Microprocessor (W9)
- Control Unit1 -

Dong Min Kim
Department of IoT
Soonchunhyang University
dmk@sch.ac.kr

Contents

01 제어 장치의 기능

❑ 컴퓨터의 기본 구성

- **제어 장치** : 컴퓨터의 모든 동작을 제어하는 CPU의 핵심 장치
- ALU, I/O 장치에 프로세서가 전송한 명령어 수행
- 주기억 장치의 명령어를 읽어 CPU의 명령 레지스터 IR로 가져오고,
- 명령 레지스터의opcode를 해독하여 제어 신호를 발생

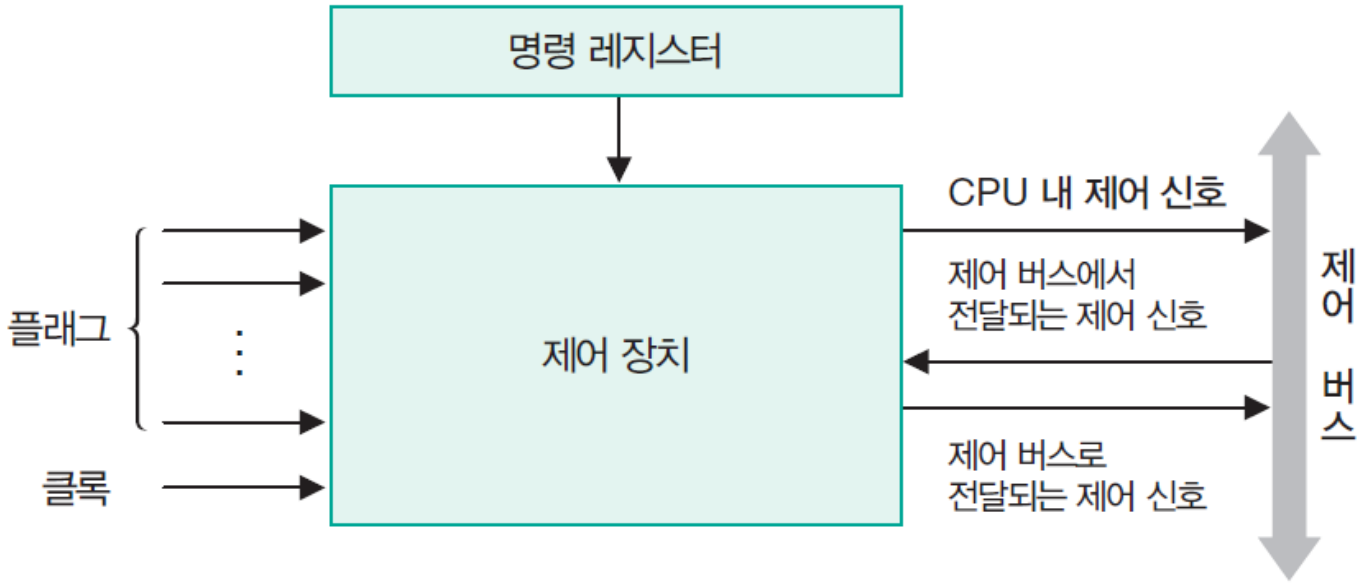


그림 5-1 제어 장치의 기능

❑ 제어 장치의 기본 기능

- CPU에 접속된 장치들에 대한 데이터 이동 순서 조정
- 명령어 해독
- CPU 내 데이터 흐름 제어
- 외부 명령을 받아 일련의 제어 신호 생성
- 실행 장치(예를 들어 ALU, 데이터 버퍼, 레지스터) 제어
- 명령어 인출, 명령어 해독, 명령어 실행 등을 순서에 맞추어 처리

02 제어 장치의 종류

1 하드와이어 제어 장치 (hardwired control)

- **논리 회로**로 만들어진 **하드웨어**로 명령어 실행 제어에 필요한 제어 신호 발생
- 회로 구조를 물리적으로 변경하지 않고는 신호 생성 방법을 수정할 수 없음
- 명령어의 opcode에 제어 신호를 생성하는 기본 데이터 포함
- 명령 디코더에서 명령 코드가 해독
- 명령 디코더는 명령어 opcode에 정의된 여러 필드를 해독하는 많은 해독기 세트로 구성

02 제어 장치의 종류

- 제어 하드웨어는 **상태 기계**(state machine)처럼 클록 사이클이 진행됨에 따라 상태가 변함
- 명령 레지스터, 상태 코드, 외부 입력 등에 따라 다르게 변함
- 프로그래밍 가능한 논리 배열(PLA)과 유사한 방식으로 구성
  - 논리식으로 설계한 고정된 논리 회로에 의해 제어 신호 생성
  - 하드와이어 제어 장치는 마이크로 프로그램 제어 장치보다 빠름
  - 고속으로 작동

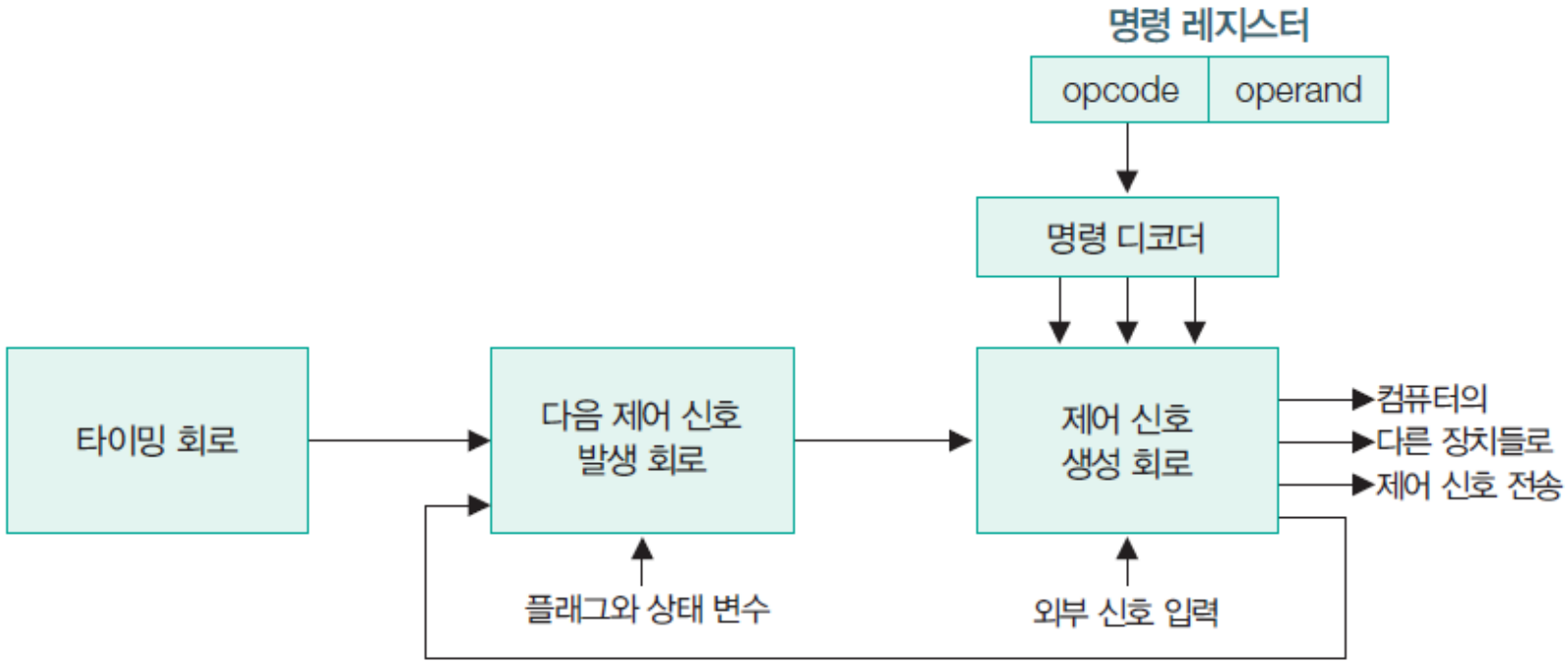


그림 5-2 하드와이어 제어의 개념

02 제어 장치의 종류

- 명령어 실행을 위한 제어 신호는 명령어 실행 사이클 전 구간에서 끊임없이 생성
- 인출 초기 : 새 명령이 제어 장치에 도착
- 명령어 해독 : 새 명령 실행과 관련된 첫 번째 상태 시작
  - 컴퓨터의 플래그 및 상태 정보가 변경되지 않은 상태로 유지되는 한 지속
  - 이들 신호 중 어떤 것이라도 변경이 일어나면 제어 장치 상태도 변경이 일어남
- 제어 신호 발생 회로에 대해 각각의 새 입력이 생성
  - 외부 신호가 나타날 때(예: 인터럽트) 외부 신호에 대한 반응(예: 인터럽트 처리)과 관련된 상태가 됨
  - 컴퓨터의 플래그와 상태 변수 값은 명령어 실행 주기에 적합한 상태를 선택하는 데 사용
- 사이클의 마지막 상태 : 프로그램의 다음 명령을 가져오기 시작하는 제어 상태
  - 프로그램 카운터의 내용을 MAR로 보내고
  - 컴퓨터의 명령 레지스터에 대한 명령어를 읽음
  - 프로그램 실행 종료 명령 : 운영체제 상태로 돌아감

2 마이크로 프로그램 제어 장치 (micro-programmed control)

- 하드와이어 제어 장치 사이의 근본적인 차이: 제어 메모리 control memory의 존재 여부
- 각 명령 실행 순서에 해당하는 일련의 해독된 제어 신호를 비트 패턴으로 만들어 제어 메모리에 저장
- 명령 인출과정은 하드와이어 제어 장치와 동일
- 그러나 각 명령의 opcode가 제어 신호를 생성하기 위해 제어 메모리에서 해당 마이크로 프로그램의 시작 주소 지시
  - 명령 레지스터의 opcode가 제어 메모리의 제어 주소 레지스터CMAR로 전송되고,
  - 마이크로 프로그램의 첫 번째 마이크로 명령이 마이크로 명령 레지스터로 읽힌다.
  - 마이크로 명령 : 인코딩된 형태로 연산코드 제공
  - 마이크로 명령 해독기에서 연산 코드 필드 해독

02 제어 장치의 종류

- 마이크로 명령 : 다음 마이크로 명령의 주소 포함
- 하나의 제어 필드는 마이크로 명령 주소 발생 장치를 제어하는 신호로 사용
- 마이크로 명령 주소 발생 장치 : 마이크로 명령 루틴의 시작 주소 생성
- 주소 발생 장치는 사상 함수에 opcode를 연결하여 루틴의 시작 주소를 생성
  - 사상 함수는 명령어 개수가 적고 체계적일 때는 효율적이지만, 명령어 개수가 많아지고 이전 버전 명령 세트에 계속적으로 추가될 때는 점점 더 복잡해짐
  - 주소 발생 장치는 사상 함수뿐만 아니라 덧셈이나 기타 다른 연산을 추가하여 주소를 찾아냄

02 제어 장치의 종류

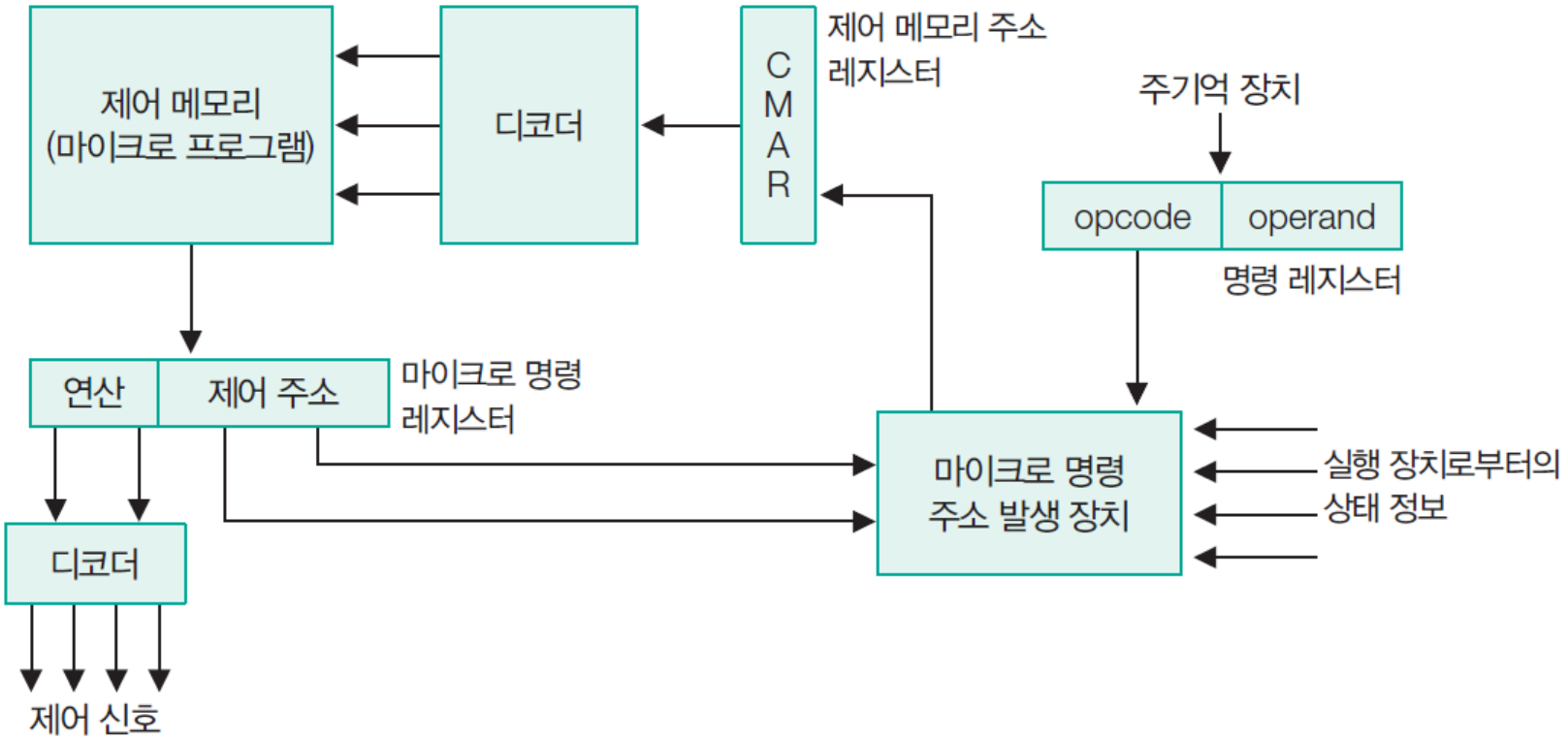


그림 5-3 마이크로 프로그램 제어의 구성과 제어 흐름

02 제어 장치의 종류

❖ 마이크로 명령 설계 순서

❶ 필요한 제어 신호의 목록 작성

❷ 유사한 의미를 가진 제어 신호를 그룹화하여 필드 구성

❸ 각 필드에 논리적인 순서 부여 : 예를 들어 ALU 연산과 ALU 피연산자를 앞에 두고 이어서 실행될 마이크로 명령의 제어 메모리 주소를 뒤에 둔다

❹ 마이크로 명령 형식에 대한 기호 범례를 작성하여 필드 값의 이름과 제어 신호 설정 방법 작성

❺ 필드 폭을 최소화하기 위해 동시에 사용하지 않는 작업을 분리하여 인코딩

- 마이크로 명령을 설계한 후 이를 기반으로 명령어 사이클인 명령어 인출, 명령어 해독, 명령어 실행 각각에 대한 마이크로 프로그램 루틴 작성
- 명령어 실행(연산) 사이클은 모든 기계어 명령마다 다르므로 일일이 정의

02 제어 장치의 종류

- 제어 신호는 프로그래머가 접근할 수 없는 제어 메모리에 제어 워드로 저장
- 프로그램으로 생성되는 제어 신호는 기계어와 유사
- 제어 기억 장치에서 마이크로 명령을 읽어 오기 때문에 속도가 느림

표 5-1 주요 용어

용어	설명
제어 워드	개별 비트가 다양한 제어 신호를 나타내는 단어다.
마이크로 루틴	기계어 제어 순서에 해당하는 일련의 제어 워드는 해당 명령의 마이크로 루틴을 구성한다.
마이크로 명령	마이크로 루틴 내의 개별 제어 워드를 마이크로 명령이라고 한다.
마이크로 프로그램	일련의 마이크로 명령을 마이크로 프로그램이라 하고, ROM 또는 RAM에 저장되며 이 장소를 제어 기억 장치Control Memory, CM라고 한다.
제어 기억 장치	모든 기계어 명령에 대한 마이크로 루틴을 저장하는 특수 기억 장치를 제어 기억 장치라고 한다.

3 마이크로 프로그램 제어 장치의 종류

❑ 수평적 마이크로 프로그램

- 제어 신호가 제어 신호당 1비트로 해독된 2진 형식으로 표현
  - 예를 들어 프로세서에 제어 신호가 50개 있다면 50비트의 제어 신호 필요
  - 한번에 2개 이상의 제어 신호를 활성화할 수 있음
- 수평적 마이크로 프로그램의 특징
  - 제어 워드가 더 길다.
  - 병렬 처리 응용에 사용된다.
  - 더 높은 수준의 병렬 처리 가능 : 병렬성이 n이면 한 번에 n개의 제어 신호 활성화 된다.
  - 추가 하드웨어(디코더)가 필요치 않음 ⇨ 수직적 마이크로 프로그래밍보다 빠름

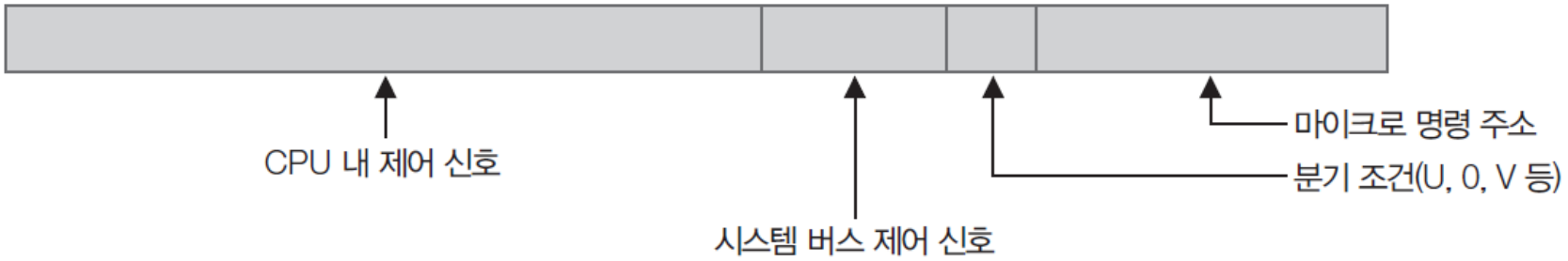


그림 5-4 수평적 마이크로 명령어의 구조 예

❑ 수직적 마이크로 프로그램

- 수직적 마이크로 프로그램에서는 제어 신호가 인코딩된 2진 형식으로 표시
- N개 제어 신호가 필요할 경우 $\lceil \log_2 N \rceil$비트 필요
- 수직적 마이크로 프로그램의 특징
  - 제어 워드가 더 짧다.
  - 유연하므로 새로운 제어 신호를 추가하기 용이
  - 낮은 수준의 병렬화 허용
  - 제어 신호를 생성하는 추가적인 하드웨어 필요 ⇨ 수평적 마이크로 프로그래밍보다 느림

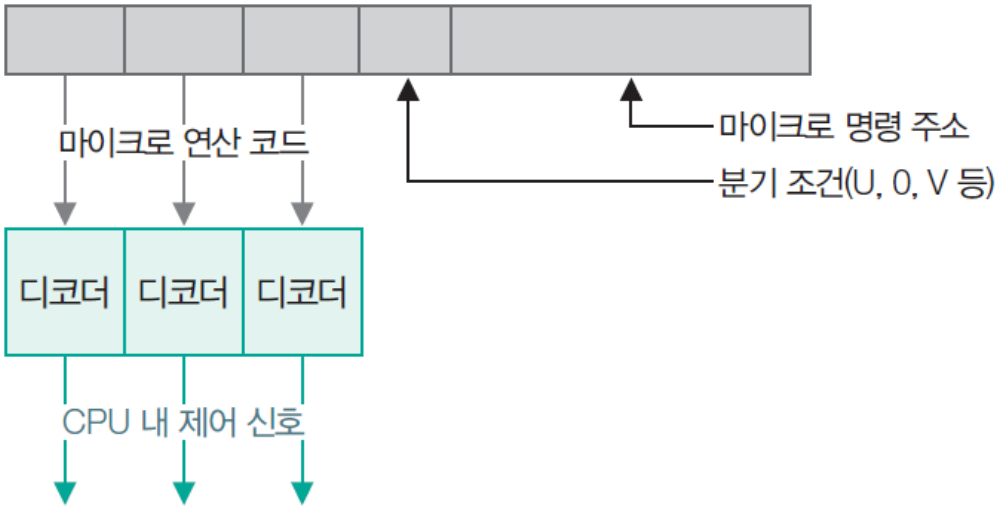


그림 5-5 수직적 마이크로 명령어의 구조 예

4 하드와이어 제어와 마이크로 프로그램 제어 비교

표 5-2 하드와이어 제어와 마이크로 프로그램 제어 비교

하드와이어 제어	마이크로 프로그램 제어
회로 기반 기술을 사용한다.	소프트웨어 기반 기술을 사용한다.
플립플롭, 게이트, 디코더 등을 사용하여 구현한다.	마이크로 명령이 명령의 실행을 제어하는 신호를 발생시킨다.
고정 명령 형식이다.	가변 명령 형식이다(명령당 16~64비트).
레지스터 기반 명령이다.	레지스터 기반이 아닌 명령이다.
ROM이 사용되지 않는다.	ROM이 사용된다.
RISC에서 주로 사용된다.	CISC에서 주로 사용된다.
해독이 빠르다.	해독이 느리다.
변경이 어렵다.	변경이 쉽다.
칩 영역이 작다.	칩 영역이 크다.

02 제어 장치의 종류

- 하드와이어 제어와 마이크로 프로그램 제어를 적절히 혼용하여 hybrid 제어 장치 구성

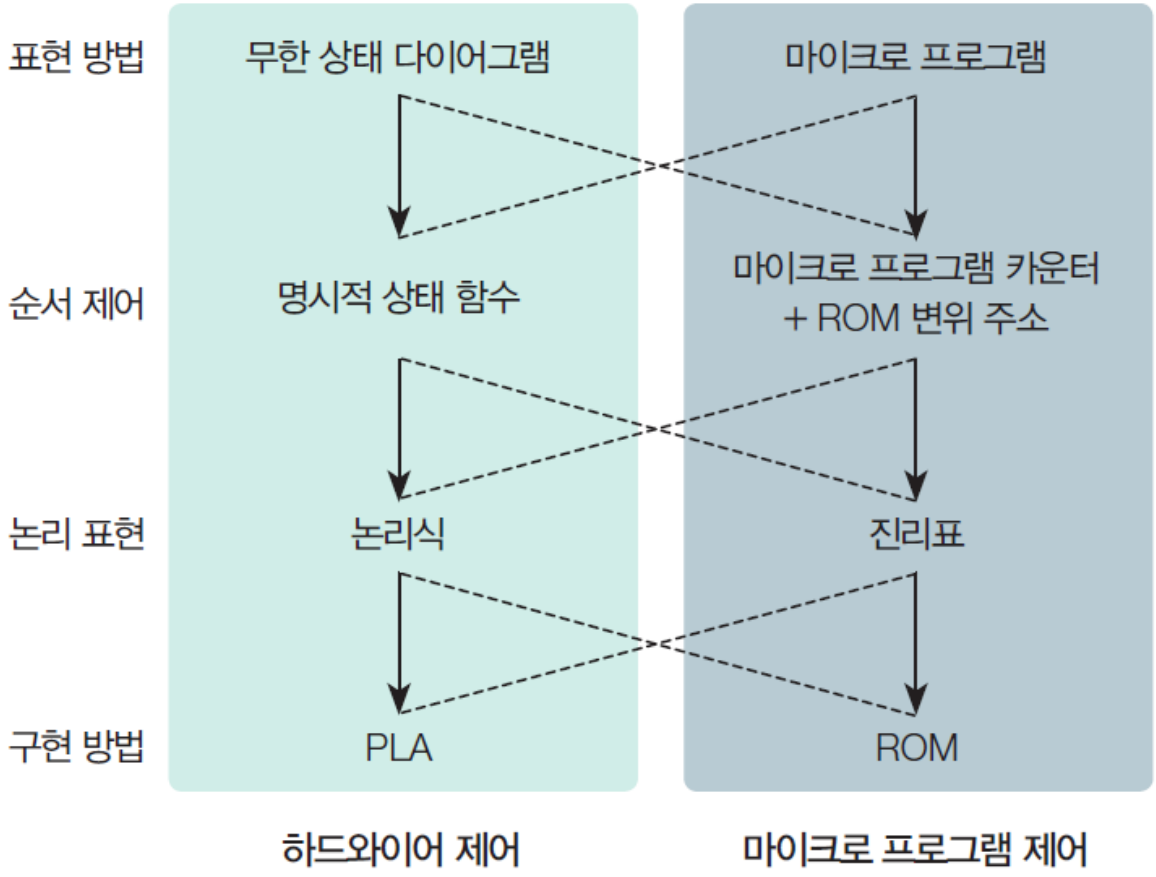


그림 5-6 하이브리드 제어의 구성 방법

❑ 명령어 사이클

- **명령어 인출** → **명령어 해독** → **명령어 실행** 사이클로 진행
- 명령어 사이클: 데이터 경로(data-path) 사이클이라고도 함
- 인터럽트 사이클
  - 인터럽트 사이클은 매 명령어 사이클이 끝나고 인터럽트 유무를 점검
  - 인터럽트가 있으면 인터럽트 처리 루틴 실행

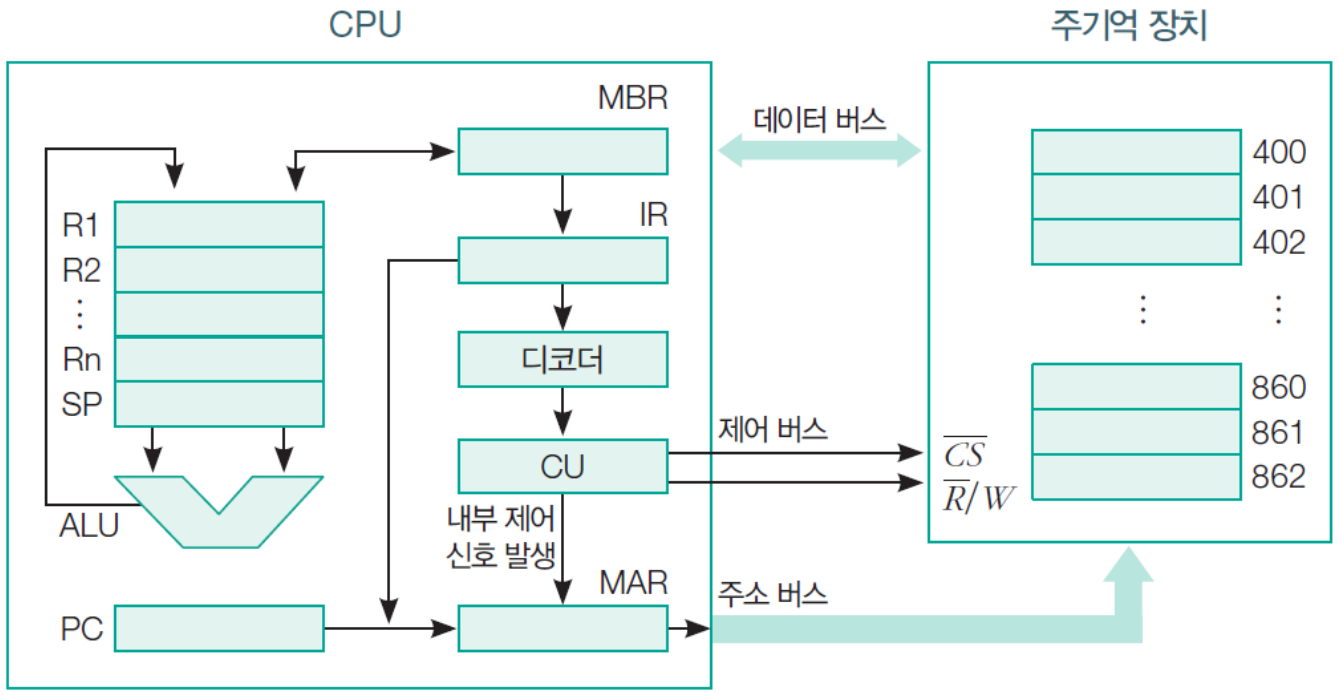


그림 5-7 CPU와 기억 장치의 제어 및 데이터 흐름

❑ 명령어 사이클(계속)

- 하나의 기계어 명령 : 일련의 마이크로 명령으로 구성된 명령어 사이클을 이루어 실행
- 각 명령어 사이클은 여러 개의 작은 단위로 구성

  - 명령어 인출, 명령어 해독, 명령어 실행, 인터럽트로 구별

- 제어 장치를 설계하려면 더 작은 단위의 마이크로 연산으로 분할(그림 5-8)

  - 각 명령은 더 짧은 하위 사이클(예 : 명령어 인출, 명령어 해독, 명령어 실행, 인터럽트)로 구성된 명령 주기 동안 실행
  - 각 하위 사이클은 하나 이상의 마이크로 연산을 가짐
  - 마이크로 연산은 프로세서의 가장 작은 동작

03 명령어 사이클

❖ **명령어 사이클 흐름도**

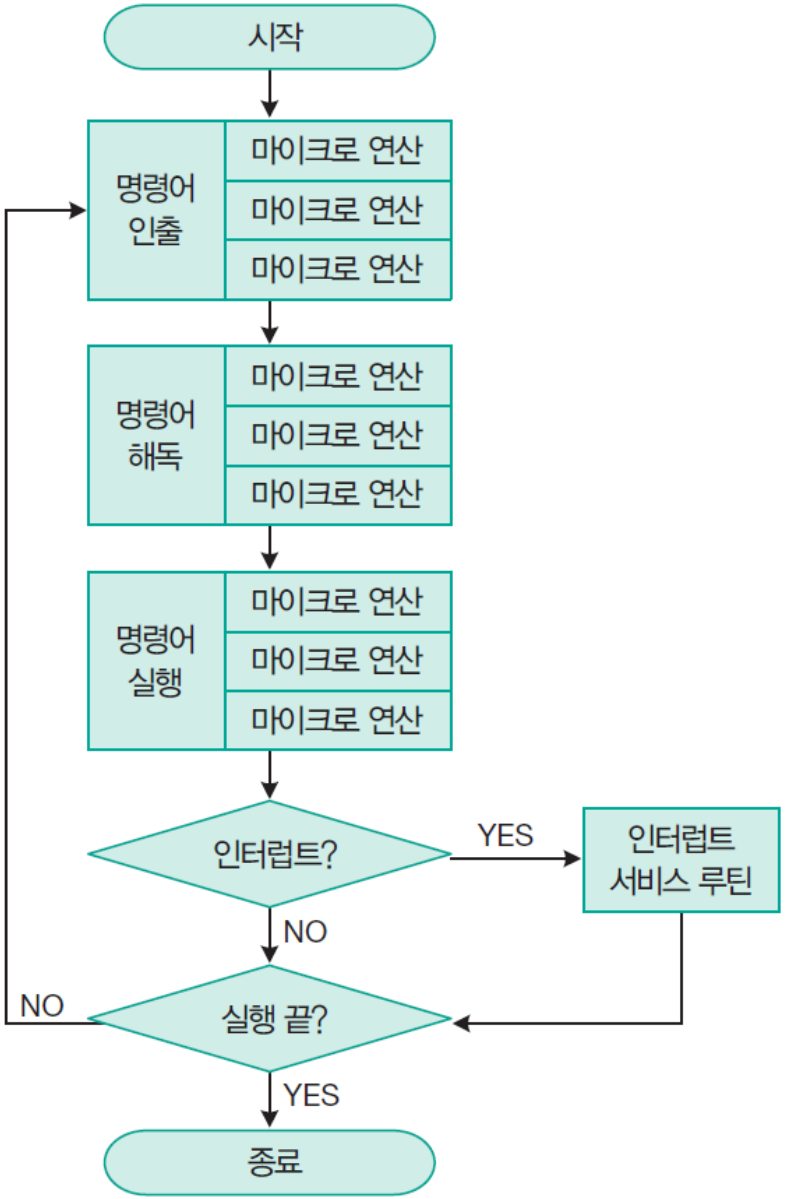


그림 5-8 명령어 사이클 흐름도

1 명령어 인출 사이클: instruction fetch

- 명령어 인출 사이클은 모든 명령어 실행의 첫 번째 단계
- 다음에 실행할 명령어를 주기억 장치에서 읽어 오는 과정

t_1 : PC ➔ MAR ➔ 주소 버스

t_2 : M[MAR] ➔ 데이터 버스 ➔ MBR

t_3 : MBR ➔ 명령 레지스터(IR)

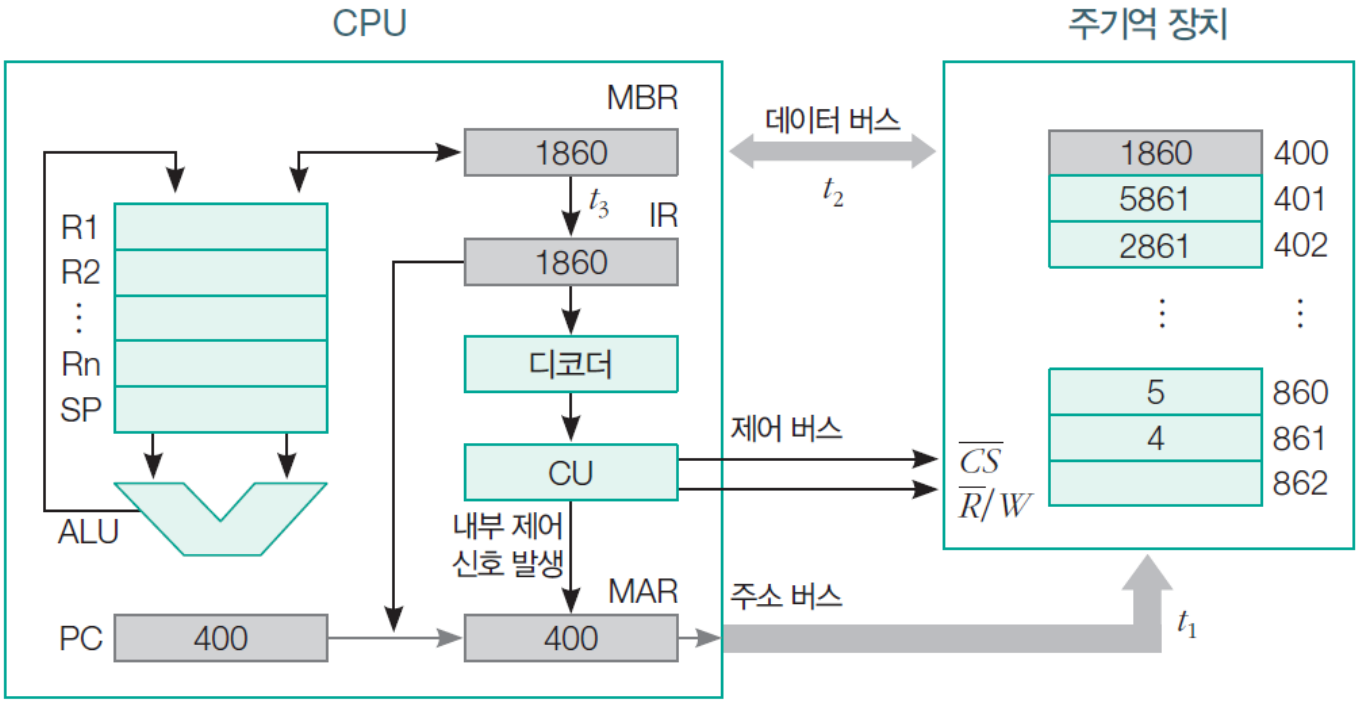


그림 5-9 명령어 인출 사이클의 마이크로 명령 실행 과정

03 명령어 사이클

- 명령어 인출 사이클이 진행되는 동안 프로세서의 레지스터 변화 과정

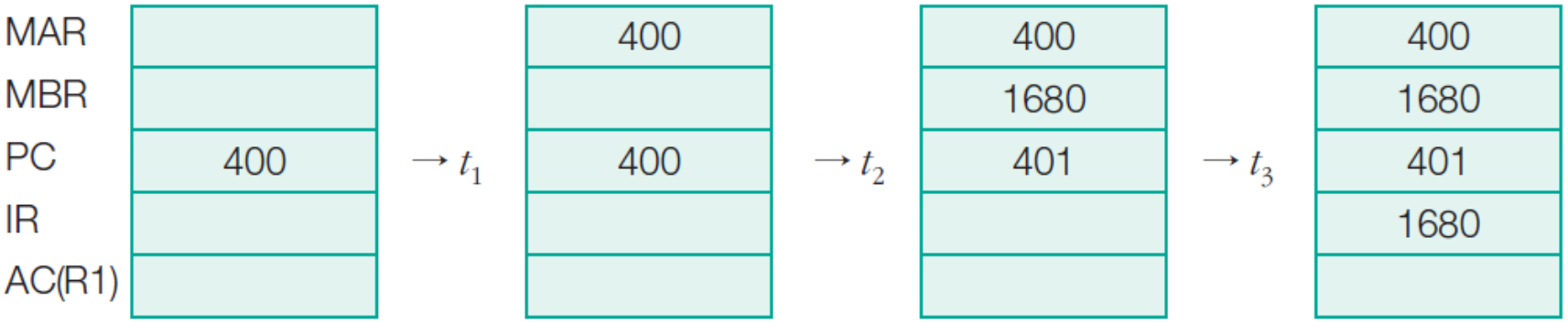


그림 5-10 명령어 인출 사이클의 진행에 따른 레지스터 변화

t_1: MAR ← (PC)

t_2: MBR ← M[MAR], PC ← (PC) + I

t_3: IR ← (MBR)

- I : 명령어의 크기
- 바이트 단위 주소 지정이고, 명령어의 기본 크기가 2 바이트라면 I는 2가 됨

03 명령어 사이클

- 마이크로 연산 $PC \leftarrow (PC) + I$는 t_2, t_3 중 어느 것과도 출동이 발생하지 않으므로 둘 중 어디에서 실행되어도 무관
- 마이크로 연산을 그룹으로 묶을 때는 다음 두 가지 간단한 규칙을 따라야 한다.

❶ 연산의 순서 준수

(MAR ← (PC))는 반드시 MAR의 주소를 사용하기 때문에 (MBR ← 주기억 장치) 앞에 와야 함

❷ 충돌을 피해야 함

동시에 동일한 레지스터에서 읽고 쓰려고 해서는 안 됨

예: 마이크로 연산 (MBR ← 주기억 장치)와 (IR ← MBR)은 동시에 실행되지 않아야 함

- 마이크로 연산 중 하나가 덧셈 연산 수행 : ALU의 기능과 프로세서 구조에 따라 덧셈 마이크로 연산을 ALU가 수행할 수도 있음

2 명령어 해독 사이클: instruction decode

- 명령어 해독 사이클 : 명령 레지스터IR의 내용 중 opcode만 해독기로 전달
- 해독기는 제어 기억 장치에서 명령 연산에 해당되는 마이크로 루틴을 찾아 해독
- 해독된 명령어에 대한 후속 마이크로 연산 발생

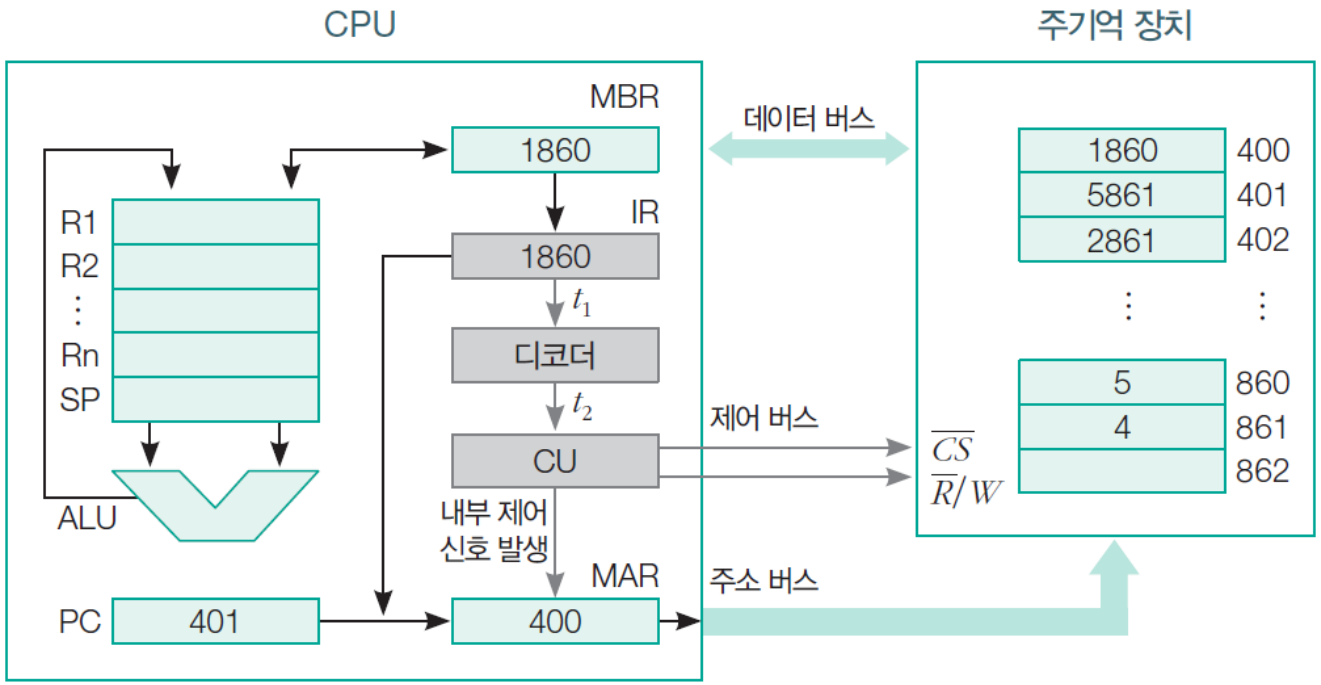


그림 5-11 명령어 해독 사이클의 마이크로 명령어 실행 과정

t_1: Decoder ← (IR:opcode) ; 여기서 opcode는 1이라고 가정한다.

t_2: Instruction Decoding ; CU가 명령어를 해독하여 제어 신호를 발생한다.

3 명령어 실행 사이클: instruction execute

- 명령어 실행 사이클: 해독된 명령어 실행 사이클
- 예: 데이터를 읽어서 레지스터 R1에 저장하는 명령어 실행 사이클
  - IR : operand ➔ MAR ➔ 주소 버스
  - M[MAR] ➔ 데이터 버스 ➔ MBR
  - MBR ➔ R1

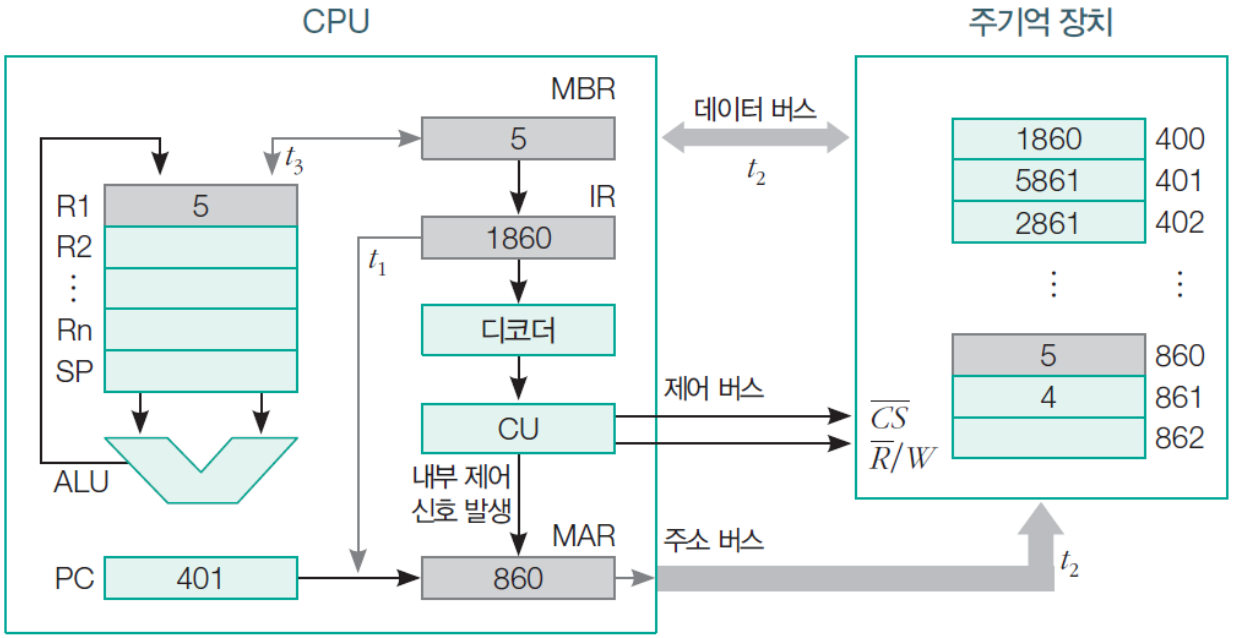


그림 5-12 명령어 실행 사이클의 마이크로 명령 실행 과정

t_1: MAR ← (IR:operand) ; 명령 레지스터의 오퍼랜드(860)를 MAR로

t_2: MBR ← M[MAR] ; 메모리에서 피연산자를 읽어 MBR로

t_3: R1 ← (MBR) ; MBR에서 R1(누산기 역할)로

03 명령어 사이클

- 명령어 인출 사이클이 진행되는 동안 프로세서의 레지스터 변화 과정

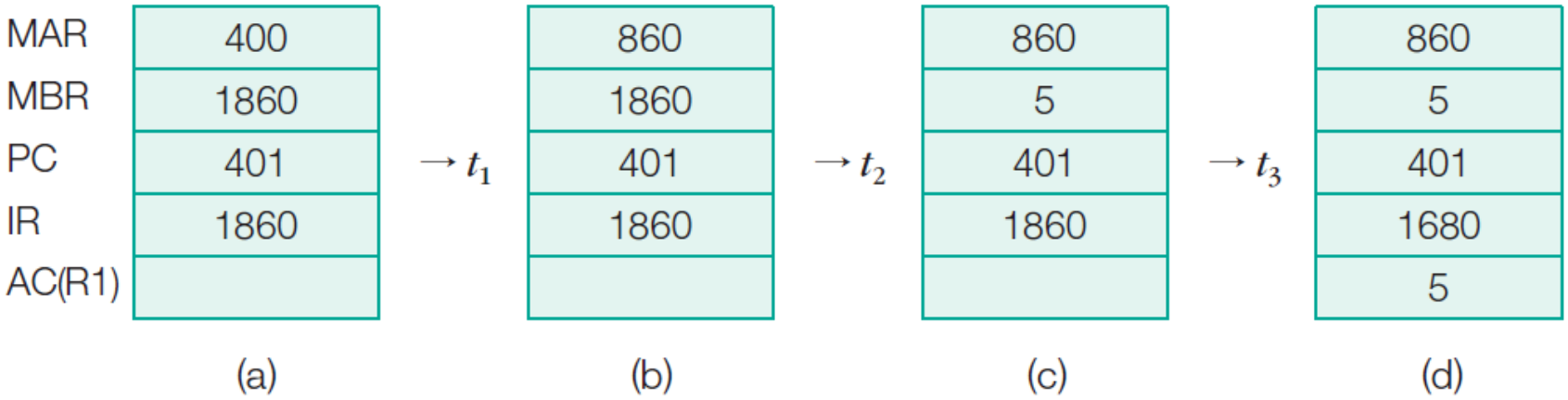


그림 5-13 명령어 실행 사이클의 진행에 따른 레지스터 변화

03 명령어 사이클

- 명령어 실행 사이클은 명령어 세트의 개수만큼 아주 다양하게 존재
- 명령 인출과정은 동일
- 레지스터 R1의 데이터와 메모리 X번지의 데이터를 ALU에서 더해 다시 R1에 저장

ADD R1, X

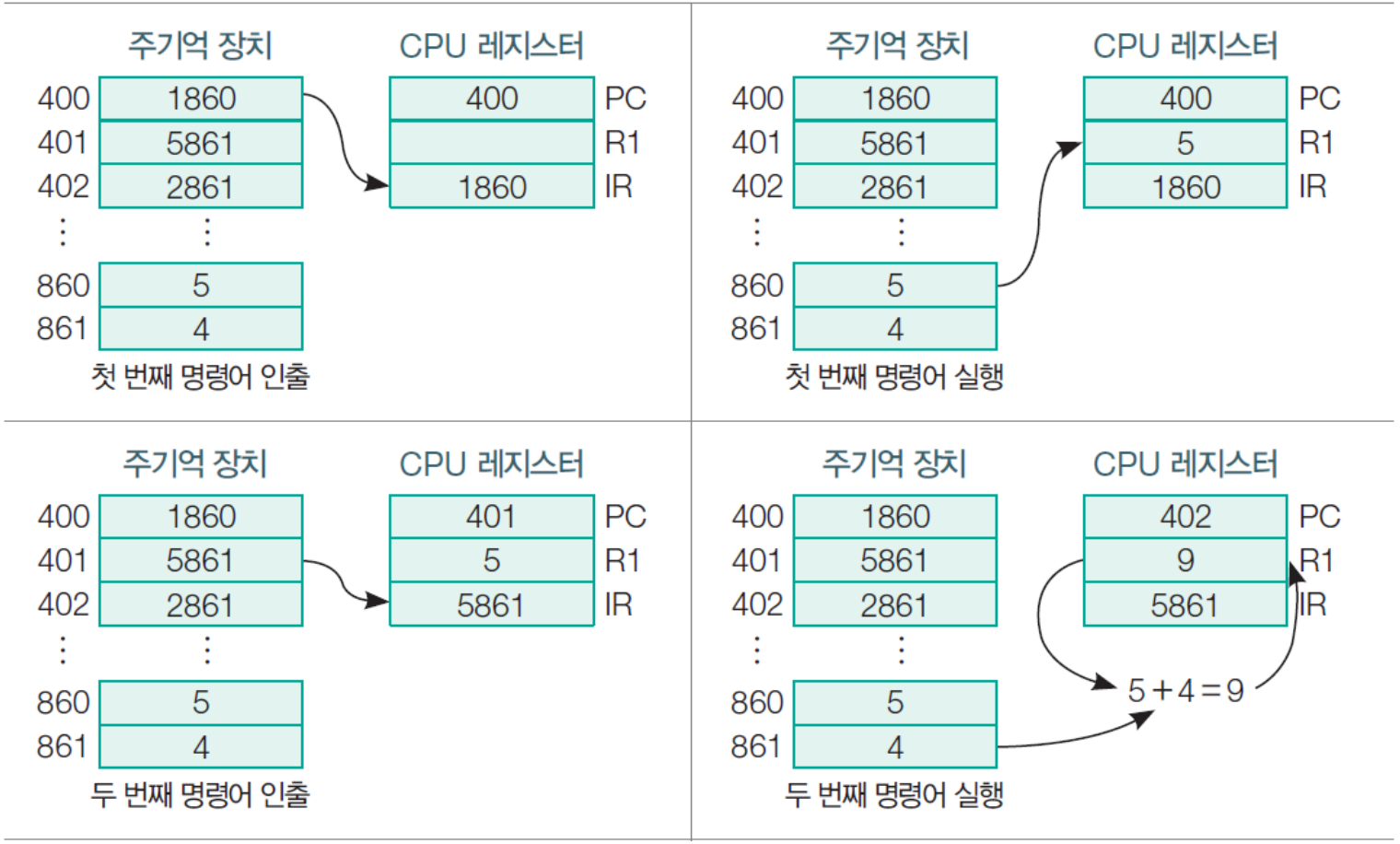
t_1: MAR ← (IR:X)

t_2: MBR ← M[MAR]

t_3: R1 ← (R1) + (MBR)

❑ LOAD(1), ADD(5), STORE(2) 3개의 명령이 실행되는 과정

- CPU내의 PC, R1, IR 등의 레지스터 내용 변화 주목



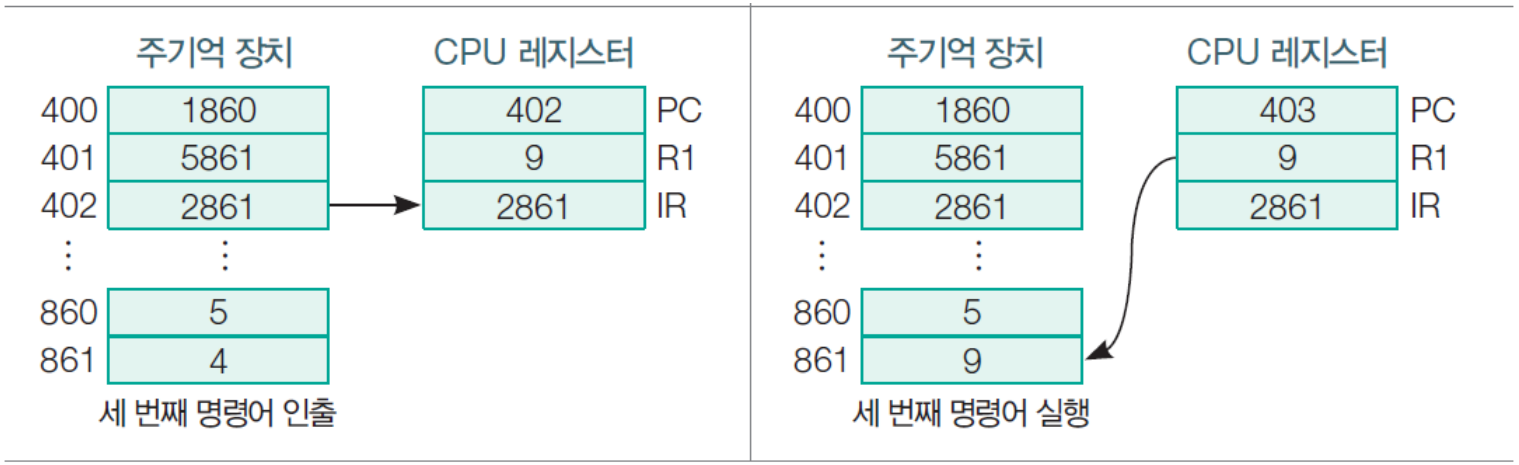


그림 5-14 LOAD, ADD, STORE가 실행되는 과정에서 레지스터 변화

예: ISZ(Increment and Skip if Zero)

- X 값을 1 증가시키고 그 결과가 0이면 바로 다음 명령을 건너뜀

ISZ X

t_1: MAR ← (IR:X)

t_2: MBR ← M[MAR] + 1

t_3: M[MAR] ← (MBR), If ((MBR) = 0) then (PC ← PC + I)

예: BSA(Branch-and-Save-Address)

BSA X

t_1: MAR ← (SP), MBR ← (PC)

t_2: M[MAR] ← (MBR), SP ← SP − I

t_3: PC ← (IR:X)

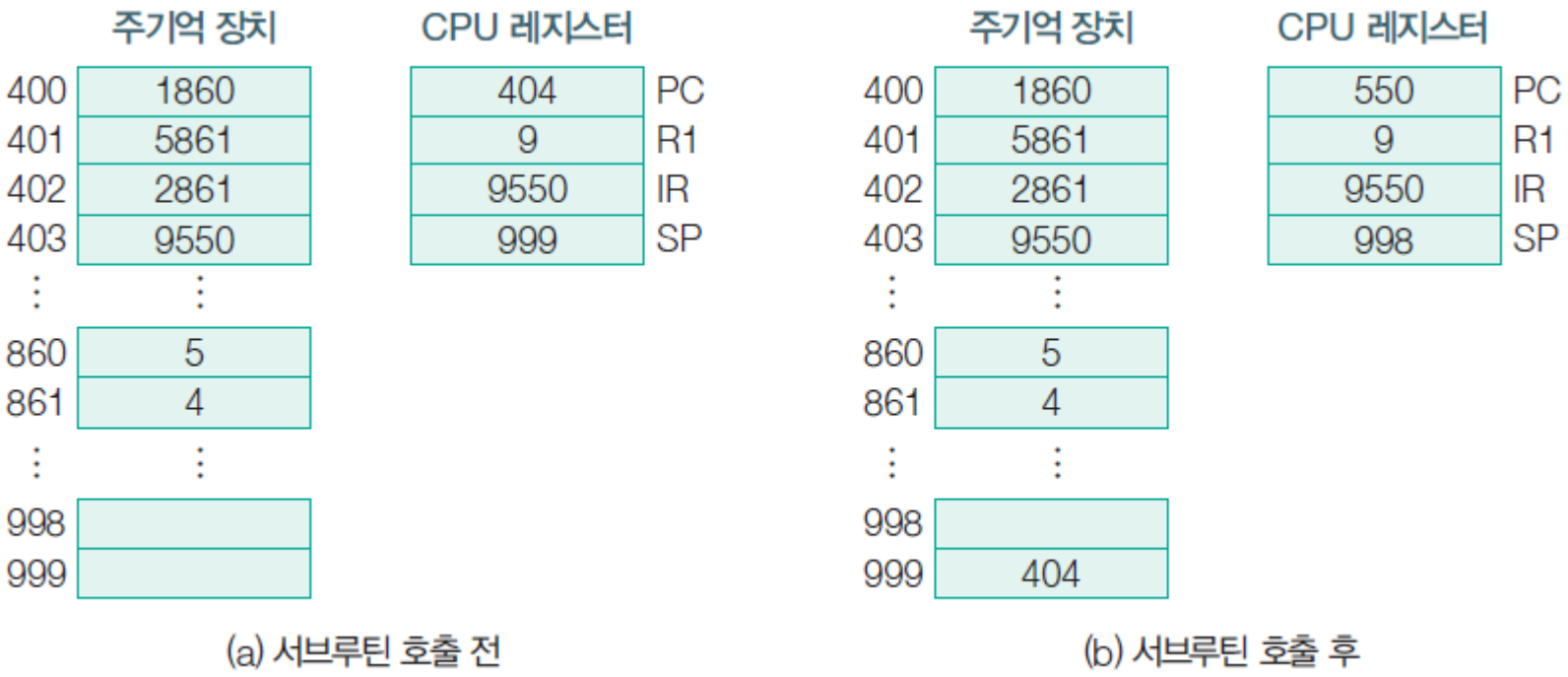


그림 5-15 서브루틴이 호출되는 과정에서 레지스터 변화

- 예: RET(return)

t_1: SP ← SP + I

t_2: MAR ← (SP)

t_3: MBR ← M[MAR]

t_4: PC ← MBR

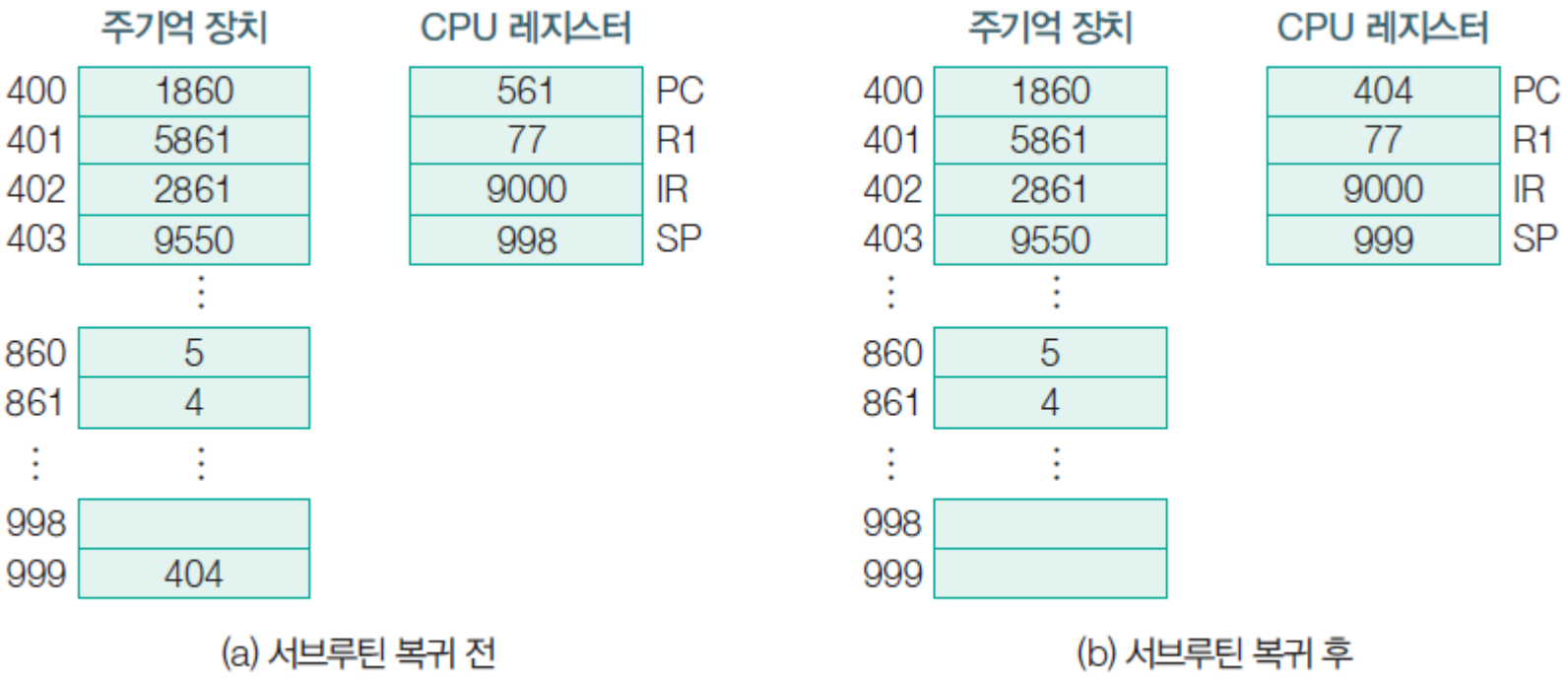


그림 5-16 서브루틴에서 복귀하는 과정에서 레지스터 변화

❑ 중첩 서브루틴이나 다중 서브루틴인 경우

- 매 서브루틴 호출마다 스택에 복귀할 주소(PC)를 저장하고,
- 복귀할 때는 스택 값을 꺼내어 PC로 가져온다.

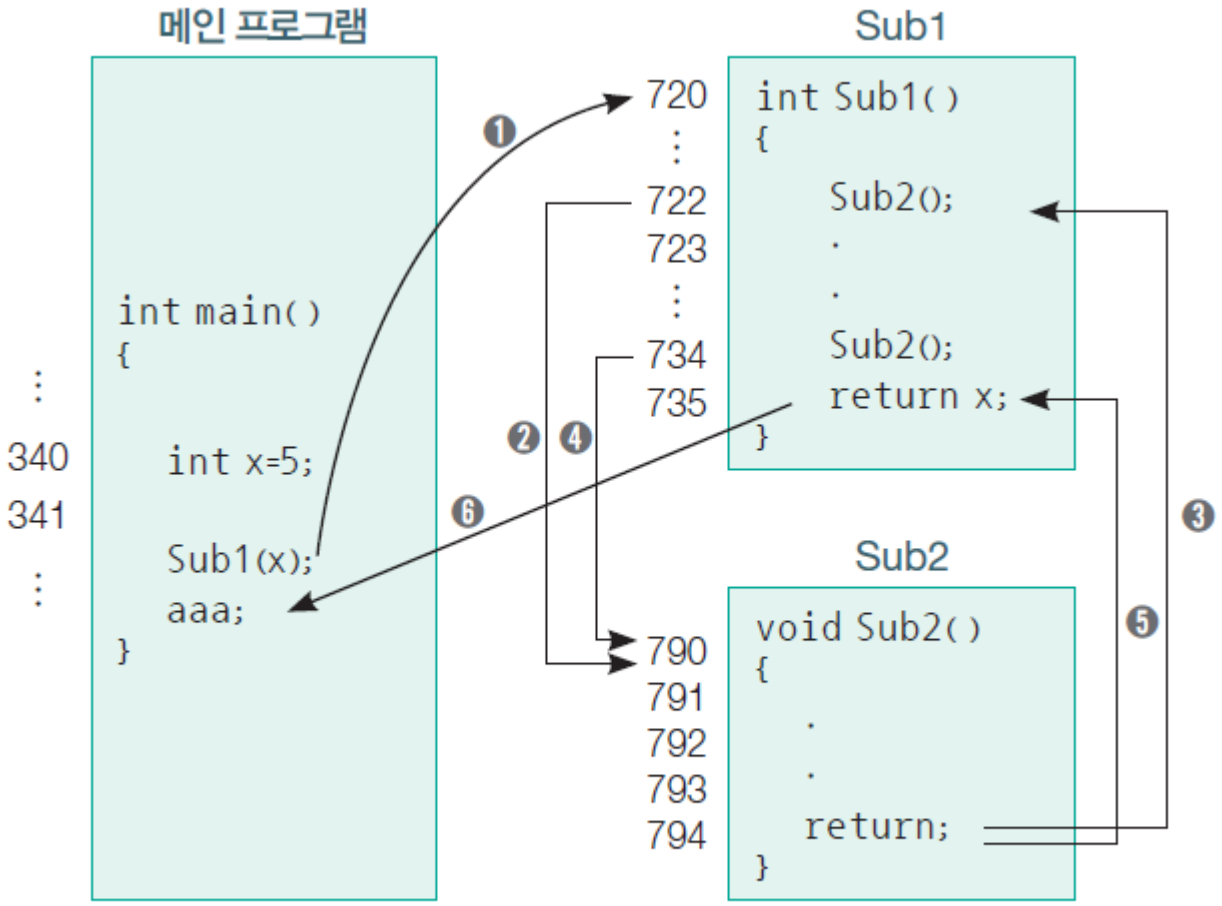


그림 5-17 다중 서브루틴이 호출되는 상황

❑ 다중 서브루틴 예에서 레지스터 변화

❶

996	
997	
998	
999	341
SP	998
PC	720

❷

996	
997	
998	723
999	341
SP	997
PC	790

❸

996	
997	
998	
999	341
SP	998
PC	723

❹

996	
997	
998	735
999	341
SP	997
PC	790

❺

996	
997	
998	
999	341
SP	998
PC	735

❻

996	
997	
998	
999	
SP	999
PC	341

그림 5-18 다중 서브루틴이 호출되는 상황에서 레지스터 변화

4 인터럽트 사이클: interrupt

- 실행 주기가 완료되면 인터럽트가 발생했는지 여부 점검
- 인터럽트가 발생했다면 인터럽트 사이클 실행
- 인터럽트 사이클은 통상적으로 다음과 같다.

$$t_1: \text{MAR} \leftarrow (\text{SP}),\ \text{MBR} \leftarrow (\text{PC})$$

$$t_2: \text{M[MAR]} \leftarrow (\text{MBR}),\ \text{SP} \leftarrow \text{SP} - \text{I}$$

$$t_3: \text{PC} \leftarrow \text{Interrupt\_Service\_Routine\_Address}$$

5 명령어 사이클

- 항상 명령어 인출, 명령어 해독, 명령어 실행 사이클 순서로 실행
- 인터럽트 사이클은 항상 명령어 실행이 끝난 후 인터럽트가 있으면 실행하고, 그렇지 않으면 다음 명령어 인출 사이클로 진행

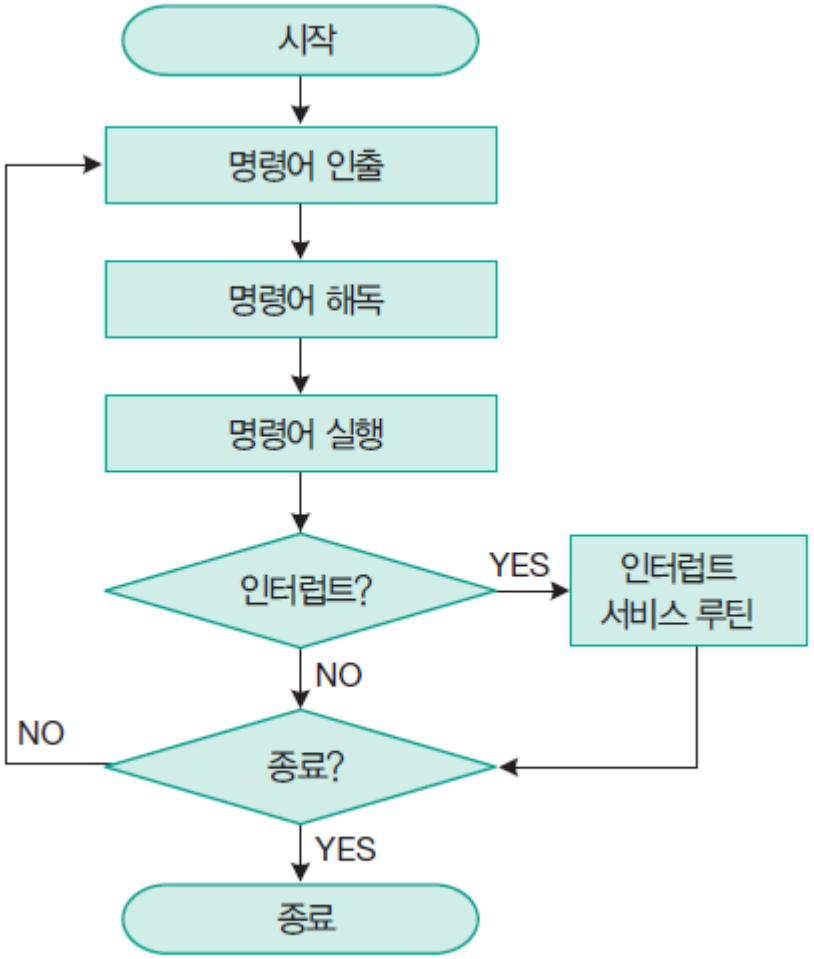


그림 5-19 명령어 사이클

Summary

- 제어 장치의 기능과 종류를 학습
- 명령어 실행 사이클을 이해하고 동작 원리를 이해