Slide 0

# 5장. 기본 컴퓨터의 구조와 설계

## 명령어 코드

Slide 1

- 컴퓨터 동작 기술시 필요것들
  - 내부 레지스터의 기술
  - 타이밍과 제어구조
  - 명령어 집합
- 컴퓨터 내부 동작 기술
  - microoperation sequence 에 의해 정의됨
  - 특수목적 디지탈 시스템 : 하드웨어에 의해 고정된 회로 사용
  - 범용 디지탈 컴퓨터 : 피연산자와 처리순서가 프로그램 방식으로 메모리에 저장되어있음
    $\rightarrow$내장프로그램 방식 이라고 함

## 명령어 코드 (계속)

Slide 2

- 명령어 코드
  - 컴퓨터에게 특정 동작을 시키는 최소단위 (실행 프로그램 형태로 메모리에 저장됨)
  - 연산코드와 피연산자들로 되어있음
- 연산 (operation) 과 마이크로 연산 (microoperation)
  - 연산 (operation)
    - 명령어 코드에 기술된 연산코드에 의하여 정의된 연산들
    - 메모리에 저장된 프로그램 명령의 최소단위
  - 마이크로 연산 (microoperation)
    - 하나의 연산을 수행하기 위하여 필요한 여러가지 명령들을 수행하는 연산들
    - 하나의 연산은 다수개의 마이크로 연산에 의하여 수행됨
      →그러므로 연산을 매크로 연산 (macrooperation) 이라고도 함

## 명령어 코드 (계속)

Slide 3

- 저장 프로그램 구조
  - 가장 간단한 컴퓨터의 구성 (그림 5-1)



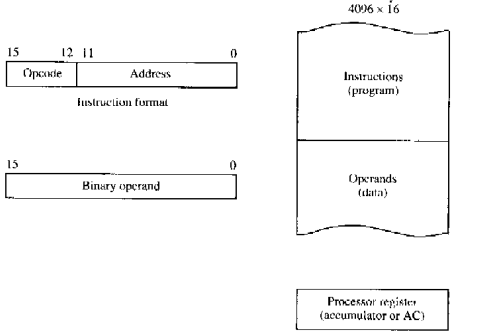


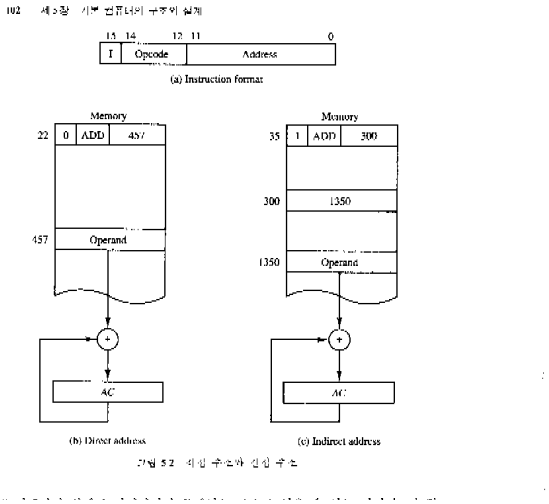

그림 5-1 내장 프로그램 구조

있다.

## 명령어 코드 (계속)

**Slide 4**

- 간접 주소 (그림 5-2)




## 명령어 코드 (계속)

**Slide 5**

- immediate addressing : 피연산자가 명령어 코드의 주소부분에 실림
- direct addressing : 피연산자가 있는 메모리의 주소가 명령어 코드의 주소부분에 실림
- indirect addressing : 피연산자가 있는 메모리의 주소를 저장하고 있는 장소의 주소가 명령어 코드의 주소부분에 실림
- 명령어 코드의 msb 를 사용 direct, indirect 구별
  - msb 가 0 이면 direct
  - msb 가 1 이면 indirect

## 컴퓨터 레지스터

- 프로그램을 수행하기 위하여 컴퓨터는 여러개의 레지스터가 필요
  - 필요한 레지스터들 (그림 5-3)

**Slide 6**

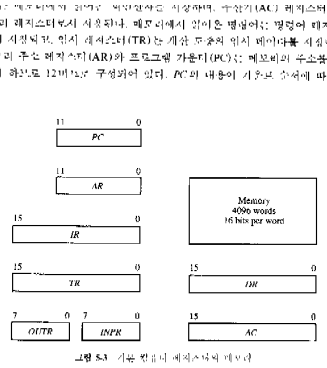



그림 5-3 기본 컴퓨터 레지스터의 다이어그램

  - 레지스터의 기능 (표 5-1)

## 컴퓨터 레지스터 (계속)

**Slide 7**

104 제5장 기본 컴퓨터의 구조와 설계

표 5-1 기본 컴퓨터를 위한 레지스터

| 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 |

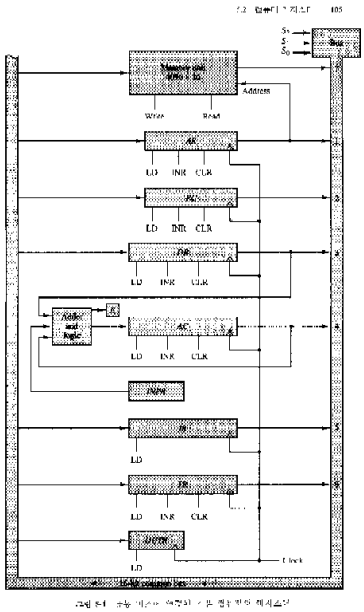
## 컴퓨터 레지스터 (계속)

**Slide 8**

- 공통 버스 시스템 (그림 5-4)

## 컴퓨터 레지스터 (계속)

**Slide 9**



## 컴퓨터 레지스터 (계속)

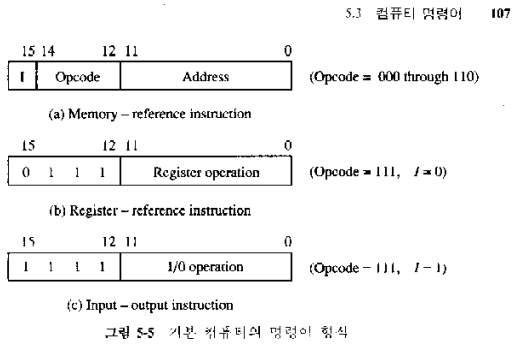
**Slide 10**

– 레지스터와 레지스터 사이, 레지스터와 메모리 사이의 정보전달을 위한 버스

## 컴퓨터 명령어

**Slide 11**

- 기본 컴퓨터의 세가지 명령어 코드 형식 (그림 5-5)




- 3 비트의 연산코드 →최대 8가지의 연산코드 가능
- 그러나 레지스터 참조명령과 I/O 명령에서 12 비트가 연산 코드로 사용됨으로 더욱 많음

## 컴퓨터 명령어 (계속)

**Slide 12**

- 이책에서는 25 가지의 명령어만 사용 (표 5-2)

## 컴퓨터 명령어 (계속)

**Slide 13**

108 — 제5장 기본 컴퓨터의 구조와 설계

표 5-2 컴퓨터 명령어

| 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 |

## 컴퓨터 명령어 (계속)

**Slide 14**

- 명령어 집합의 완전성
  - 모든 데이타 처리를 수행할수 있도록 하기위한 명령어들
    1. 산술, 논리, 시프트 명령어
    2. 메모리와 프로세서 레지스터 간의 정보 이동 명령어
    3. 상태조건을 검사하는 명령과 프로그램 제어 명령어
    4. 입력과 출력 명령어
    5. 표 5-2 는 위의 명령어들을 모두 제공하는 최소한의 명령어임
       복잡한 명령은 프로그램으로 구현

## 타이밍과 제어

**Slide 15**

- 레지스터간 또는 메모리간의 정보 전송 및 연산을 위하여 제어 장치가 필요
- 제어장치의 방식
  - 하드와이어 (hardwired) 방식
  - 마이크로 프로그램 (microprogrammed) 방식
- 하드와이어 (hardwired) 방식
  - 게이트, 플립플롭, 디코더 등의 디지탈 회로를 이용하여 구현
  - 장점 : 속도 빠름
  - 단점 : 구조 변경시 배선 변경해주어야함
- 마이크로 프로그램 (microprogrammed) 방식
  - 제어 메모리에 저장된 제어정보를 이용
  - 장점 : 설계 변경시 제어 메모리의 마이크로 프로그램만 변경하면 됨
  - 단점 : 하드와이어 방식에 비해 속도가 느림

## 타이밍과 제어 (계속)

**Slide 16**

- 기본 컴퓨터의 제어 장치 (그림 5-6)

## 타이밍과 제어 (계속)

**Slide 17**

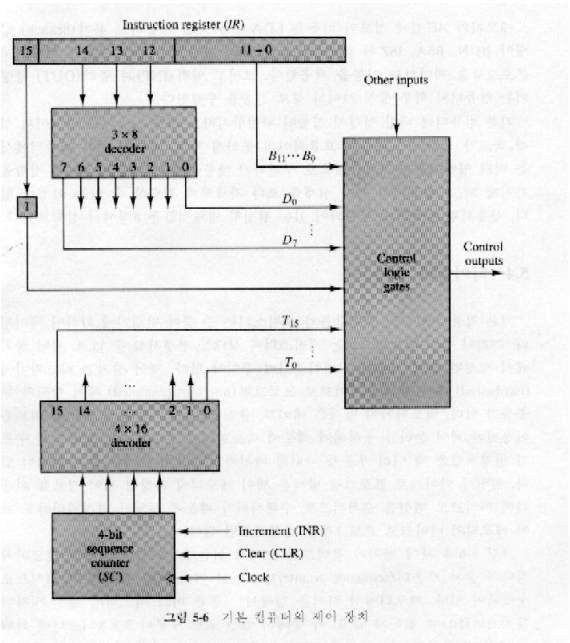



그림 5-6 기본 컴퓨터의 제어 장치

## 타이밍과 제어 (계속)

**Slide 18**

- 연산코드 해석하는 3 × 8 디코더
- 직접 및 간접 주소의 정보를 저장하는 플립플롭
- 16 개의 타이밍 신호를 발생하기위한 4비트 카운터와 4 × 16 디코더
- 순차 카운터는 동기적으로 클리어 될수 있음
  - 예) $T_0, T_1, T_2, T_3, T_4$ 와 같은 순차에서 시간 $T_4$ 에서 카운터가 clear 되게함
    등작 기술 → $\boxed{D_3T_4 : SC \leftarrow 0}$
    타이밍 신호 (그림 5-7)

## 타이밍과 제어 (계속)

**Slide 19**



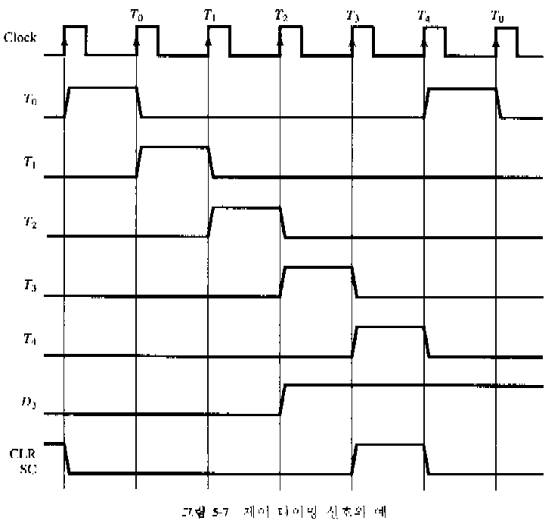



그림 5-7 제어 타이밍 신호의 예

## 타이밍과 제어 (계속)

**Slide 20**

- 타이밍 신호 예)
  - $\boxed{T_0\text{:AR} \leftarrow \text{PC}}$
    $\rightarrow T_0$ 가 1일때 PC 의 내용을 버스에 올리고 AR 에 로드하라
    실제 동작은 다음 클럭의 상승변이에서 일어남

## 명령어 사이클

**Slide 21**

- 기본 컴퓨터에서 명령어 사이클의 단계
  1. 명령어를 메모리에서 가져온다 (fetch)
  2. 명령어를 디코딩한다
  3. 간접 주소방식의 명령어일 경우 메모리로부터 유효주소를 읽어온다
     - 유효주소 : 실제 연산할 데이타가 있는 주소
  4. 명령어를 실행한다
- HALT 명령을 만날때 까지 위의 단계를 반복

## 명령어 사이클 (계속)

**Slide 22**

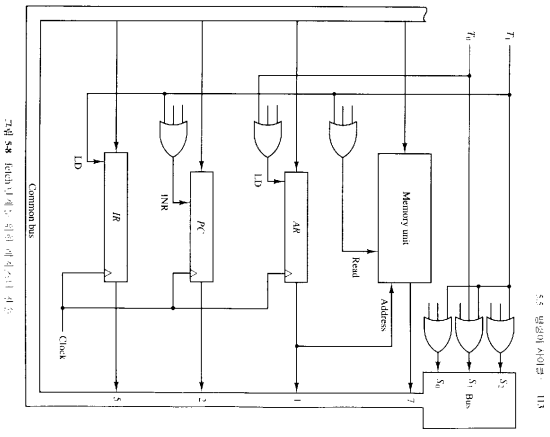
- Fetch 와 디코드
  - 초기 PC 는 프로그램의 첫 명령어를 가리킴
  - 순차 카운터는 0으로 타이밍 변수 $T_0$ 를 가리킴
  - 매 클럭마다 순차 카운터는 증가함
  - fetch 와 디코드에 대한 레지스터 전송문

    $T_0$ : AR ←PC
    $T_1$: IR ←M[AR], PC ←PC + 1
    $T_2$: $D_0, \cdots, D_7$ ←Decode IR(12-14), AR ←IR(0-11), I ←IR(15)

  - $T_0, T_1$ 에서의 전송문의 구현 (그림 5-8)

## 명령어 사이클 (계속)

**Slide 23**




- 명령어 종류의 결정

## 명령어 사이클 (계속)

**Slide 24**

- $T_3$ 에서의 명령어 결정에 대한 흐름도 (그림 5-9)

## 명령어 사이클 (계속)

**Slide 25**



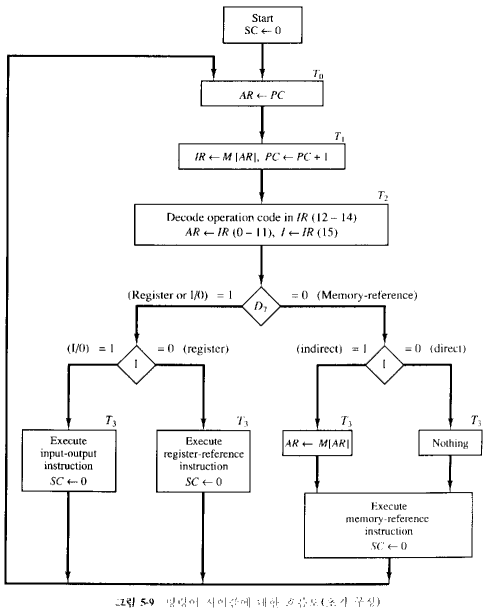



그림 5-9 명령어 사이클에 대한 흐름도(조건 구성)

## 명령어 사이클 (계속)

Slide 26

- $T_3$ 에서의 레지스터 전송문

| |
|---|
| $D_7'IT_3$ : AR ←M[AR] |
| $D_7'I'T_3$ : 아무런 일도 하지 않음 |
| $D_7I'T_3$ : 레지스터 참조 명령어 수행 |
| $D_7IT_3$ : 입출력 명령어 수행 |

- 레지스터 참조 명령어
  - $D_7I'T_3$ 에서 수행됨
  - IR(0-11) 에 있는 비트로 12가지 명령어 구성 (표 5-3)

## 명령어 사이클 (계속)

Slide 27

표 5-3 레지스터 참조 명령어의 수행

$D_7I'T_3 = r$ (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 | $rB_7$: | $AC \leftarrow$ shr $AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$ | Circulate right |
| CIL | $rB_6$: | $AC \leftarrow$ 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 |

## 메모리 참조 명령어

Slide 28

- 메모리 참조 명령어들 (표5-4)

표 5-4 메모리 참조 명령어

| 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 $PC \leftarrow PC + 1$ |

  - $T_4$ 에서 수행됨
  - AND (AND memory word to AC) 명령어

## 메모리 참조 명령어 (계속)

Slide 29

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

  - ADD (Add memory word to AC) 명령어

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

  - LDA (Load memory word to AC) 명령어

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

  - STA (Store content of AC in memory) 명령어

$D_3T_4$ : M[AR] ←AC, SC ←0

  - BUN (Branch unconditionally) 명령어

$D_4T_4$ : PC ←AR, SC ←0

  - BSA (Branch and save return address) 명령어

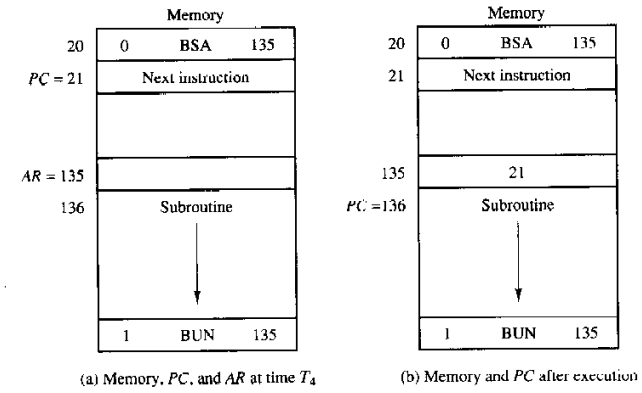
$D_5T_4$ : M[AR] ←PC, AR ←AR + 1
$D_5T_5$ : PC ←AR, SC ←0

## 메모리 참조 명령어 (계속)

Slide 30

* BSA 명령의 실행 예 (그림 5-10)

120 — 제 5 장 기본 컴퓨터의 구조와 설계




(a) Memory, *PC*, and *AR* at time $T_4$ (b) Memory and *PC* after execution

그림 5-10 BSA 명령어의 실행 예

## 메모리 참조 명령어 (계속)

Slide 31

- ISZ (Increment and skip if zero) 명령어

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

- 제어 흐름도
  * 메모리 참조 명령어에 대한 모든 마이크로 연산 (그림 5-11)

## 메모리 참조 명령어 (계속)

Slide 32

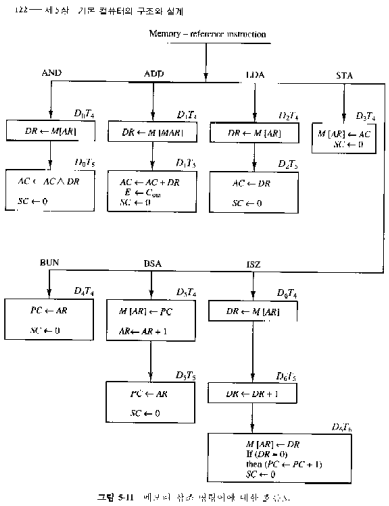



그림 5-11 메모리 참조 명령어에 대한 흐름도.

* 최대의 필요한 타이밍이 $T_6$ 이므로 3비트 순차카운터(SC)로 충분

## 입출력과 인터럽트

Slide 33

- 외부 컴퓨터 사용자와의 데이타 인터페이스를 위해 입출력 장치가 필요함
  - 입력 장치 : 키보드
  - 출력 장치 : 프린터, 모니터
  - 이를 단말 장치 (terminal device) 라고 부름
- 입출력 구성
  - 보통 8비트 단위의 직렬 통신으로 정보교환
  - 입출력 구성 (그림 5-12)

## 입출력과 인터럽트 (계속)

Slide 34



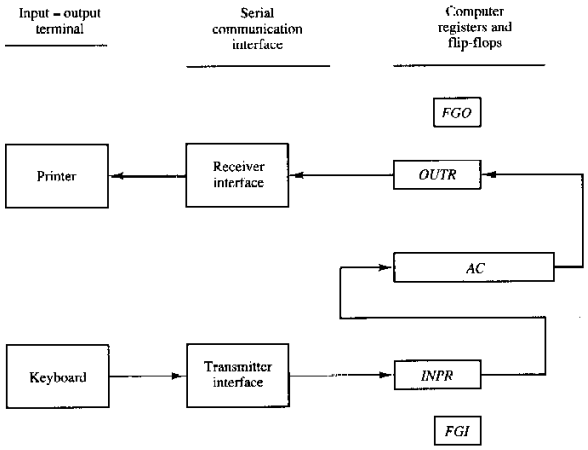



그림 5-12 입출력 구성

## 입출력과 인터럽트 (계속)

Slide 35

- 입력 장치와 컴퓨터 사이의 다른 타이밍 속도의 동기 →flag F/F 이용
  - FGI (flag input)
    - →Keyboard 입력
    - →INPR (input register) 에 데이타 입력, FGI = 1 이됨
    - →컴퓨터는 FGI = 1 이면 데이타 가져오고 0 으로 만듦
    - →FGI = 0 이어야만 새로운 keyboard 입력가능
  - FGO (flag output)
    - →출력할 데이타 발생
    - →FGO = 1 이면 출력 데이타를 OUTR (output register) 로 전송, FGI = 0 으로 만듦
    - →출력장치는 FGO = 0 이면 OUTR 에서 데이타 가져다가 출력 후 FGI = 1 로 만듦
    - →FGI = 1 이어야만 새로운 출력데이타 전송가능

- 입출력 명령어
  - 입출력 명령어들 (표 5-5)

## 입출력과 인터럽트 (계속)

Slide 36

124 — 제 5 장 기본 컴퓨터의 구조와 설계

표 5-5 입출력 명령어

$D_7IT_3 = p$ (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 | $pB_{11}$: | $AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$ | Input character |
| OUT | $pB_{10}$: | $OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$ | 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 |

입출력 명령어

- 프로그램 인터럽트
  - 입출력장치의 속도가 컴퓨터의 속도에 비해 매우 늦음

    →컴퓨터가 데이타가 있는지 레지스터를 점검하는 것은 비 효율적 (polling 방식)

## 입출력과 인터럽트 (계속)

Slide 37

  - 데이타가 외부장치가 데이타 전송이 준비되었을때만 이를 컴퓨터에게 알림

    →인터럽트 방식

    플래그 체크 필요없이 다른 일을 수행하다가

    인터럽트가 걸려오면 데이타를 가져오고 원래 수행중이던 프로그램으로 복귀

    중요한 작업시 인터럽트가 들어오면 안됨

    →인터럽트를 허용할것인지 결정하는 플립플롭 필요

    →인터럽트 인에이블 플립플롭 (interrupt enable flip-flop) IEN 이용

    · 명령어에 의해 set 되거나 clear 될수 있음
  - 인터럽트가 처리되는 순서도 (그림 5-13)

## 입출력과 인터럽트 (계속)

Slide 38



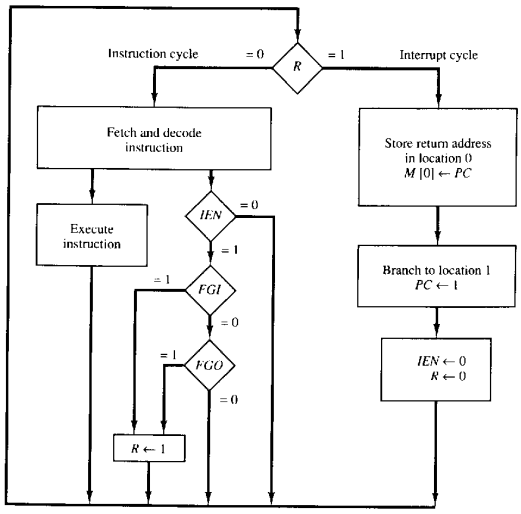



그림 5-13 인터럽트 사이클의 흐름도

## 입출력과 인터럽트 (계속)

– 인터럽트 사이클 수행 과정 (그림 5-14)

Slide 39



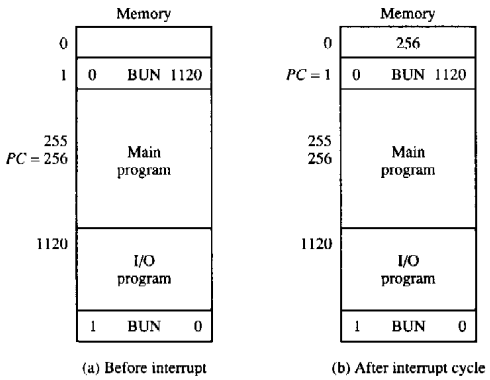



그림 5-14 인터럽트 사이클의 수행 과정

## 입출력과 인터럽트 (계속)

- 인터럽트 사이클
  - 인터럽트 사이클은 IEN=1 이고 FGO 나 FGI 가 1일때 수행 (R=1)
  - 타이밍 신호 $T_0, T_1, T_2$ 외에 모든 타이밍 신호에서 수행됨

**Slide 40**

  - 레지스터 전송문 $T_0' T_1' T_2' (IEN)(FGO + FGI)$ : R $\leftarrow$1
  - 그러므로 기 기술한 명령어의 조건은 $R' T_0, R' T_1, R' T_2$ 로 수정됨
  - 인터럽트 사이클에서의 수행 레지스터 전송문

$RT_0$ : 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

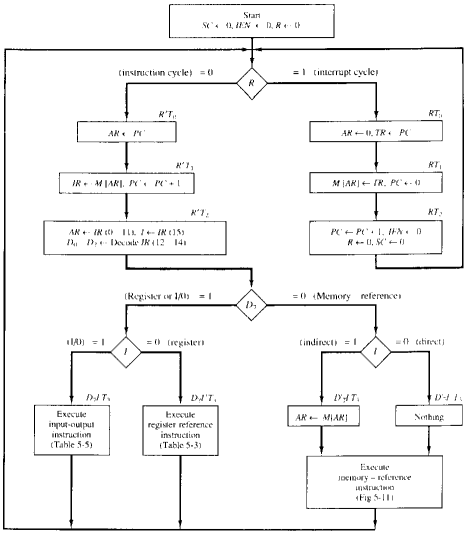
## 컴퓨터에 대한 와전한 기술

**Slide 41**

- 인터럽트 사이클과 명령어 사이클을 포함한 기본 컴퓨터의 최종 흐름도 (그림 5-15)

## 컴퓨터에 대한 와전한 기술 (계속)

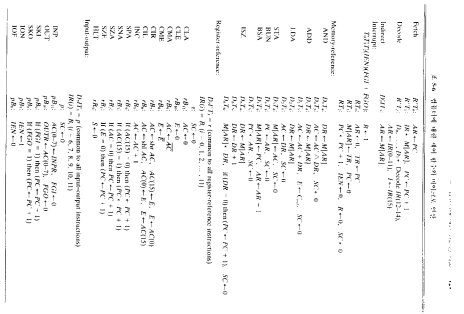
**Slide 42**




## 컴퓨터에 대한 와전한 기술 (계속)

**Slide 43**

- 기본 컴퓨터의 레지스터 전송문 (표 5-6)



## 기본 컴퓨터의 설계

- 설계시 필요한 하드웨어 요소

1. 16비트의 4096 워드를 가진 메모리
2. 9개의 레지스터: AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
3. 7개의 플립플롭: I, S, E, R, IEN, FGI, FGO
4. 2개의 디코더 : 3 × 8 등작 디코더 , 4 × 16 타이밍 디코더

**Slide 44**

5. 16비트 공통 버스
6. 제어 논리 게이트들
7. AC의 입력에 연결된 가산 논리회로

- 제어 논리 게이트
  - 그림 5-6의 제어논리게이트의 필요한 출력들
    1. 9개의 레지스터의 입력을 제어하는 신호
    2. 메모리의 쓰기 및 읽기 입력을 제어하는 신호
    3. 플립플롭을 세트, 클리어, 보수화시키는 신호

## 기본 컴퓨터의 설계 (계속)

4. 버스를 사용할 레지스터를 선택하는데 사용되는 $S_2, S_1, S_0$ 에 대한 신호
5. AC 에 대해 가산 논리 회로를 제어하는 신호

- 레지스터와 메모리에 대한 제어
  - 레지스터에 대한 제어입력들
    - LD (로드)
    - INR (인크리멘트)

**Slide 45**

    - CLR (클리어)
  - AR 레지스터의 제어입력 구성
    - 표 5-6에서 AR 의 내용을 변경시키는 모든 문장을 찾음

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

## 기본 컴퓨터의 설계 (계속)

* 제어입력별로 구분

$$\text{LD(AR)} = R'T_0 + R'T_2 + D_7'IT_3$$
$$\text{CLR(AR)} = RT_0$$
$$\text{INC(AR)} = D_5T_4$$

* 제어입력별로 구현 (그림 5-16)

Slide 46

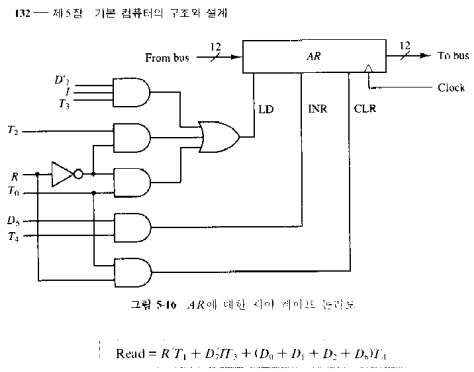



그림 5-16 AR에 대한 레지스터 제어 논리도

Read = $RT_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$

## 기본 컴퓨터의 설계 (계속)

– AR 과 같은 방법으로 다른 레지스터들도 제어입력을 만듦

* 예) 메모리 읽기

$$\text{Read} = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$$

Slide 47

- 단일 플립플롭에 대한 제어

– 7개의 단일 플립플롭도 위와 유사하게 구할수 있다.

– IEN 에 대한 제어입력 구성

$pB_7$ : IEN ←1
$pB_6$ : IEN ←0
$RT_2$ : IEN ←0

– IEN 에 대한 제어입력 구성 (JK 플립플롭 이용) (그림 5-17)

## 기본 컴퓨터의 설계 (계속)

**Slide 48**



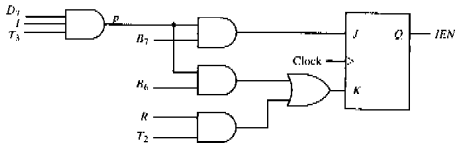



그림 5-17 *IEN*에 대한 제어 입력

- 공통 버스에 대한 제어
  - 공통버스는 선택입력 $S_2, S_1, S_0$ 에 의하여 제어됨 (표 5-7)

## 기본 컴퓨터의 설계 (계속)

**Slide 49**

따라서 $x_i$에 대한 부울식은 다음과 같이 나타낼 수 있다.

표 5-7 버스의 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 |

- 표 5-7 에서 x 는 입력변수
- 표 5-7 은 이진 인코더에 해당
  - ∗ 인코더에 대한 부울식

## 기본 컴퓨터의 설계 (계속)

$$\boxed{\begin{aligned} S_0 &= x_1 + x_3 + x_5 + x_7 \\ S_1 &= x_2 + x_3 + x_6 + x_7 \\ S_2 &= x_4 + x_5 + x_6 + x_7 \end{aligned}}$$

**Slide 50**

* $x_1$ 을 1 로 만드는 논리표 (즉 AR 이 버스에 실리게 하는 논리표)
  - 표 5-6 에서 구함

$$\boxed{\begin{aligned} &D_4T_4 : \text{PC} \leftarrow \text{AR} \\ &D_5T_5 : \text{PC} \leftarrow \text{AR} \end{aligned}}$$

  - 결국 $\boxed{x_1 = D_4T_4 + D_5T_5}$
  - 다른 $x$ 도 이와같이 구함

## 누산기 논리의 설계

- AC 레지스터와 관련된 주변회로 (그림 5-19)

**Slide 51**

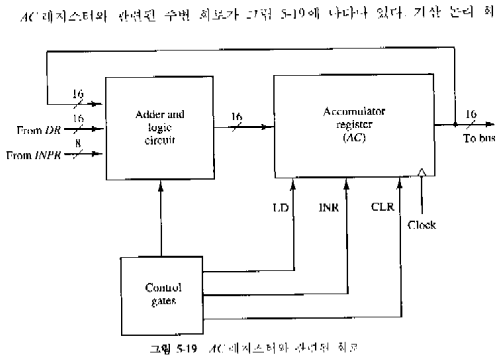



그림 5-19 AC 레지스터와 관련된 회로

  - 세가지 입력 (LD, INR, CLR)
  - 표 5-6 으로 부터 AC 와 관련된 제어입력 찾음

## 누산기 논리의 설계 (계속)

**Slide 52**

| |
|---|
| $D_0T_5$ : AC ←AC ∧ DR |
| $D_1T_5$ : AC ←AC + DR |
| $D_2T_5$ : AC ←DR |
| $pB_{11}$ : AC(0-7) ←INPR |
| $rB_9$ : AC ←$\bar{AC}$ |
| $rB_7$ : AC ←$shr$ AC, AC(15) ←E |
| $rB_6$ : AC ←$shl$ AC, AC(0) ←E |
| $rB_{11}$ : AC ←0 |
| $rB_5$ : AC ←AC + 1 |

- AC 레지스터에 대한 제어
  - 구해진 제어입력을 구현 (그림 5-21)

## 누산기 논리의 설계 (계속)

**Slide 53**

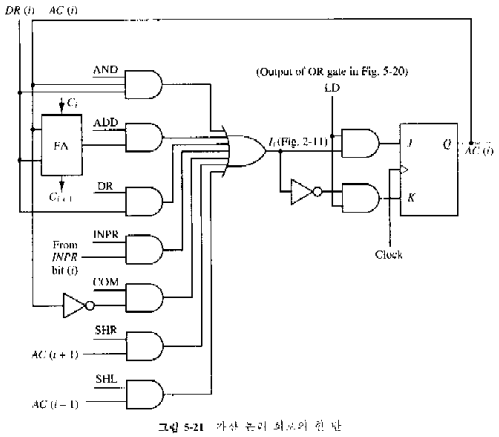



그림 5-21 [illegible]