 타이밍 디코더
- 16비트 공통 버스, 제어 논리 게이트들, AC 입력에 연결된 가산 논리 회로

◈ 제어 논리 회로의 출력

- 9개 레지스터의 입력을 제어하는 신호
- 메모리의 쓰기 및 읽기 입력을 제어하는 신호
- 플립플롭을 세트, 클리어, 보수화시키는 신호
- 버스를 사용할 레지스터를 선택하는데 사용되는 S_2, S_1, S_0에 대한 신호
- AC에 대해 가산 논리 회로를 제어하는 신호

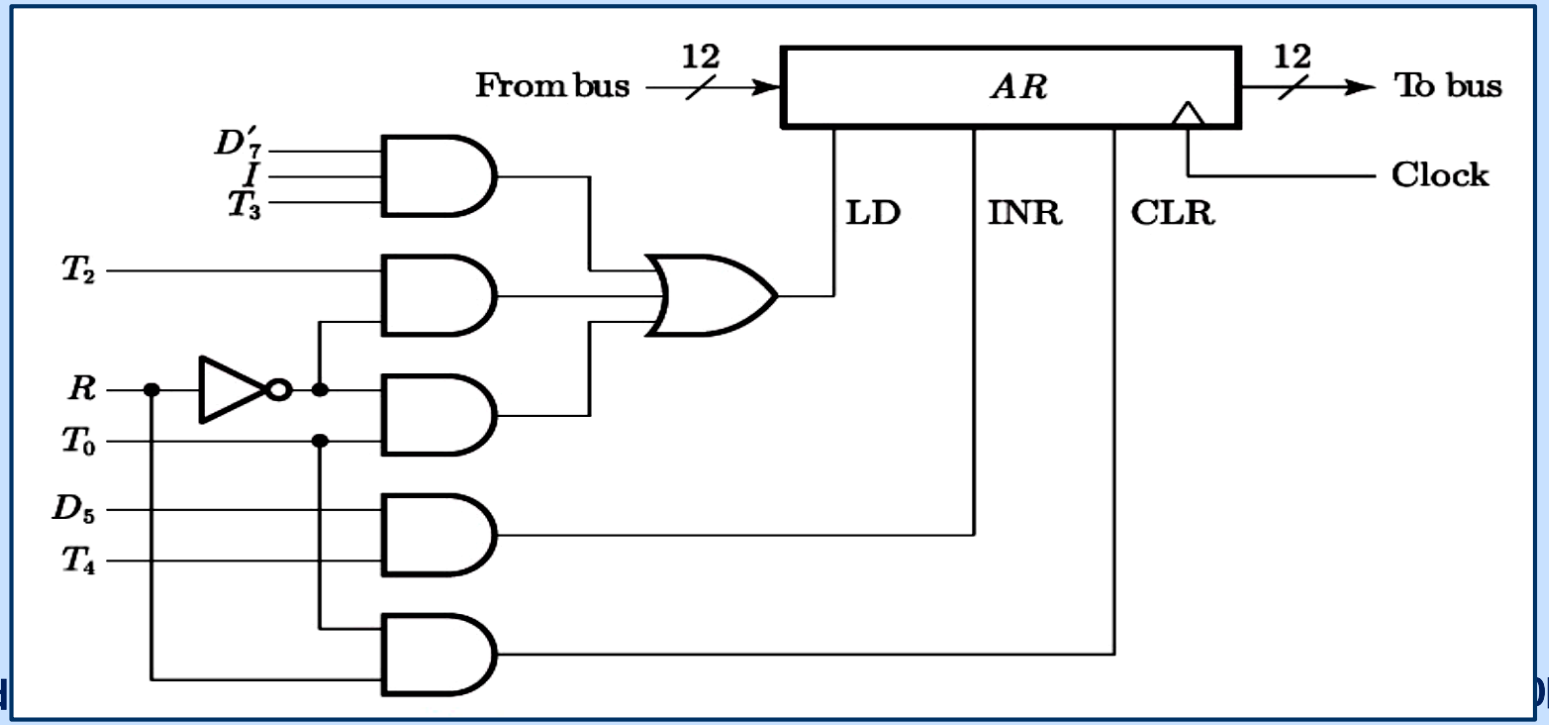
◈ 레지스터와 메모리에 대한 제어 (LD, INR, CLR)

- AR 레지스터의 제어 입력을 구성하는 게이트 구조를 찾으려면 표 5-6에서 AR의 내용을 변경시키는 모든 문장을 찾아야 함

$R'T_0$: AR ← PC (LD), $R'T_2$: AR ← IR(0-11) (LD)

D'_7IT_3 : AR ← M[AR] (LD), RT_0 : AR ← 0 (CLR), D_5T_4 : AR ← AR+1 (INR)

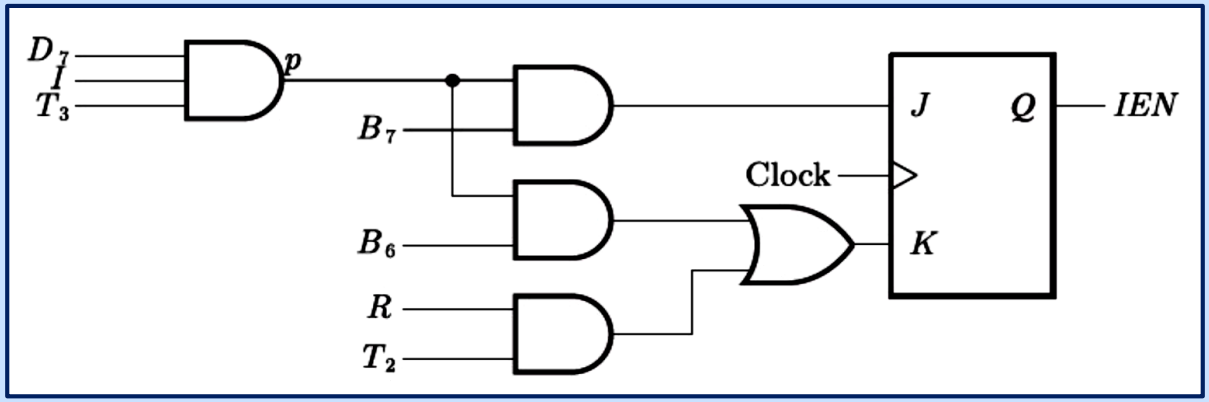
⇒ $AR(LD) = R'T_0 + R'T_2 + D'_7IT_3$, $AR(CLR) = RT_0$, $AR(INR) = D_5T_4$



- 메모리 읽기 제어 입력은 기호 ← M[AR]로 표시되는 읽기 동작들로부터 구할 수 있음 ($READ = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$)

◈ 단일 플립플롭에 대한 제어

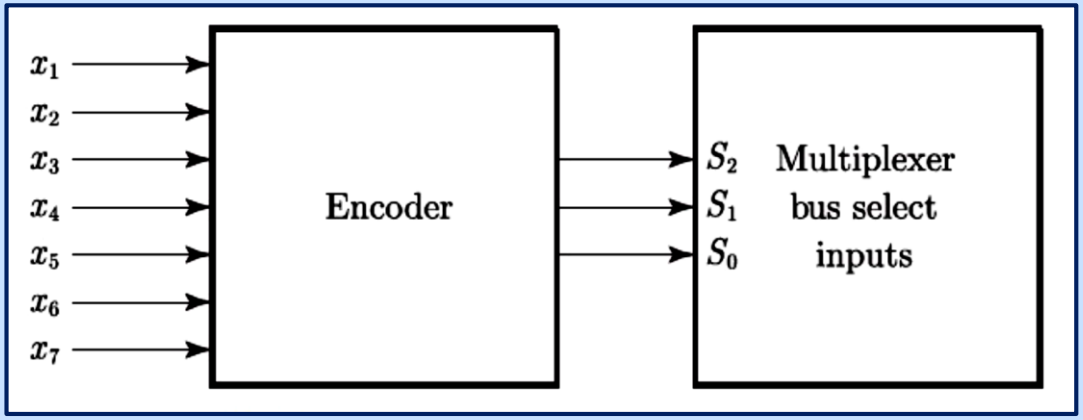
- 7개의 플립플롭에 대한 제어 게이트 논리도 유사한 방법으로 구할 수 있음
- IEN은 ION과 IOF 명령의 결과로 값이 변하며 인터럽트 사이클의 마지막에서 0으로 클리어됨 (pB_7 : IEN ← 1, pB_6 : IEN ← 0, RT_2 : IEN ← 0)



◈ 공통 버스에 대한 제어

- 공통 버스는 S_2,S_1,S_0에 의해서 제어되므로 인코더에 의해 나타낼 수 있음

- 버스 선택 회로를 위한 인코더



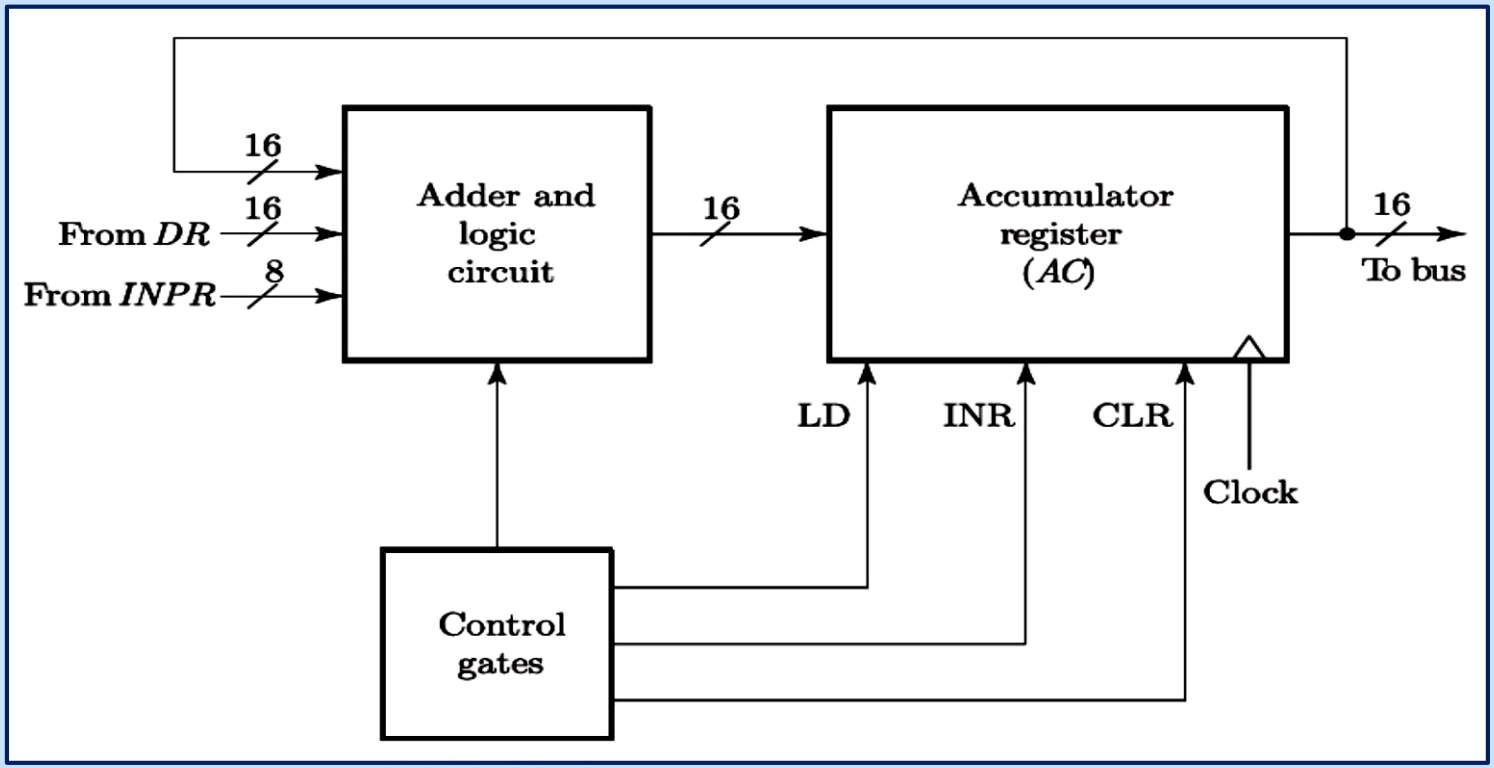
Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

- 인코더에 대한 부울식

$$S_0 = x_1 + x_3 + x_5 + x_7, \quad S_1 = x_2 + x_3 + x_6 + x_7, \quad S_2 = x_4 + x_5 + x_6 + x_7$$

- x_1을 1로 만드는 논리는 AR을 버스의 근원지로 선택하는 문장을 뽑아서 구할 수 있음 (D_4T_4 : PC ← AR, D_5T_5 : PC ← AR ⇒ $x_1 = D_4T_4 + D_5T_5$)
- x_7을 만드는 논리는 메모리 읽기 신호 (READ)와 동일한 부울식으로 표현됨

5.10 누산기 논리의 설계



◈ AC 레지스터에 대한 제어

- 가산 논리 회로에는 AC의 LD, INR, CLR 입력을 제어하기 위한 논리 게이트와 가산 논리 회로의 동작을 제어하기 위한 논리 게이트들이 포함되어 있음
- AC의 값을 변경하는 레지스터 전송문

D_0T_5 : AC ← AC ∧ DR (AND=D_0T_5), D_1T_5 : AC ← AC + DR (ADD=D_1T_5)

D_2T_5 : AC ← DR (DR=D_2T_5), pB_{11} : AC(0-7) ← INPR (INPR=pB_{11})

rB_9 : AC ← $\overline{AC}$ (COM=rB_9), rB_7 : AC ← shr AC, AC(15) ← E (SHR=rB_7)

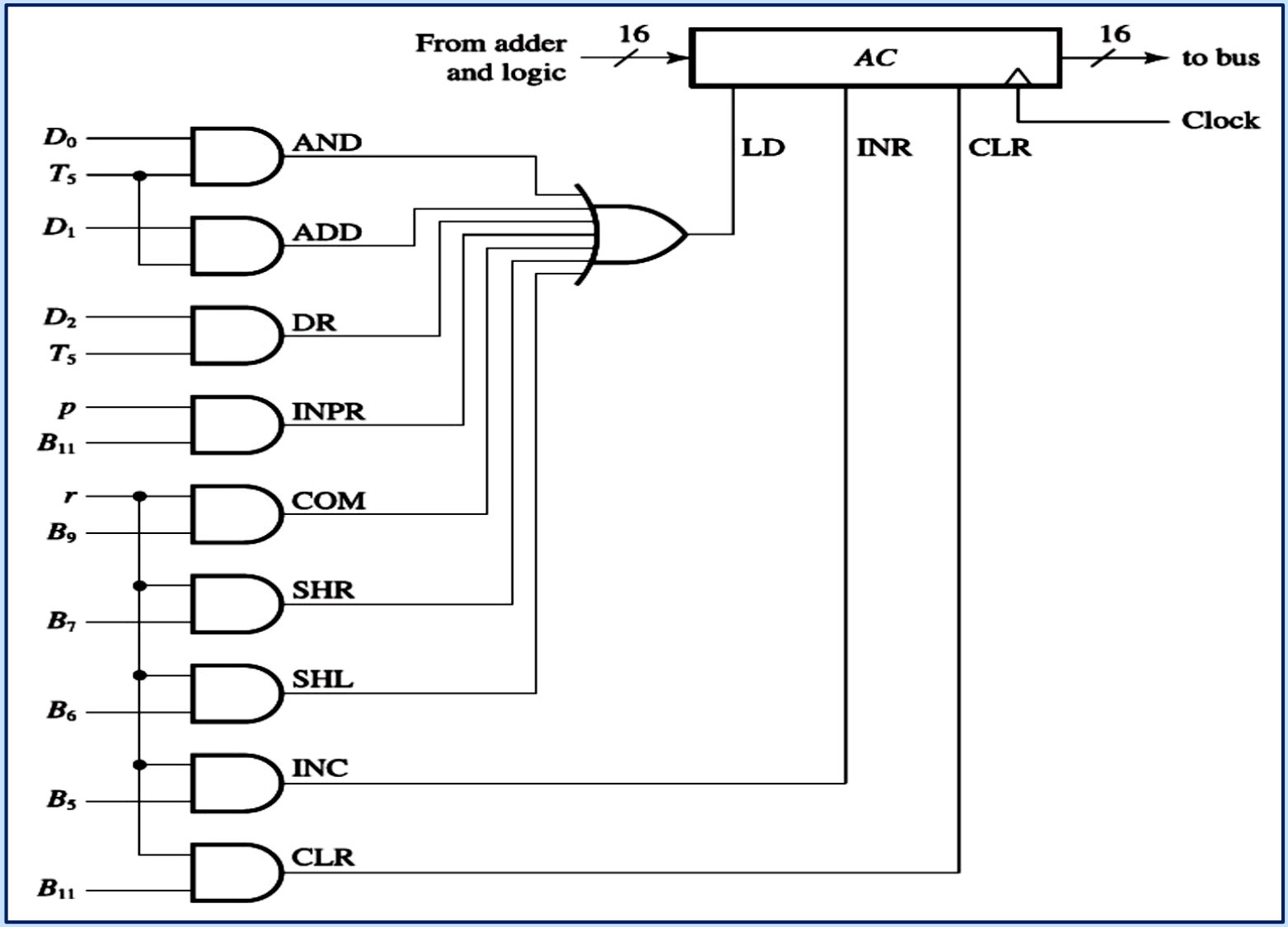
rB_6 : AC ← shl AC, AC(0) ← E (SHL=rB_6)

⇒ AC(LD) = AND + ADD + DR + INPR + COM + SHR + SHL

rB_{11} : AC ← 0 (CLR=rB_{11}) ⇒ AC(CLR) = CLR

rB_5 : AC ← AC + 1 (INC=rB_5) ⇒ AC(INR) = INC

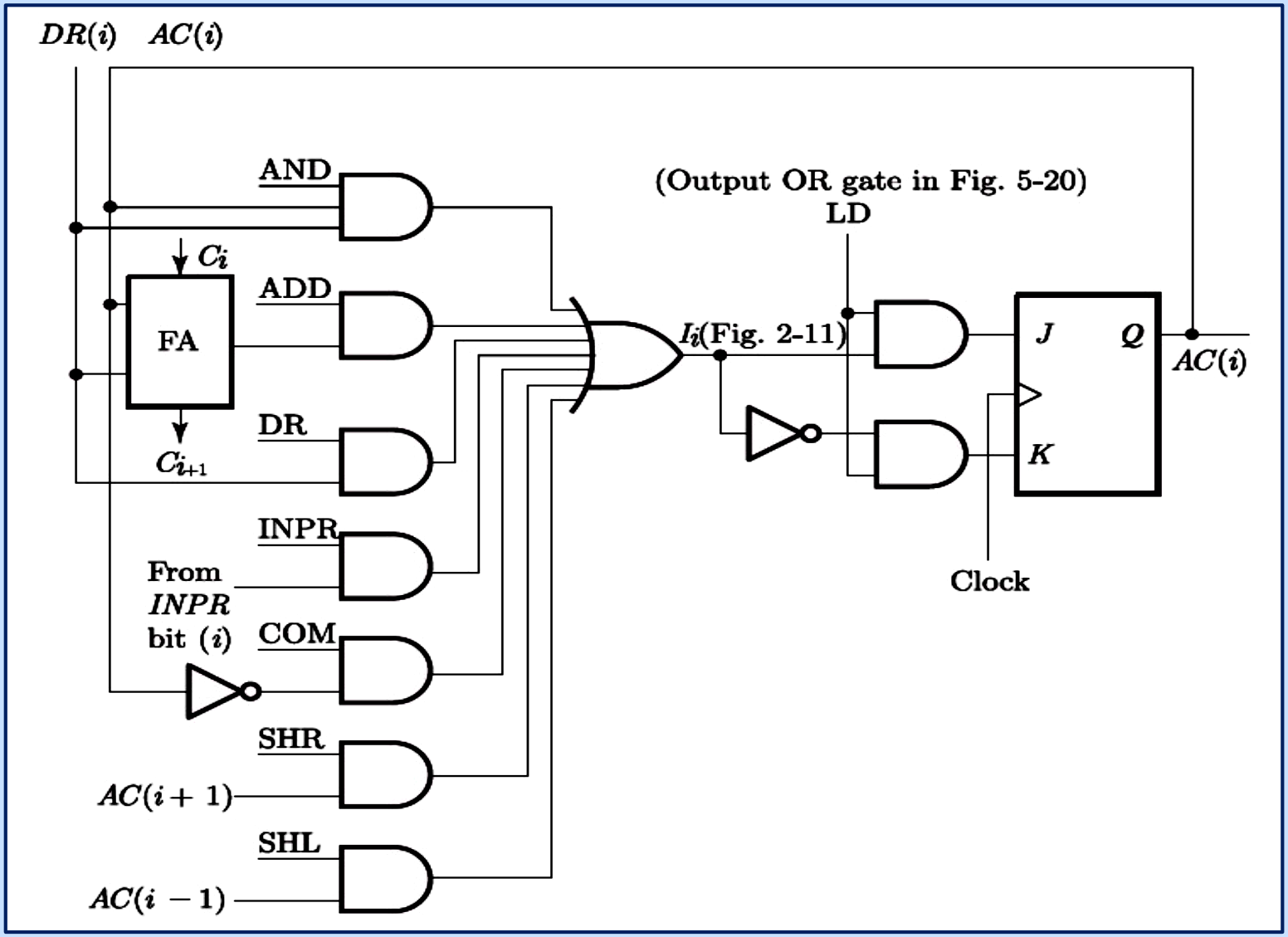
- AC의 LD, INR, CLR을 제어하는 게이트 구조



◈ 가산 논리 회로

- AC의 각 비트에 해당하는 16개의 단으로 나뉘어져 있으며 7개의 AND 게이트와 1개의 OR 게이트, 1개의 전가산기로 구성되어 있음
- 각 AND 게이트의 입력은 그림 5-20에서 같은 기호로 표시된 게이트 출력이 연결됨

– 가산 논리 회로의 한 단


DR(i) AC(i)
AND
(Output OR gate in Fig. 5-20)
LD
Ci
ADD
FA
Ii(Fig. 2-11)
J
Q
AC(i)
DR
Ci+1
K
INPR
From
INPR
bit (i)
COM
Clock
SHR
AC(i + 1)
SHL
AC(i − 1)