9. 마이크로코드와 제어 설계

9.1 하드와이어드 제어 vs 마이크로프로그래밍

> CPU 제어장치 설계 방식의 두 축

🧠 제어장치란?

제어장치(Control Unit)는
CPU 내부에서 명령어의 실행을 위한 제어 신호들을 생성하는 블록이다.

> ALU, 레지스터, 메모리 등 다양한 구성요소들이 **언제 무엇을 해야 할지 제어 신호를 보내는 역할**을 수행한다.

🔁 제어 방식의 두 가지 유형

분류	방식
하드와이어드 제어	회로로 고정된 제어 논리
마이크로프로그래밍	ROM 형태의 마이크로명령어로 제어 신호 생성

🔧 1. 하드와이어드 제어 (Hardwired Control)

✅ 정의

논리 회로(조합 논리회로)를 통해 직접 제어 신호를 생성하는 구조

> 명령어의 Opcode, 상태, 클럭 등 입력에 따라
> **게이트 회로, 디코더, FSM**으로 직접 제어신호를 만들어냄

✅ 동작 흐름

```
Instruction → Decoder → Control Logic (AND, OR, NOT 등) → Control Signals
```

✅ 특징

항목	설명
속도	빠름 (즉시 신호 생성) ⚡
복잡도	고정 회로 설계, 복잡한 명령어 어려움
변경/확장	어려움 (하드웨어 재설계 필요)
RISC 아키텍처에 적합	단순하고 빠른 제어에 유리

💾 2. 마이크로프로그래밍 (Microprogrammed Control)

✅ 정의

제어 신호를 마이크로명령어(microinstruction)로 만들어서
ROM/PLA에 저장된 마이크로프로그램을 순차적으로 실행해 제어 신호를 생성하는 방식

> 즉, 제어장치를 마치 **작은 내부 CPU처럼 구성**
> → 명령어마다 제어 프로그램(마이크로프로그램)을 따로 실행

✅ 동작 흐름

```
Opcode → Microinstruction Address → Control Store (ROM) → Control Signals
```

- 제어 저장소(Control Store)에 저장된 마이크로명령어를 읽어
  여러 클럭 사이클에 걸쳐 제어신호를 순차 실행

✅ 특징

항목	설명
속도	느림 (마이크로명령 순차 실행) 🐢
유연성	제어 프로그램만 바꾸면 구조 수정 가능
복잡한 명령어	CISC 구조에 적합 (다단계 실행 지원)
설계 쉬움	새로운 명령어 추가가 용이

🧩 하드와이어드 vs 마이크로프로그래밍 비교표

항목	하드와이어드 제어	마이크로프로그래밍 제어
제어신호 생성 방식	조합 논리 회로	마이크로명령어 해석
속도	빠름 (1 사이클에 제어 가능)	느림 (여러 사이클 필요)
설계 복잡도	회로가 복잡	소프트웨어 설계 유연
확장성	낮음 (회로 수정 필요)	높음 (프로그램 수정)
명령어 구조	단순, 고정형	복잡한 명령도 처리 가능
대표 아키텍처	RISC	CISC (ex: x86 초기, VAX)

🧪 예시

✅ 하드와이어드 제어 예

- MIPS, ARM Cortex-M
- 단순 명령어 집합 + 빠른 제어 + 고정형 구조

✅ 마이크로프로그래밍 제어 예

- Intel 8086, IBM System/360
- 복잡한 명령어를 제어하기 위해 마이크로코드 사용

📌 요약 정리

항목	설명
하드와이어드 제어	논리 회로 기반, 빠르고 단순하지만 유연성 부족
마이크로프로그래밍	ROM 기반 마이크로명령어 실행, 느리지만 유연
적합 대상	RISC → 하드와이어드 / CISC → 마이크로프로그래밍
현대 트렌드	단순 명령은 하드와이어드, 복잡 연산은 일부 마이크로코드 혼합 방식 사용 (하이브리드 구조)

9.2 마이크로명령어의 구조

> CPU 내부 제어를 위한 가장 작은 명령 단위

🧠 마이크로명령어(Microinstruction)란?

마이크로명령어(Microinstruction)는
하나의 제어 사이클 동안 CPU 내부 구성요소들을 제어하기 위한 이진 명령어다.

> 제어장치가 메모리(제어 저장소, Control Store)에 저장된 이 **마이크로명령어**를 순차적으로 실행하여
> 하나의 머신 명령어(예: `ADD R1, R2`)를 완성함

🔁 동작 흐름 개요

1. 명령어 디코딩 (예: `ADD`)
2. 해당 명령어에 대응하는 **마이크로프로그램 시작 주소** 결정
3. 제어 저장소에서 마이크로명령어를 순차적으로 실행
4. 각 마이크로명령어는 ALU, 레지스터, 메모리 등에 제어 신호 전송

🧱 마이크로명령어의 내부 구성

📦 기본 구성 필드

필드	설명
제어 신호 필드	ALU, 레지스터, 버스 제어용 신호
주소 필드	다음 마이크로명령어 주소 지정
조건 필드	분기 조건 (Zero, Negative 등)
상태 필드	CPU 상태 플래그 (예: Carry, Overflow)

✅ 예시: 마이크로명령어 필드 구성 (일반형)

필드	비트 수	기능
ALU 제어	4비트	ADD, SUB, AND, OR, SHL 등
레지스터 읽기	3비트	소스 레지스터 선택
레지스터 쓰기	3비트	목적지 레지스터 선택
메모리 제어	2비트	읽기, 쓰기, 없음
조건 분기	2비트	무조건, 조건부, 없음
다음 주소	6~12비트	마이크로프로그램 흐름 제어

📏 수평형 vs 수직형 마이크로명령어

구분	수평형 (Horizontal)	수직형 (Vertical)
제어신호 수	많음 (수십~수백 비트)	적음 (압축됨)
명령어 길이	김 (대역폭 큼)	짧음
실행속도	빠름 (병렬 제어)	느림 (디코딩 필요)
디코더	필요 없음	필요함
하드웨어 비용	큼	작음
예시	고속 제어장치	마이크로제어기, 임베디드

> 현대 CPU에서는 일반적으로 **수직형을 기본으로 하되, 일부 병렬 제어 필드는 수평형 형태로 혼합**함

📊 마이크로명령어 실행 예

예: `ADD R1, R2` 명령어 처리에 필요한 마이크로명령어 시퀀스

사이클	마이크로동작
1	R1 → TEMP (레지스터 읽기)
2	TEMP + R2 → ALU 결과
3	결과 → R1 (쓰기)

→ 각 동작마다 대응되는 **마이크로명령어**가 실행됨

🧠 마이크로프로그램 흐름 제어 방식

방식	설명
순차 진행	다음 주소 = 현재 주소 + 1
조건 분기	ALU 결과에 따라 마이크로프로그램 분기
제어 명령 기반	`Jump`, `Call`, `Return` 등의 제어 흐름 명령어 포함
명령어 맵 테이블 사용	각 머신 명령어마다 시작 주소를 저장

🛠️ 마이크로명령어 작성 예시 (수평형)

```
ALU: ADD,
SrcReg: R2,
DstReg: R1,
Mem: NONE,
Cond: ALWAYS,
NextAddr: 001011
```

→ 위 하나의 마이크로명령어가 특정 제어 신호를 동시에 발생

🧪 현실 예: IBM System/360, Intel x86

시스템	설명
IBM System/360	마이크로코드 기반의 대표 시스템, 명령어마다 수십개의 마이크로명령어 실행
Intel x86 (초기)	CISC 명령어를 마이크로코드로 구현 (현대는 하이브리드 구조)
AMD K10	복잡 명령어를 마이크로-Op 단위로 변환하여 OOE에 투입 (μOps Cache 존재)

📌 요약 정리

항목	설명
마이크로명령어	제어 신호 집합, ROM에 저장된 내부 명령어
구성 필드	ALU, 레지스터, 메모리 제어, 분기 조건, 다음 주소
형태	수평형(빠름, 큼) vs 수직형(느림, 작음)
흐름 제어	순차, 조건 분기, 제어 전이
현대 구조	마이크로-Op 기반 Out-of-Order 실행과 결합됨

9.3 마이크로코드 제어 유닛 설계

> 제어 신호를 "프로그래밍"하는 방식

🧠 마이크로코드 제어 유닛이란?

마이크로코드 제어 유닛(Microcoded Control Unit)은
CPU의 각 명령어를 실행하기 위해 필요한 **제어 신호 시퀀스를 "마이크로프로그램" 형태로 저장하고**,
그 순서에 따라 실행하는 제어 구조이다.

> 하드웨어가 아닌 **ROM에 저장된 제어 명령어(microinstruction)**로 CPU 구성요소들을 제어함.

📦 전체 구성 블록 다이어그램

```
        +--------------------------+
        |     Control Address      |  ← 마이크로주소 생성기 (UADDR)
        |       Sequencer          |
        +----------+---------------+
                   |
                   v
        +--------------------------+
        |      Control Memory      |  ← 마이크로프로그램 저장 (ROM or RAM)
        +--------------------------+
                   |
                   v
        +--------------------------+
        |   Control Word Register  |  ← 현재 마이크로명령어
        +--------------------------+
                   |
                   v
        +--------------------------+
        |   CPU 내부 제어 신호들    |  → ALU, RegFile, BUS, MEM, etc.
        +--------------------------+
```

🔄 동작 순서 요약

1. **명령어 디코드 단계**에서 해당 명령의 **마이크로프로그램 시작 주소 결정**
2. **주소 생성기(UADDR)**가 해당 주소를 Control Memory에 전달
3. **마이크로명령어(Microinstruction)**를 Control Memory에서 읽음
4. **제어 신호(Control Word)**를 발생시켜 각 구성요소 제어
5. **다음 마이크로명령 주소** 계산:
    - 순차 실행 (`UADDR + 1`)
    - 조건 분기
    - 명령어 해석 기반 점프

🧠 핵심 구성 요소 설명

✅ 1. Control Memory (제어 저장소)

- 마이크로명령어들을 저장하는 ROM (or RAM)
- 각 주소마다 하나의 제어 워드 (수십~수백 비트)

✅ 2. Control Address Sequencer (주소 생성기)

- 다음에 실행할 마이크로명령어의 주소를 결정
- 입력: 분기 조건, 상태 플래그, 명령어 디코딩 결과
- 구현 방식:
    - 순차 증가기 (Counter)
    - 조건 분기 회로
    - 명령어 맵핑 테이블

✅ 3. Control Word Register (CWR)

- 현재 마이크로명령어를 일시 저장
- CWR의 각 비트는 CPU 내부 신호선으로 연결됨 (예: ALU 연산, 레지스터 읽기, 쓰기 등)

📏 마이크로프로그램 흐름 제어 방식

방식	설명
순차 흐름	다음 주소 = 현재 주소 + 1
조건 분기	ALU 결과, 플래그에 따라 분기
제어 명령 기반 전이	Jump, Call, Return 등
Opcode Mapping	명령어 디코더의 출력에 따라 시작 주소 결정

🧪 예: 명령어 → 마이크로프로그램 흐름

예: `ADD R1, R2`

마이크로주소	동작
`000100`	R1 → TEMP
`000101`	TEMP + R2 → ALU
`000110`	결과 → R1
`000111`	다음 명령어로 넘어가기 위한 fetch 수행

🔧 마이크로코드 저장 방식

방식	설명
ROM	변경 불가, 고정형 CPU 설계
EPROM/EEPROM	초기 CPU 개발 시 재작성 가능
RAM 기반 (Soft Microcode)	부트업 시 로딩 가능 → 마이크로코드 업데이트 지원
→ 현대 x86 CPU는 이 방식을 사용	

💡 현대 CPU에서의 활용

항목	설명
Intel x86 (Micro-op Fusion)	CISC 명령어를 여러 개의 μ-op로 변환, 일부는 마이크로코드 기반
AMD Zen 아키텍처	복잡 명령은 마이크로코드 ROM에서 해석 후 μ-op으로 변환
마이크로코드 업데이트	펌웨어 형태로 배포되어 보안 패치 수행 가능 (ex. Spectre 대응)

📌 요약 정리

항목	설명
마이크로코드 제어 유닛	마이크로명령어로 제어 신호를 생성하는 구조
Control Memory	마이크로프로그램 저장소 (ROM or RAM)
Sequencer	다음 마이크로주소 결정 로직
Control Word	ALU, Reg, BUS 등에 제어 신호 출력
현대 활용	복잡 명령어 처리, 보안 패치, 유연한 명령어 정의 가능성 제공