제어 유니트 개요

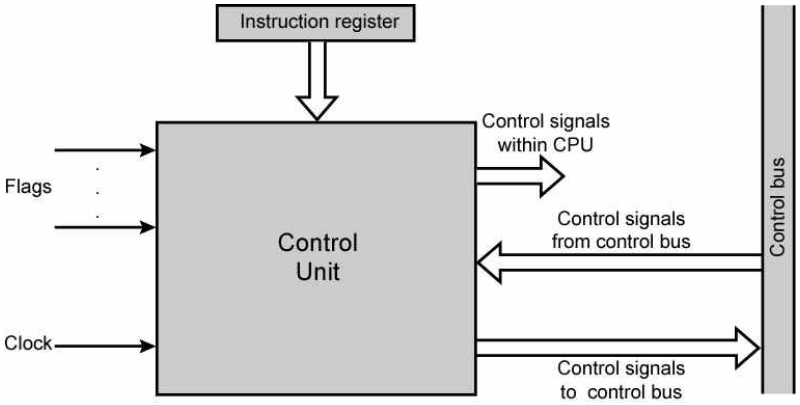
1. 제어 유니트

- **제어 유니트(Control Unit)**
  - 컴퓨터의 동작을 제어하는 유니트
  - 컴퓨터 구성요소에 필요한 제어신호를 발생시키는 회로.

- **제어 유니트 기능**
  - 명령어 해석
    - Instruction register (IR) 에 저장된 명령어 decode
  - 시스템에 필요한 모든 제어신호 생성
    - ALU, register, memory, i/o device 에 필요한 제어신호 생성.

1.1 제어 유니트 모델

- 제어 유니트 모델



1.2 제어 유니트의 입력 신호

- **입력신호**
  - 명령어 레지스터(IR)의 Op-Code 필드
    - 현재 실행하고 있는 명령어의 동작을 정의하는 필드
  - CPU 상태 플래그(Flags)
    - ALU 연산 결과값 반영한 플래그
  - 입출력 장치 제어신호
    - 입출력 장치와 송수신 하는 신호
      - 인터럽트, 통지(Acknowledgement) 신호

1.3 제어 유니트의 출력 신호

- **출력신호**
  - CPU 제어신호
    - ALU 기능 선택 : 연산 및 데이터 전송
    - 레지스터 제어신호 : 레지스터 Load, 또는 레지스터 선택 신호
  - 메모리 제어신호
    - 메모리 인터페이스 신호 : 제어버스 신호
  - 상호 연결 제어신호
    - 구성요소들을 상호연결하는 회로에 대한 제어 신호
    - 상호연결 구현 방식에 따라 제어신호 다름.
    - 상호연결 구현 방식 : 버스 기반, 멀티플렉스 기반

2. 제어 유니트 설계

- **명령어 세트에 포함된 명령어들에 대한 MOP 정의**
  - 명령어 사이클 정의
  - 명령어 사이클을 구성하는 동작 사이클 정의
  - 동작 사이클을 구성하는 MOP 정의

- **데이터 경로부 설계**
  - 조합회로와 순차회로를 사용하여 데이터 경로부 구현

- **데이터 경로부의 제어신호 정의**
  - ALU의 연산 선택신호
  - 레지스터, 메모리 제어신호
  - 상호연결 회로에 필요한 제어신호 : 버스기반 연결, 멀티플렉서 기반 연결

2. 제어 유니트 설계

- **제어 유니트의 입력 및 출력 신호 정의**
  - 입력신호 : 명령어, 상태정보
  - 출력신호 : 데이터 경로부의 데이터 흐름을 제어하는 신호

- **제어 유니트 구현**
  - Hard-wired 방식 구현
  - Micro-programmed 방식 구현

3. 제어 유니트 구현방식

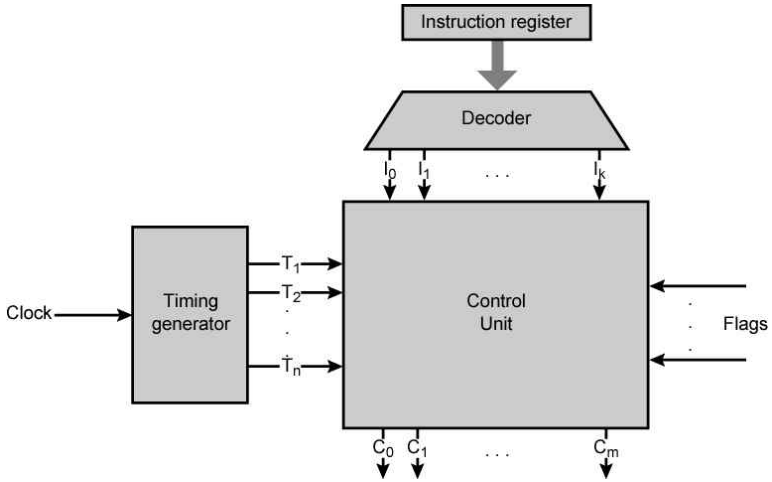
- **제어 유니트 구현 방법**
  - Hard-wired 방식
    - 논리 회로를 사용한 직접 구현
    - 고성능, 유연성 저하
    - RISC에 적합
  - Micro-programmed 방식
    - 마이크로 프로그램 방식을 사용한 구현
      - Micro-operation, Micro-instruction, Micro-program 사용.
    - 장점 : 설계가 단순, 저비용.
    - 단점 : 느리다.
    - CISC에 적합

Hard-wired 제어 유니트

1. Hard-wired 제어 유니트 개요

- **논리회로를 사용한 직접 구현**
  - 조합회로와 순차회로 설계방법에 따라 설계
  - 명령어가 간단한 RISC 계열 프로세서에 적합

- **제어 유니트 구성**
  - 명령어 해석회로
    - 디코더(Decoder) 회로 : 명령어의 Op-Code 부분을 해석하여 명령어 식별
  - 제어신호 출력 타이밍 생성
    - 타이밍 발생기(Timing Generator)



1.1 Hard-wired 제어 유니트 설계 절차

- **Hard-wired 제어 유니트 설계 절차**
  - 명령어 세트를 구성하는 명령어들을 실행하는 MOP(Micro-operation) 들을 정의.
  - MOP을 실행할 수 있는 데이터 경로부(Datapath) 설계
  - 데이터 경로부를 제어하는 제어신호 정의
  - 제어신호를 생성하기 위한 타이밍 회로 설계
  - 타이밍에 따른 제어신호 출력 로직구현.

1.2 Hard-wired 제어 유니트 장단점

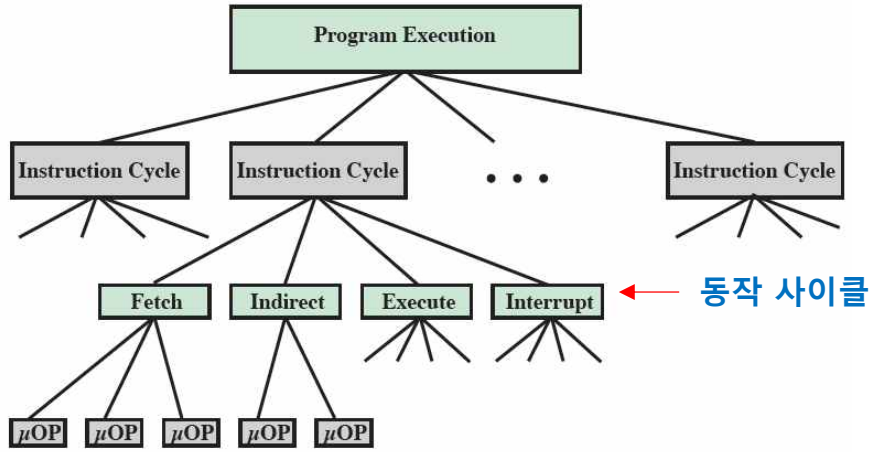
- **Hard-wired 제어 유니트 장점**
  - 설계가 비교적 간단
  - 제어 신호 발생이 빠르다.

- **Hard-wired 제어 유니트 단점**
  - 복잡한 회로가 필요하다.
  - 설계와 테스트가 쉽지 않다.
  - 회로의 유연성이 부족하다.
  - 새로운 명령어의 추가가 쉽지 않다. 재설계 필요.

- **유연한 제어 유니트 필요 → 마이크로 프로그램을 사용한(micro-programmed) 제어 유니트**

2. 명령어의 MOP 정의

- 프로그램 = 명령어들의 배열
- 프로그램 실행 = 순서적 실행. 분기 명령어를 사용한 순서흐름 변경
- 명령어 실행 = 명령어 사이클의 실행
- 명령어 사이클(Instruction Cycle)을 구성하는 동작 사이클(Operation Cycle)은 여러 개의 Micro-operation(MOP) 들로 구성.



2. 명령어의 MOP 정의

- **명령어의 Micro-operation**
  - CPU의 Atomic RTL(Register Transfer Level) operation.
  - Micro-operation 유형
    - 레지스터와 레지스터사이의 데이터 전송
    - 레지스터와 메모리와의 데이터 전송
    - 산술 및 논리 연산

2.1 명령어 MOP 정의 사례

- 명령어 인출 사이클의 MOP 구성

t1: MAR <- (PC)
t2: MBR <- (memory)
t3: IR <- (MBR), PC <- (PC) +1

- 오퍼랜드 인출 사이클의 MOP 구성

t1: MAR <- IR.addr
t2: MBR <- (memory)
t3: MAR <- MBR

2.1 명령어 MOP 정의 사례

- **실행 사이클의 MOP 구성**
  - 명령어마다 다르게 구성.

(ADD R1, x)

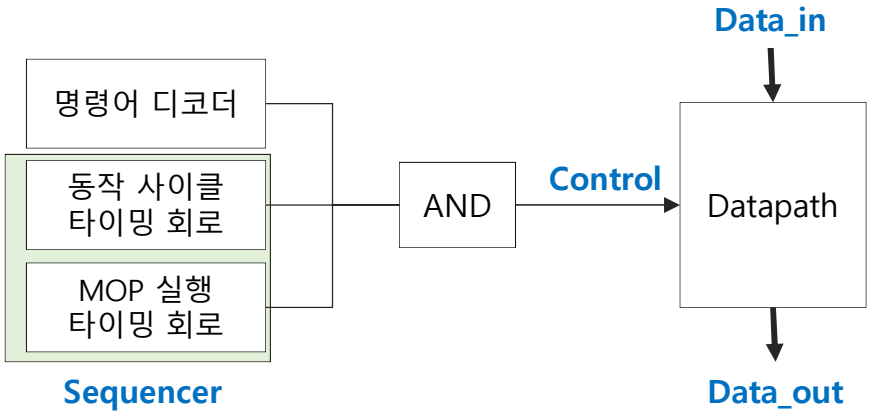
t1: MAR <- IR.addr
t2: MBR <- (memory)
t3: R1 <- R1 + MBR

(ISZ x : increment & skip if zero)

t1: MAR <- IR.addr
t2: MBR <- (memory)
t3: MBR <- MBR + 1
t4: (memory) <- MBR, if MBR=0, PC++

3. 타이밍 발생기

- **타이밍 발생기 설계**
  - 명령어 사이클의 MOP 분석
  - 명령어 실행에 필요한 시퀀서(Sequencer) 설계
  - 시퀀서에서 발생한 타이밍 신호와 명령어 해석결과를 AND 한 결과가 최종 제어신호.



3. 타이밍 발생기

- **시퀀서 구성**
  - 동작 사이클 타이밍 회로
    - 명령어 사이클을 구성하는 동작 사이클 구성을 분석
      - 가장 긴 사이클을 기준으로 회로 설계
  - MOP 실행 타이밍 회로
    - 동작 사이클을 구성하는 MOP 실행라인 분석
      - 가장 긴 라인을 기준으로 회로 설계
  - 카운터와 디코더 회로를 사용하여 타이밍 신호 생성

Micro-programmed 제어 유니트

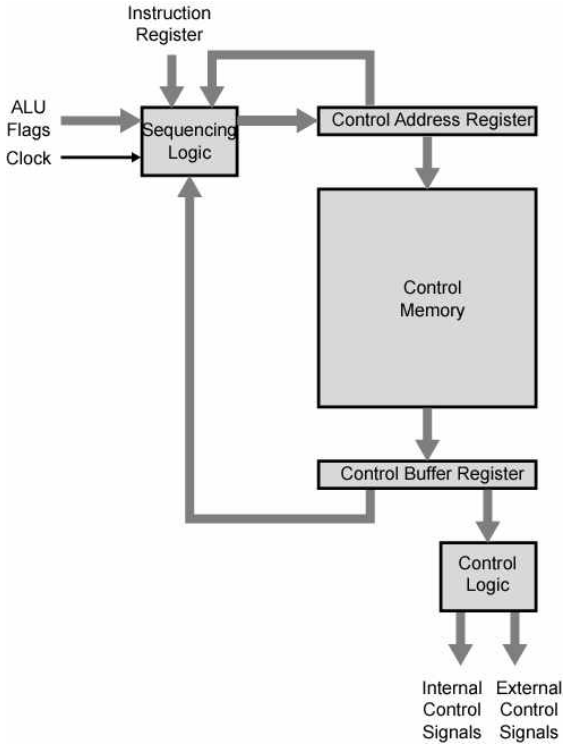
1. Micro-programmed 제어

- **Micro-program을 사용한 제어 유니트 구현**
  - 명령어 실행에 필요한 제어신호를 Micro-program 을 사용하여 생성.

- **Micro-program 구성**
  - 제어 유니트이 생성하는 모든 제어신호 세트를 Control Word 로 정의.
    - Control word = Micro-instruction, Micro-code
  - 각 명령어를 구성하는 Micro-operation을 Control Word 로 표현.
    - 명령어 단위로 Control Word Sequence 정의 = Micro-program
    - Micro-program 을 Control Memory에 저장
  - Micro-operation 실행 타이밍에 따라 Control Word 를 출력.
    - Sequencer 를 사용하여 Control Memory 어드레스 생성
    - Control Memory에 저장된 Micro-instruction 을 출력.

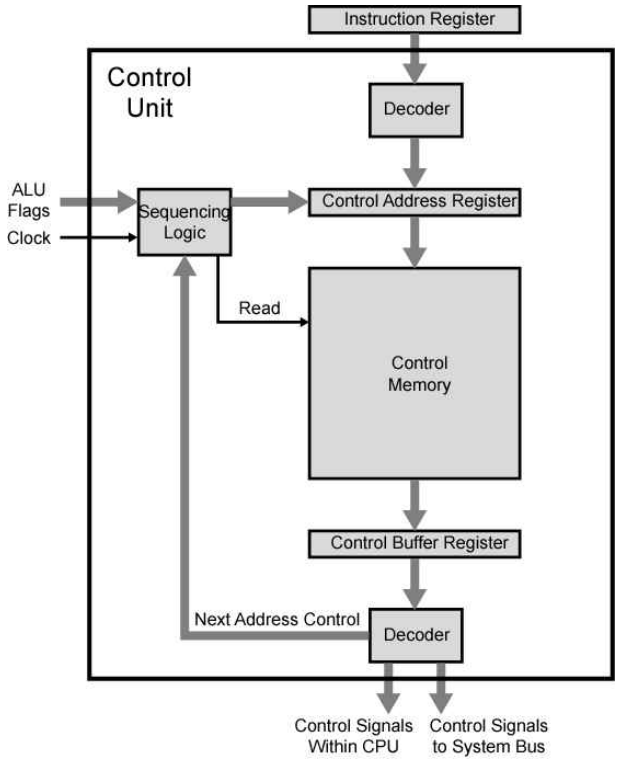
1.1 Micro-programmed 제어 유니트 구성요소

- **Micro-programmed 제어 유니트의 구성요소**
  - 시퀀스 회로 (Sequencing Logic)
    - Control Word 출력 순서 및 타이밍 생성
  - 제어 레지스터 및 디코더 (Control register & Decoder)
    - 제어 메모리 액세스를 위한 어드레스 및 데이터 저장
    - Control Word 해석
  - 제어 메모리 (Control Memory)
    - Control Word 를 저장하고 있는 메모리



1.2 Micro-programmed 제어 유니트 동작

- **제어 유니트의 동작절차**
  - 시퀀싱(Sequencing)회로가 read 명령어 발행.
  - Control address register(CAR)를 사용하여, control memory를 액세스하여 micro-instruction 을 읽어서 control buffer register(CBR)에 저장.
  - CBR 값을 해석해서 제어신호와 다음 micro-instruction의 어드레스를 결정.
  - 시퀀싱 회로가 다음 어드레스를 CAR 에 로딩.

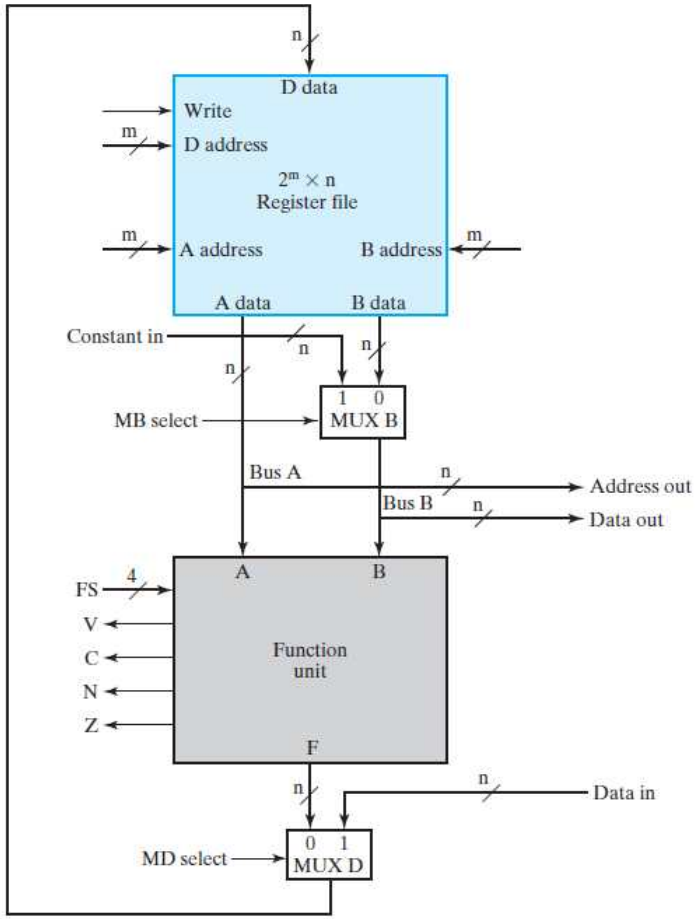


1.3 Micro-programmed 제어 유니트의 시퀀싱

- **Micro-program에서 다음 어드레스 결정방법**
  - ALU 플래그와 CBR 값에 따라 결정
  - 다음 micro-instruction 위치
    - CAR++
  - JUMP micro-instruction을 사용해서 새로운 routine 으로 이동.
    - CBR의 어드레스 필드를 CAR에 로딩.
  - 새로운 machine instruction routine으로 이동.
    - IR의 opcode를 참조하여 CAR에 적절한 어드레스를 로딩.

2. Control Word

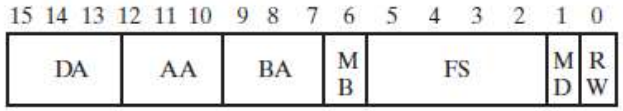
- **데이터 경로부 예시**
  - 레지스터 파일
    - 2 개의 read port
    - 1 개의 write port
  - Functional Unit
    - FS (function select) : 연산유형 선택
    - Constant in : 상수 오퍼랜드
    - Data in/out : 데이터 전송
    - 상태 플래그 : V,C,N,Z



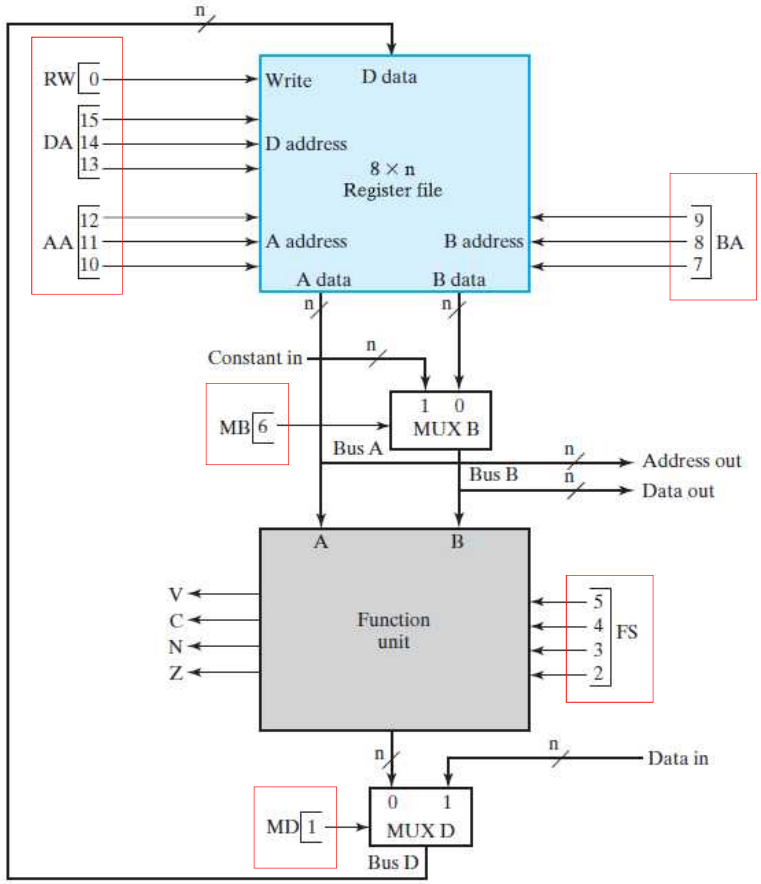
Data Path

2.1 Control Word 정의

- 데이터 경로부에 대한 제어신호 정의
- 제어신호를 Control Word에 매핑



Control Word Definition



Control Signal Definition

2.2 Control Word 인코딩

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
$R0$	000	Register	0	$F = A$	0000	Function	0	No Write	0
$R1$	001	Constant	1	$F = A + 1$	0001	Data in	1	Write	1
$R2$	010			$F = A + B$	0010				
$R3$	011			$F = A + B + 1$	0011				
$R4$	100			$F = A + \overline{B}$	0100				
$R5$	101			$F = A + \overline{B} + 1$	0101				
$R6$	110			$F = A - 1$	0110				
$R7$	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = \text{sr } B$	1101				
				$F = \text{sl } B$	1110				

2.3 Micro-operation과 제어 신호 매핑

- **ALU = Functional Unit**

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + I$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	_	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	_	_	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	_	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	_	_	$R3$	Register	_	_	No Write
$R4 \leftarrow$ Data in	$R4$	_	_	_	_	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

3. Micro-instruction

- **Micro-instruction 설계 요소**
  - 하나의 Micro-instruction에서 동시에 실행해야 할 Micro-operation의 최대 수
  - 제어 신호의 표현 또는 인코딩 방법
  - 다음 Micro-instruction 어드레스 지정방법

- **Micro-instruction 포맷**
  - Control information : 제어신호
  - Branch condition : 제어신호 흐름제어에 사용된 조건
  - Branch address : 다음에 출력할 제어신호가 저장된 어드레스.
    - 제어 메모리 어드레스.

- **명령어 수의 증가와 명령어와 관련된 RTL 하드웨어 증가로, 제어 유니트에서 발생시켜야 할 제어 신호 수의 증가**
  - Micro-instruction(Control Word) 의 길이의 증가
  - 명령어당 생성해야 할 Micro-instruction 수의 증가

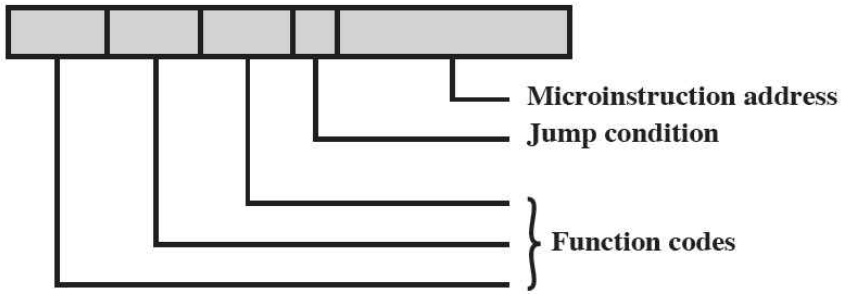
3.1 Micro-instruction 유형

- **Micro-instruction 유형**
  - Vertical micro-programming
    - Encoded Control word
    - 각 micro-instruction이 하나 또는 소량의 MOP 실행에 필요한 제어신호 포함.
  - Horizontal micro-programming
    - Direct Control word
    - 여러 개의 MOP을 동시 실행가능.

- **절충형**
  - 독립적인 제어 신호들을 별도 분리해서 Micro-instruction의 필드에 별도 할당.
  - 병렬성을 훼손하지 않는 범위에서 별도 인코딩.

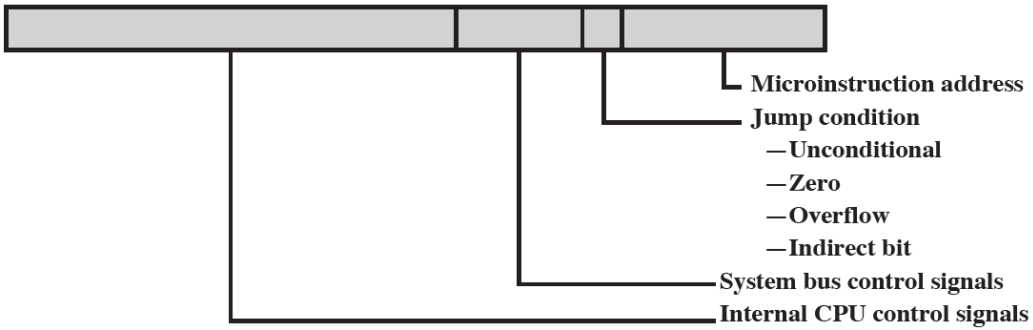
3.2 Vertical Micro-Programming

- **Vertical micro-programming**
  - n 개의 제어신호를 log2n 비트로 표현 : Micro-instruction 길이 단축
  - 제어신호의 동시출력(병렬 제어)이 제한적임.
  - 외부에 별도 디코더 회로 필요



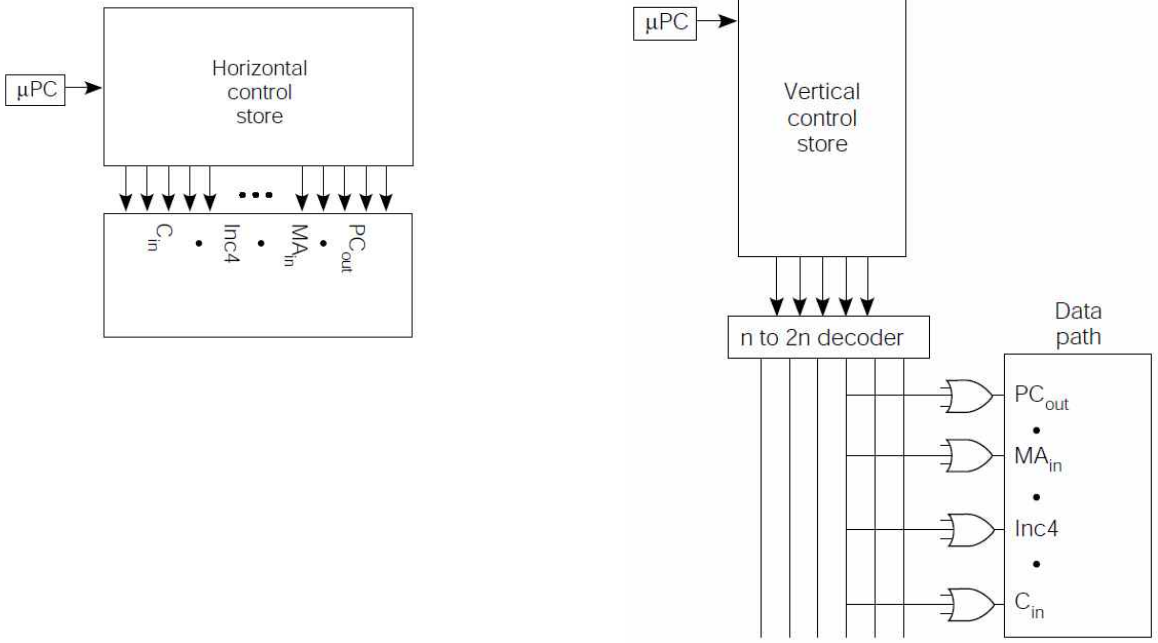
3.3 Horizontal Micro-Programming

- **Horizontal micro-programming**
  - Micro-instruction 길이가 길다.
  - 제어신호와 micro-instruction 의 비트가 1:1 관계
  - 제어 신호의 동시 출력이 용이



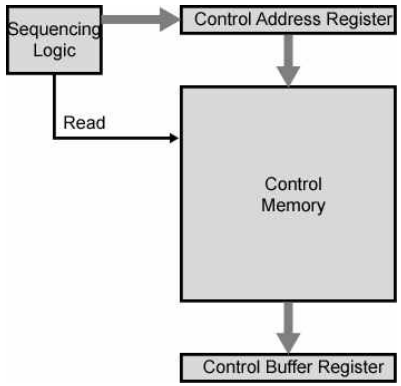
3.4 Vertical/Horizontal 비교

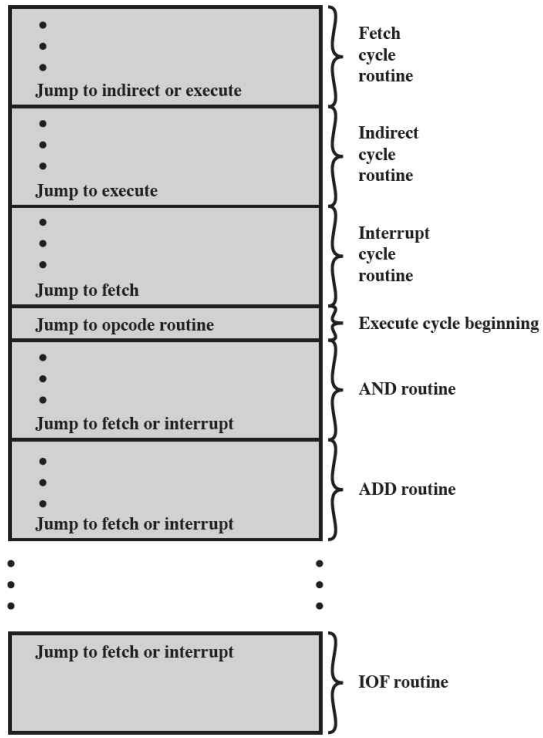
- **Vertical micro-programming vs. Horizontal micro-programming**



4. Control Memoy 구성

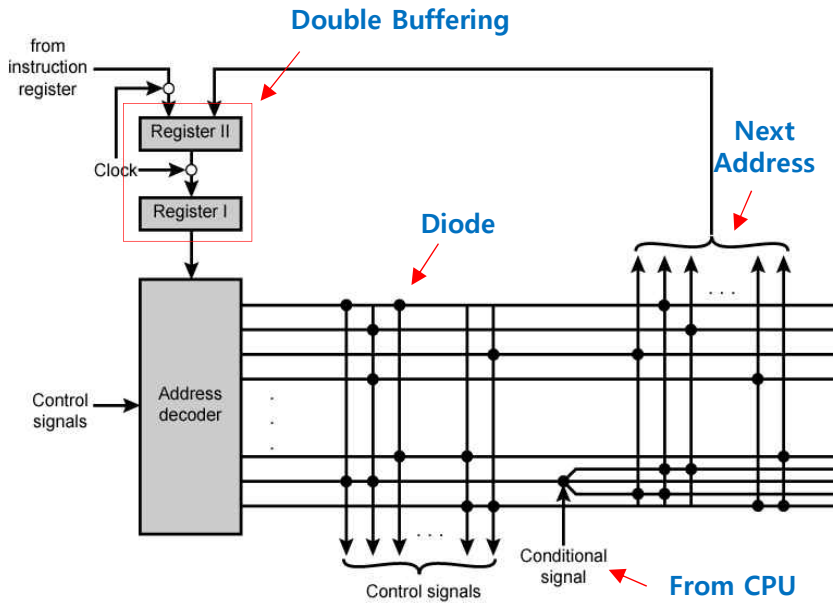
- **Control memory 구성**
  - Micro-program 저장
    - 동작 사이클별 루틴(Routine) 저장
    - 동작 사이클 전환시 분기
- **루틴별 실행 형태**
  - 명령어 인출, 오퍼랜드 인출, 인터럽트 사이클 루틴은 반복 호출
  - 실행 사이클 루틴은 명령어에 따라 선택적 실행





4.1 Wilkes 제어 유니트

- **1951 micro-programmed 제어 유니트 처음 제안.**
  - 다이오드 (Diode)로 구성된 연결 매트릭스 (Matrix)를 사용
    - 제어신호 생성
    - 다음 사이클 어드레스 생성
  - Register II 의 입력
    - IR 레지스터 또는 next address
  - 안정적 동작을 위해 이중 버퍼링 (Double Buffering)
    - Register I, Register II 사용



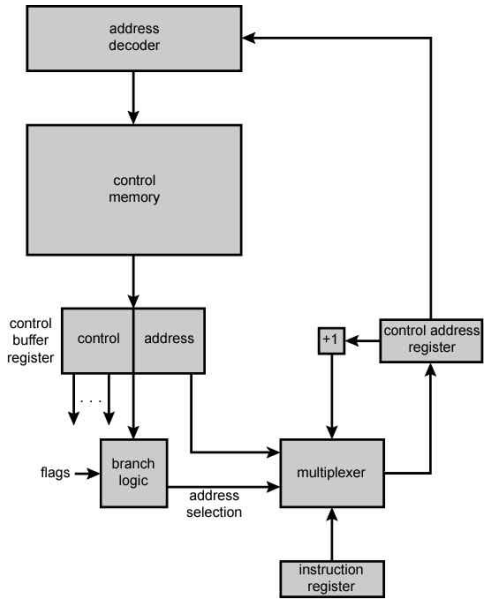
5. 시퀀싱 방법

- **시퀀싱(Sequencing)**
  - 다음에 실행할 Micro-instruction의 위치 값(어드레스)을 결정하는 과정
    - 현재 micro-instruction, condition flags, IR 값에 따라, 다음 address가 다르게 결정.

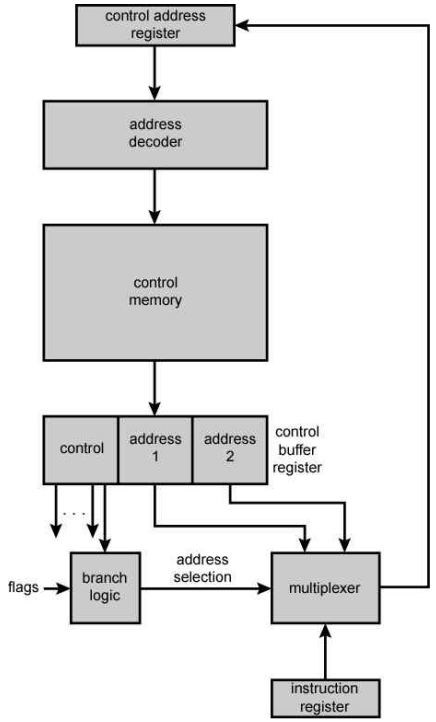
- **Sequencing 방법**
  - 분기 발생시 Micro-instruction의 어드레스 필드 값을 사용하여 다음 어드레스를 결정
  - 어드레스 필드 포맷 유형
    - Single address field
    - Two address fields
    - Variable format

5.1 분기 제어 회로

- 분기 제어 회로



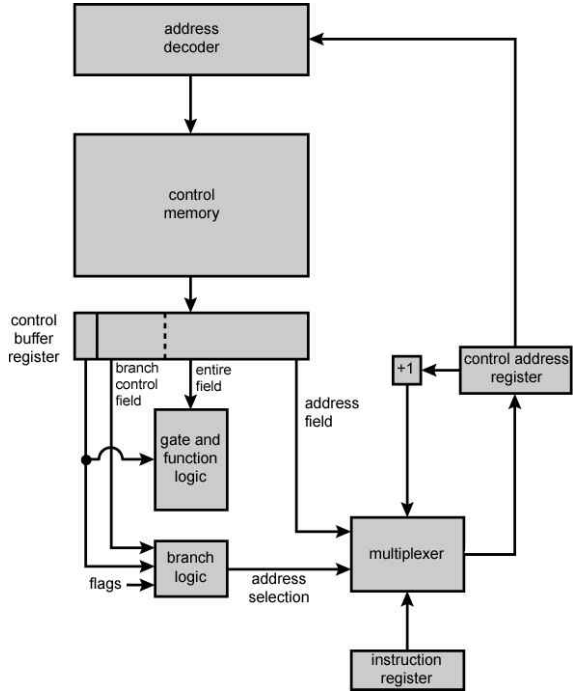
1 address fields



2 address fields

5.1 분기 제어 회로

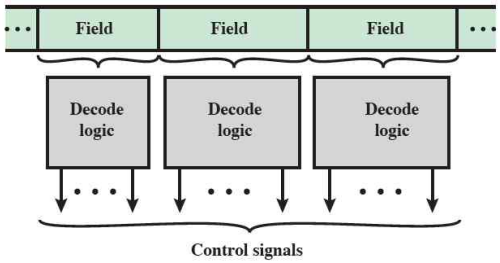
- 분기 제어 회로



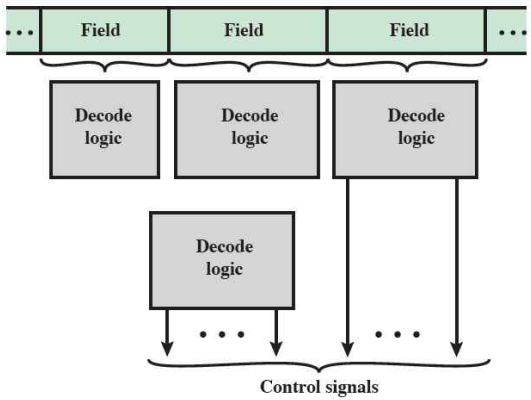
Variable address format : address field 길이가 가변.

7.2 Micro-Instruction 인코딩

- **Micro-instruction 인코딩(Encoding)**
  - 제어 메모리의 폭(Width)와 Micro-programming을 용이하게 하기 위해 Micro-instruction을 인코딩.



Direct Encoding



Indirect Encoding

end