20강_마이크로프로그램

Created	@Aug 13, 2020 12:04 PM
Property	너무 길어 그리고 모르겠다 다시 듣기
Tags	RE

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bccb532e-885f-4009-9738-c534be83f45c/_.pdf

컴퓨터구조론(김종현)

마이크로프로그램(microprogram)

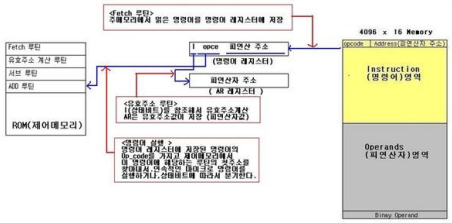
컴퓨터의 명령을 해독하여 실행하는 전자회로의 제어에 사용되는 프로그램. 컴퓨터의 중앙처리장치는 기억장치에서 명령을 꺼내 해독하고, 그에 따라 필요한 데이터를 기억장치에서 읽고, 산술연산 등을 하거나

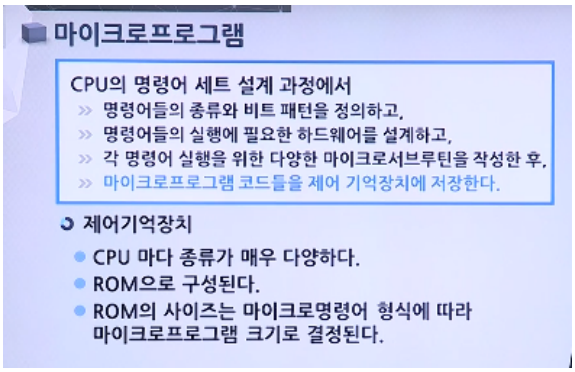
https://www.scienceall.com/%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8microprogram/

정상에 서자

▬ 마이크로 프로그래밍 기법 .(control memory)ROM.))* 마이크로 연산을 수행하기 위해서는 제어신호가 필요하는데 , 제어신호는 비트들의 모임으로 되어있다. 이러한 비트들의 모임을 제어 워드라 한다.

http://dreamform.egloos.com/v/2805479

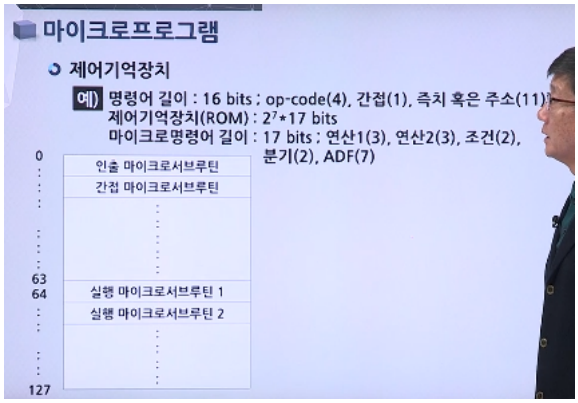




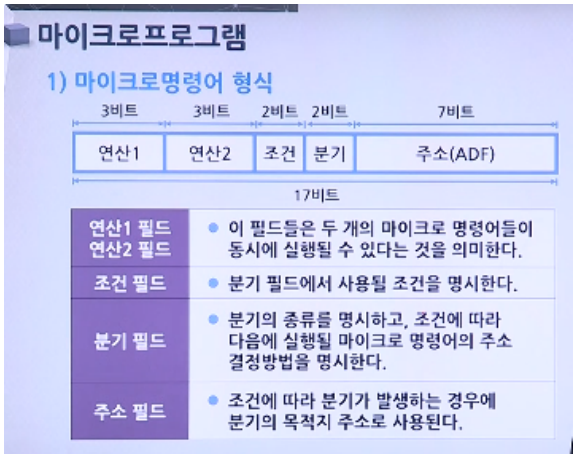
제어기억장치 CPU마다 종류가 다양하다 == 마이크로 프로그램이 다 다르다(하드웨어 구조가 다르다)..모든게 다르다..(명령어 마이크로 명령어 등등)
메모리니까 ROM으로 구성된다 → ROM의 사이즈는 마이크로명령어 형식에 따라 마이크로프로그램 크기로 결정된다

ROM은 작을수록 많이 들어간다~(한정된 면적). 마이크로프로그램을 간결하게 짜야

한다



여기의 16bit는 마이크로가 아니라 cpu가 메모리에서 가져오는 그런 명령어 길이가 16비트이다



연산 1과 2는 마이크로 오퍼레이션과 따로 보아도 되지만 / 조건, 분기, 주소는 3가지를 같이 보면 된다

READ 메모리를 읽어 CPU로 가져온다 (MAR의 지정된 주소값의 내용을 MDR로 가져오기)
PC값을 MAR에다가 줘서 PCTOMAR(PCTAR)
IRTOMAR IR의 내용을 MAR에다가 붙인다 IRTAR
WRITE 의미 CPU의 내용을 메모리에 쓴다
ADD 어큐뮬레이터 내용, MDR 내용 더한다 ADD

모든 마이크로 연산이 연산 1 필드와 전부 다름

2) 마이크로명령어의 2진 코드와 기호

연산2 필드에 위치할 마이크로-연산

코드	마이크로-연산	기호
000	None	NONE
001	PC ← PC+1	INCPC
010	MDR ← AC	ACTDR
011	MDR ← PC	PCTDR
100	PC ← MDR	DRTPC
101	MAR ← SP	SPTAR
110	AC ← AC-MDR	SUB
111	PC ← IR(addr)	IRTPC

마이크로프로그램

2) 마이크로명령어의 2진 코드와 기호

조건 필드에 위치할 마이크로-연산

- U : 무조건 분기
- I : 'I=1'이면 간접 사이클 루틴을 호출
- S : 누산기에 저장된 데이터의 부호가 '1'이면 분기
- Z : 누산기에 저장된 데이터가 '0'이면 분기

코드	조건	기호	설명
00	1	U	무조건 분기
01	I 비트	I	간접 주소 지정
10	AC(S)	S	누산기(AC)에 저장된 데이터의 부호
11	AC=0	Z	AC에 저장된 데이터=0

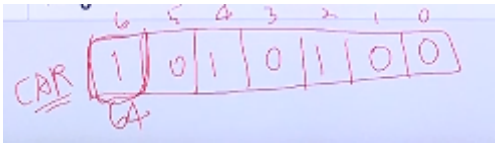
U : unconditional. 무조건 분기함
I : indirect. 간접사이클을 부를 때 쓰는 조건. I가 1인 경우에만 간접 사이클 루틴을 호출
S : 2의 보수, 사인비트. 누산기에 저장된 데이터의 부호 ALU 연산 결과가 음수이면 사인비트가 1이 된다. 데이터의 부호가 1이면 마이너스를 갖는다

2) 마이크로명령어의 2진 코드와 기호

분기 필드에 위치할 마이크로-연산

JUMP	조건에 따라 CAR의 내용을 주소필드(ADF)의 값 혹은 CAR +1 의 값으로 저장한다.
CALL	조건에 따라 CAR의 내용을 주소필드(ADF)의 값 혹은 CAR +1 의 값으로 저장한다. 여기서, 주소필드의 값으로 저장할 때는 복귀 할 주소 값을 SBR에 저장한다.
RET	마이크로서브루틴으로부터 복귀 (SBR에 저장된 내용을 CAR로 적재)
MAP	사상방식(mapping)에 의하여 분기 목적지 주소 결정

리턴 : 마이크로서브루틴을보ㅜ터 복귀



2) 마이크로명령어의 2진 코드와 기호

분기 필드에 위치할 마이크로-연산

코드	기호	설명
00	JMP	'조건=1'이면 CAR ← ADF '조건=0'이면 CAR ← CAR+1
01	CALL	'조건=1'이면 CAR ← ADF, SBR ← CAR+1 '조건=0'이면 CAR ← CAR+1
10	RET	CAR ← SBR(서브루틴으로부터의 복귀)
11	MAP	CAR_6 ← "1", $CAR_{5..2}$ ← IR(op-code), $CAR_{1,0}$ ← "0"

3) 마이크로프로그램

인출 사이클의 마이크로서브루틴

```
        ORG 0
FETCH : PCTAR NONE  U  JMP  NEXT  : MAR ← PC
        READ  INCPC U  JMP  NEXT  : MDR ← M[MAR], PC ← PC+1
        DRTIR NONE  U  MAP        : IR ← MDR
```

Addr	μ-OP1	μ-OP2	CD	BR	ADF
0000000	001	000	00	00	0000001
0000001	100	001	00	00	0000010
0000010	110	000	00	11	0000000

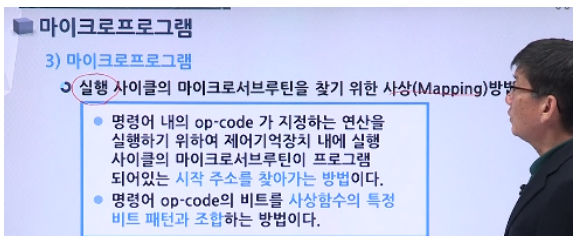
FETCH : (연산필드1) + (연산필드2)

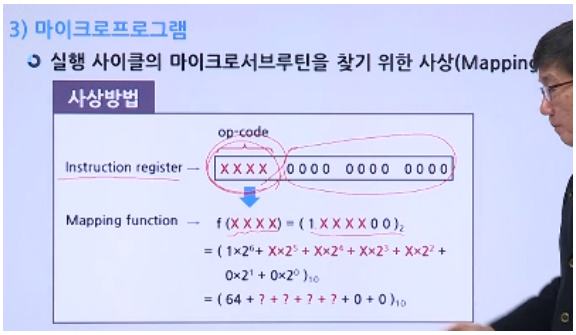
FETCH 루틴의 이름
Addr 은 주소고 그 옆의 테이블 내용들이 마이크로프로그램 코드

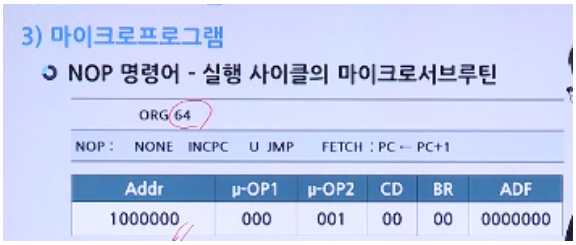
간접 사이클의 마이크로서브루틴

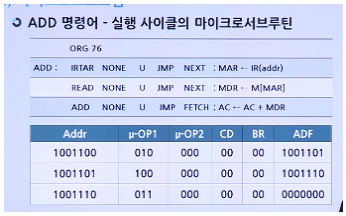
```
        ORG 4
INDRT : IRTAR NONE  U  JMP  NEXT  : MAR ← IR(addr)
        READ  NONE  U  JMP  NEXT  : MDR ← M[MAR]
        DRTIR NONE  U  RET        : IR(addr) ← MDR
```

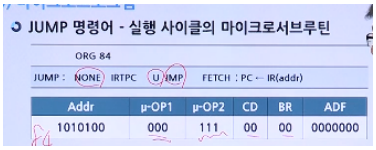
Addr	μ-OP1	μ-OP2	CD	BR	ADF
0000100	010	000	00	00	0000101
0000101	100	000	00	00	0000110
0000110	110	000	00	10	0000000











▼ 스크립트

1. 다음 이제 마이크로프로그램에서 하나씩 저희가 한번 살펴보도록 하죠. 그래서 저희가 이제... CPU의 명령어 세트 설계 과정, 앞에서 설명드린 내용이죠. 명령어들의 종류와 비트 패턴을 정의해서 어떻게 할건가 결정을 하고, 그 다음에 명령어들의 실행에 필요한 하드웨어를 설계하고요 그 다음에 각 명령어 실행을 위한 다양한 마이크로서브루틴을 작성한다.. 그리고 나서 이제 이거를 다 코드로 만드는 것이죠. 그래서 마이크로프로그램 코드들을 제어 기억장치에 저장한다.. 이렇게 되는 것입니다. 그렇게 되면 이제 마이크로프로그램 작성이 끝나는 것이죠.

2. 그러면 이 제어기억장치에는 어떤 것이냐? 즉 CPU 마다 종류가 매우 다양하다.. 이렇게 되있죠. 즉, 마이크로 프로그램이 다 다르다는 뜻입니다. 그 얘기는 하드웨어 구조가 다 다르다.. 이런 뜻이 되겠죠. 그러니까 명령어도 다르고, 또 마이크로 명령어도 다르고, 모든 게 다 다르다.

3. 그래서 주로 이제 ROM으로. 이게 메모리니까 ROM으로 구성이 된다. 이렇게 되겠고요. 그 다음에 ROM의 사이즈는 마이크로명령어 형식에 따라 마이크로프로그램 크기로 결정된다.

4. 이 말은.. 마이크로프로그램이 크면 ROM의 사이즈도 크다. 그러면 예를들어 CPU가 이만한데 ROM이 여기 안에 들어가 있다. CPU 내에.. ROM이 이게 커지면 CPU 내에 위치를.. 면적을 많이 잡아먹으니까, 결론은 ROM은 작아야 겠다. 작고 많은 게 들어가야 되겠다. 그런 것이죠. 그럼 작고 많은게 들어갈 수 없죠? 한계가 있죠? 그래서 마이크로프로그램은 굉장히 컴팩트하게 간결하게 짜야된다. 그런 결론을 얻을 수 있습니다.

5. 그러면 이제 제어기억장치. 한번 예를 보시면, 명령어 길이가 16 bits. 이거는 이제 마이크로가 아니라 그냥 cpu가 메모리에서 가져오는 그런 명령어를 말합니다. 그 명령어 길이가 16 bits 다. 그래서 op-code 부분이 4bits, 간접비트 1bits, 또는 즉치니 주소 이런걸 결정하는 오퍼랜드 부분이 11bits. 이렇게 되었다고 가정을 하면, 제어기억장치는 사이즈가 얼마나 되나.. 그걸 이제 보시면, 여기가 이제 ROM 이라고 보시면 됩니다. 얘가 이제 전체 ROM 이죠 이게.. ROM.

6. ROM 에.. 마찬가지로 메모리이니까, 주소 체계를 가지고 있습니다. 그래서 0번지부터 127 까지, 그러면 128개 있다는 뜻이 되겠죠. 그러면 이 폭은 어떻게 되느냐 이 폭은.. 이 폭은 이제 bit 수가 되겠죠. 그래서 마이크로명령어를 저희가 작성할 때 최소로 만들기 위해서 비트를 몇 비트를 해야되냐? 그래서 예를 들어 17 bits 를 구성을 해서 연산종류가 몇 개냐? 따져보니까 아.. 3 bits. 3 bits 면 여덟 가지죠? 그정도면 충분하다. 그런데 연산1과 연산2가 있습니다. 이 얘기는 연산1하고 연산2는 동시에 할 수 있다는 뜻입니다. 이게..

7. 그래서 연산1에 들어가는 종류, 그 다음 연산2에 들어가는 종류. 합치니까 한 8개, 8개.. 최대 16개 까지면 된다.. 그래서 3 bits 로 잡은 것이고 그 다음에 조건은 몇 가지가 있는지 따져 봤더니.. 한 4가지 정도. 그래서 이제 2 bits 로 만들고, 분기도 몇가지가 있는지 봤더니 한 4가지.. 그래서 2 bits.

8. 그 다음에 주소필드(ADF)는.. 여기 이제 주소필드입니다. 이거는. 주소. 그러니까 ROM 의 주소죠. ROM 의.. 주소필드인데.. 이것은.. 예를 들면 0번지에서 어디로 갈까? 저 밑으로 갈 수 있는..이런 거를 말하는겁니다. 어디로 갈껀가?

9. 그래서 주소필드는 0부터 127까지 128이니까 2의 7승. 그래서 7 bits 면 된다. 128 이니까.. 그래서 딱 ADF는 7 bits 로 작성을 했습니다. 그래서 전부 합쳐 보

니까 17 bits 다. 그래서 저희가 이제 2의 7승. 주소죠? 주소값에 곱하기 17 bits. 이만큼의 사이즈가 ROM 의 사이즈다. 이렇게 말씀을 드릴 수 있는 것입니다. 그래서 제가 여기 이렇게 적은 것이죠. 이렇게..

10. 그러면 여러분이 여태까지 보시기에 17.. 이건 2의 지수... 보통 저희가 2진수 할 때 2의 지수승을 쓰는데 이건 17이다. 뭐 그건 전혀 이상할 게 없습니다. 그러니까 컴팩트하게 짜다 보니까 굳이 이거를 2의 지수승으로 만들면, 16 또는 그 다음에 32 이렇게 가야하는데 너무 차이가 나니까 상관없이 그냥..이렇게 이제 사이즈를 저희가 결정할 수가 있는 것입니다.

11. 그럼 마이크로명령어 형식은 어떻게 되냐? 제가 앞에 계속 반복해서 말씀드리지만, 명령어 형식 얘기하고 마이크로 명령어 형식 얘기합니다. 즉, CU(Control Unit)안에 들어가는 메모리. ROM 에 들어가는 마이크로형식은 어떻게 되나? 비슷합니다. 명령어 형식하고.. 그래서 필드가 지금 17 bits 구성이 되있는 중에서 연산.. 앞에서 말씀드린 연산1 필드가 3 bits, 그 다음에 연산2, 그 다음에 조건, 분기, 그 다음에 주소 이렇게 해서 전부 17 bits 로 구성이 된거죠.

12. 그래서 연산1과 2에는 이제 설명을 드리겠지만, 앞에서 계속 해왔던 마이크로 오퍼레이션이라는 거. 그것의 종류를 여기다가 이제 비트패턴으로 1대1 매핑을 시키는 그런 과정이 되죠. 그래서 연산1과 2가 있으면 동시에 수행이 되는 거고, 연산1만 있으면 연산2는 아무것도 안하는거고.. 이런 형태가 되겠죠. 그래서 그거를 1대1 대응시켜서 이 비트패턴으로 3 bits 짜리가, 예를들어 000 이 들어가면, 아.. 얘는 뭐다.. 그 다음에 뭐 001이 들어가면 얘는 무슨 현상이다. 이렇게 알아볼 수 있게 만드는 것입니다.

13. 그러니까 표에 그렇게 만들어 놓는 것이죠. 그 다음에 조건도 2 bits 이니까, 00 일때는 무슨 조건.. 01 일때 무슨 조건.. 그런 조건을 쭉 나열하면 되는 것이고, 분기.. 마찬가지고, 주소는 아까 말씀드린 대로 그 메모리.. ROM 에 있는 메모리.. 0 에서 부터 127번지 까지 이 값에 따라 그 원하는 곳에가서 마이크로명령을 실행할 수 있게 만드는.. 그런 형태죠.

14. 그래서 이런 명령어 형식을 정리해보면, 연산1과 2필드는 그.. 두 개의 마이크로 명령어들이 동시에 실행될 수 있다는 것을 의미한다.. 그죠? 각각도 되고. 두개가 같이 써있으면 두 개가 동시에 실행되는 그런 의미입니다.

15. 조건은 분기 필드에서 사용될 조건을 명시한다. 분기는 분기의 종류를 명시하고, 조건에 따라 다음에 실행될 마이크로 명령어의 주소. 주소는 이거죠? 그 다음에 주소필드. 이거는 조건에 따라 분기가 발생하는 경우에 분기의 목적지 주소.. 같은 내용입니다. 다 이게.. 그러니까 이 3개를 한번 어울어서 보시면, 제가 이렇게 써놨지만, 이렇게 제가 말씀드릴 수 있는거죠.

16. 분기를 하고 싶은데.. 어떨 때 분기를 하겠느냐..그거죠. 그게 바로 조건인 겁니다. 조건이 맞아야 분기를 한다는 뜻이죠. 그러면 분기를 한다. 분기는 어디로 간다는 뜻인데, 어디로가? 어딘지 몰라요.. 그걸 결정해 주는게 얘다. 그래서 이 3개는. 이 3개는 같이 연동해서 보시면 되겠습니다.

17. 연산1과 2는 마이크로 오퍼레이션과 따로 보셔도 되지만, 앞으로 보실 때 조건, 분기, 주소는 제가 말씀드린 것처럼 분기하고 싶은데 무슨 조건일 때 어디로. 이렇게 해서 3가지를 같이 보시면 될 것 같습니다.

18. 예 이제 표로 정리가 된건데.. 여기 나와 있는 그 코드가 이 코드가 바로 그 아까.. 연산1 필드에 들어갈 2진비트 패턴입니다. 그래서 3 bits 되있었죠? 000부터 111까지..이 종류마다 1대1.. 얘하고 이게 1대1 매핑이 되있는 겁니다. 이게. 이렇게..

19. 그래서.. 많이 익숙한 거.. 뭐가 있죠? PC값을 MAR 에 준다. 그 패스를 결정할 때는 비트패턴을 뭘 준다? 001을.. 그 3 bits 에다 써주면 되는 것이죠.

20. 그래서 저희가 코드를 이렇게 쓱~ 쓰다보니까.. 코드가 이렇게 써있으면 100 그러면.. 이게 뭐지? 이런 느낌이 들죠. 그래서 얘를 좀 보기 좋게, 저희가 이해하기 쉽게 표현한 게 예를 들면, READ 라고 썼습니다. READ. 이 의미는 즉, MAR.. 이게 의미가 뭡니까? MAR 에 있는 그 지정된 주소값. 그 주소값에 있는 내용. 내용을 MDR로 가져와라. 다시 말하면, 메모리에 있는 내용을 CPU로 가져와라. 결국엔 메모리를 읽는다는 뜻이죠. 그래서 이제 READ 라고 쓴 것이고.

21. 이 경우는 PC 값을 MAR 에다 주니까, PC.. 이게 PC TO..이런 뜻이죠. TO.. 영어로 TO. 그 다음에 MAR.. 이거를 줄여 가지고 PC TO MAR. PC T.. M도 빼고 AR.. 이렇게 이제 쓴 것입니다. (PCTAR) 그렇게 기호를 하다 보니까 IR TO MAR.. 즉, IR에 있는 내용을 MAR에다 붙인다. IR에 있는 MAR에 준다. (IRTAR) 이렇게 기호화 한 것이죠.

22. 그 다음에 이제 WRITE 라는 것은 의미를 보시면 뭡니까? CPU에 있는 내용을 메모리에 쓴다는 것이죠. 그래서 CPU 에서 메모리에 쓸 때 나가는 그 출구가 뭐죠? MDR 이죠.. 그래서 MDR에 있는 것을 어디다? 메모리.. 그 주소 지정된 메모리에다 써라.. 그래서 저희가 이제 WRITE. 이렇게 쓴거죠.

23. 그 다음에 좀 다른 거.. ADD.. 이거는 뭡니까? 더하는 거죠. 그래서 그거는.. 더하는 건 뭐죠? 어큐뮬레이터에 있는 내용하고, MDR에서 가져온. 이런 내용들을 서로 더한다는 뜻이 되니까, 요렇게.. 된 거를 이제 ADD.

24. 그래서 이제 앞으로.. 저.. 가시면.. 이렇게 기호로 마이크로프로그램이 표현될 건데, 즉 PCTAR 이렇게 써있으면 이거는, 아.. 001, READ 되 있으면 100, 뭐 DRTAC 이렇게 되있으면 101, 이렇게 이제 1대1 매핑을.. 즉 대응관계를 갖고 이제 그..마이크로명령 형식의 연산1 필드에 아마 써놓게 될 것입니다.

25. 마찬가지로 연산2 필드.. 여기도 똑같습니다. 제가 이제 굳이 부연설명 안해도 이제 여러분이 보시면 아실텐데.. PC에다 하나 더한 거. PC를 증가시킨다. 인크리먼트피씨(INCPC) 이렇게 이제 기호를 만든 것이죠. 그 다음에 MDR에 AC 값 주는 거는 이건 AC TO MDR (ACTDR).. 이거는 다 이제.. 형태가 같죠. 이것도 스택포인터값을 MAR에다가 (SPTAR). 이건 이제 아까 플러스인 걸 ADD 라고 했으니까, 이제 이건 마이너스. 빼기는 이제 SUB 라고 저희가 쓴 것이고.

26. 그 다음에는 뭐 다른 기호의 표현에 특별한 건 없습니다. 그리고 여기 눈여겨 보실 게 이 NONE 라는 게 있습니다. NONE. None는 말그대로 아무 것도 안한다. 노 오퍼레이션 이라는 뜻이죠. 노 오퍼레이션. 즉, 아무 것도 안한다.. 그런 뜻이 되겠습니다. 그게 코드로는 이렇게.. 쓰는.. 그런 형태가 되겠죠.

27. 그래서 마찬가지로 앞에서 설명드린 연산1 이나 연산2 필드는 이렇게 연산에 따라서 1대1 매핑하는, 즉 이렇게 썼으면 이거를 100 이렇게 바꿔주시면 됩니다. 그런데 주의하실 점 하나가 연산1 하고.. 연산2 을 보시면, 여기 나와 있는 이 모든 마이크로 연산의 연산1 필드하고 똑같은 게 하나도 없습니다. 전부 다릅니다.

28. 단지 똑같다는 것은 뭐냐.. 이 None.. 아무것도 안하는 이거.. 이거만 공통으로 있고 나머지 부분.. 예를.. PC 에다가 하나 더해서 PC 값으로 변화시켜주는 이 내용은 연산1 필드에는 없습니다. 그러니까 여기에 맞는 거는 연산2, 앞에서 말씀드린 거는 그.. 연산1 에다가 비트를 써주면은 되는것이죠.

29. 그 다음에 이제 조건입니다. 조건 필드는 앞에서 제가 2 bits 라고 말씀을 드렸었는데, 2 bits 이니까 이제 네 가지를 저희가 결정할 수 있는 조건에.. 그러니까 여기 보면 U 한거는.. 무조건.. 언컨디셔널(unconditional) 이런 뜻이죠. 그래서 무조건 분기하는.. 조건이 없어요. 그냥 무조건 가는거에요. 그게 이제 저희가 기호로 무조건이라고 해서 U 이렇게 쓰는거고.

30. 그 다음에 I는 그 인다이렉트(indirect) 해서 간접사이클을 부를 때 쓰는 조건인데요. 이것도 I 가 1인 경우에만 간접 사이클 루틴을 호출하는 겁니다. 1이 아니면 안 부르죠.. 그게 이제 I 에 대한 I비트에 대한 조건인 거에요. 그러니까 I가 0 이면 이건 안 부르고, 1일때만 부른다. 이런 조건을 제시하는 겁니다.

31. 그 다음에 S 라는 것은 이제 그.. 2의 보수로 이제 사인비트..를 얘기하는 거죠. 그래서 누산기에 저장된 데이터의 부호. 즉 ALU 연산한 결과가 만약에 마이너스가 나왔다.. 음이 나왔다.. 그러면 그 사인비트가 1이 되거든요. 그걸 보고 결정을 하게 됩니다. 그래서 여기 데이터의 부호가 1이면.. 그 얘긴 이제 마이너스를 갖는다.. 이런 뜻이에요.

32. 그 다음에 이제 Z 라는 건 제로. 누산기에 저장된 데이터가 제로(0). 값이 없고 제로다 이러면 분기하는, 이런 조건을 제시하고 있습니다.

33. 그래서 여기 지금 표에 나와 있는 거.. 이런 건 2 bits 로.. 이렇게 나와 있으면 아.. 이건 싸인. 즉, 데이타 부호에 따라서 조건이 맞으면 분기한다.. 이런 개념으로 보시면 되겠습니다.

34. 그 다음에 이제 분기.. 그.. 필드.. 앞에서 조건을 말씀 드렸고, 그 다음에 분기 필드에 위치할 마이크로-연산 이게 있습니다. 여기도 2 bits 를 제가 앞에서 설정을 했다고 말씀을 드렸는데, 즉 JUMP 일때, CALL, RET(리턴), MAP 이렇게 되있습니다. JUMP 는 비트가 00, CALL 은 01, RET 은 10, 11. 이렇게 이제 두 비트가 설정이 되는 것이죠.

35. 그래서 이 분기 필드에 예를 들어 10 이다 그러면? 아 이건 리턴(RET)이구나. 11 이다 그럼 매핑이구나.. 이렇게 이제 알게되는 거죠. 물론 그게 다 조건이 맞아야 됩니다. 조건이 안맞으면 안 하게 되는 것이죠.

36. 그래서 여기보면 조건에 따라.. 이렇게 되있죠. 앞에서 제가 조건 설명드린 그 조건에 따라, 여기 CAR의 내용. 이걸 뭐라 그랬죠? 제가 프로그램카운터와 같은 역할을 한다 그랬죠? 이 내용을 주소필드(ADF). 주소에 있는 값. 주소필드에 있는 값, 혹은 이렇게 하나 증가시키는 값으로 저장한다. 이 얘기는..

37. JUMP를 하는데 바로 밑으로 JUMP를 할꺼냐. 즉, 하나 증가시키면 NEXT 그죠? 바로 밑으로 JUMP 할꺼냐 아니면, 주소필드에 어떤 값을 줘요. 멀리 JUMP 할때 그 주소값으로 JUMP 할꺼냐? 그런 뜻입니다. 이게. 조건은 무조건 조건이 맞아야 됩니다. 조건이.. 틀리면 안되요.

38. 그 다음에 이제 CALL 이라는 게 있는데, 01 이라고 세팅되어 있는 CALL 은. 이것도 조건에 따라.. 조건이 안 맞으면 안되죠? CAR 의 내용을 주소필드(ADF). 즉 여기 주소필드. CAR의 내용을 주소필드 값 혹은 플러스 1 한 값으로 저장한다. 이렇게 되겠습니다. 즉, NEXT로 가든지 아니면, 이 주소필드에 있는 값. 그쪽으로 멀리 분기한다. 그래서 거기있는 서브루틴을 처리한다.. 이런 개념이 되겠습니다.

39. 근데 CALL 은 서브루틴을 부르는 것이니까 반드시 돌아와야죠. 그래서 여기서 보면, 주소필드의 값으로 저장할 때는.. 다시 말하면 CAR 내용을 +1 이 아닌 주소필드로 할때는 그 쪽으로 가야되니까 다시 돌아와야 될겁니다. 그래서 돌아와야 될 주소. 즉, 복귀 할 주소 값을 SBR에 저장한다. 여기서 앞에서 제가 말씀드렸듯이 CAR은 프로그램 카운터 생각하시면 되고, SBR 은 스택을 생각하시면 됩니다. 즉, 돌아올 주소값을 반드시 가지고 있어야 돌아올 곳을 안다.. 이런 뜻입니다.

40. 그 다음에 이제 리턴이라는 것은 마이크로서브루틴으로부터 복귀. 이 RET 는 다른 경우가 없습니다. 어디 갔다가 돌아오는 경우에 쓰는 거니까, 반드시 마이크로서브루틴으로부터.. 끝나면, 그 서브루틴이 끝나면 그 다음에 이제 다시 원위치 하는 그런 분기 형태죠. 그래서 분기.. 원래.. 돌아오는 곳. 주소는 여기 저장을 했잖아요. SBR에.. 했기 때문에 여기 SBR에 저장된 내용을 어디로? 다시.. 프로그램

카운터에 줘야 되죠. 그 역할을 CAR 이라 그랬죠? 그래서 SBR에 저장된 내용을 CAR로 다시 적재한다. 이런 과정이 되겠죠.

41. 그 다음에 이제 MAP. MAP 라고 되있는 거는 사상방식(mapping)에 의하여 분기 목적지 주소 결정. 이거는 여기 지금 설명드린 거하고 다른데, 그.. IR 명령어의 op-code 부분. 즉.. 이거는 op-code 부분. 이 op-code 부분을 해독을 해서, 얘가 지금 LAOD 인지, ADD 인지, 또는 SUB 인지.. 뭐 이런 종류의 여러 명령어가 실행될 수 있는 그 루틴을 찾는 방식입니다. 이게. 그래서 이런 거하고는 조금 다르죠. 얘는.. 그래서 매핑은 제가 또 다음에 나오니까, 그때 자세히 설명을 드리도록 하겠습니다.

42. 그래서 이런 분기에 앞에서 설명드린 JUMP 나 CALL, RETURN, MAP 이렇게 네 가지. 얘를 이제 bits로 만들면, 이건 00, 01, 10, 11 이렇게 되있는 것이죠. 그래서 그거.. 이제 설명드린 그 내용을 이제 그.. 어떤 수식같은걸로 정리를 해본건데, JUMP 즉, 조건이 맞으면 입니다. 조건이.. 그러면 주소필드(ADF)에 있는 값으로 JUMP 하는 것이고. 욜루.. CAR을 바꿔 준다는 얘기는 그리로 JUMP 한다는 뜻이죠. 조건이 아니면 그냥 그 밑으로 가라. 그래서 하나 증가시키는 그런 뜻입니다.

43. 그 다음에 CALL 인 경우도 조건이 1 이면, 조건이 맞으니까 이 주소필드 값으로 CAR 을 바꿔줘라. 그러니까 그리로 가라 이런 소리가 되겠죠. 근데 CALL 에서는 반드시 갔다 돌아 와야 되기 때문에 SBR에다 뭔가를 줘야되요. 그 돌아올 정보를 줘야 되니까, 돌아올 정보가 현재 진행되고 있는게 CAR 이라면, 바로 그 다음 건 CAR +1 입니다. 그래서 CAR +1을 SBR에 주는 것이죠. 조건이 맞지 않으면, CALL에 해당되지 않으니까 그냥 그 다음으로 가라.. 이런 뜻이 되겠죠. 그러니까 여기 있는거나 같은 개념이 되는거죠. 조건이 안 맞으니까..

44. 그 다음에 이제 리턴(RET). RET 은 말씀드렸듯이 서브루틴으로 복귀하는거니까 그 복귀하는 주소가 누가 갖고 있죠? SBR 이 갖고 있죠. 그래서 SBR 값을 CAR 에다 주는 이런 형태가 리턴이 되겠죠.

45. 그 다음에 이제 매핑(MAP)은. 여기 지금 이렇게 수식으로 제가 표현했는데 이 뜻이 뭐냐면, 음.. 제가 여기다 그림을 그릴께요. 이렇게.. CAR 이 일곱 비트가 있습니다. 이렇게. 일곱비트. 이게 이제 CAR 이다. 레지스터다.. 이게. 그럼 얘가 번호가 이제 여기서부터 0,1,2,3,4,5,6 이렇게 되있죠? 이게 bit 번호입니다.

46. 그럼 이게 지금 이 밑에..밑에 첨자로 쓴 것이 비트 번호를 말합니다. 그대로 쓰시면 됩니다. CAR 에 6. 이 여섯번째, 6번째 bit 를 1로 써라. 그럼 여기에 1로 써라 이 소리에요 1.. 그 다음에 이건 좀 있다 하고요. CAR에 1과 0. 이건 0을 채워라. 0을 채워라 이런 뜻입니다. 0.

47. 그 다음에 이제 나머지, 5번 4번 3번 2번 비트는 뭘로 채우냐? IR의 op-code. 그렇죠. IR이 이렇게 있으면, 이 부분에 네 비트짜리 아까 예를 들때.. op-code 가 여기 이렇게 있었죠? op-code..

48. 이 op-code 에 들어가는 부분이 예를 들어 0101 네 비트가 이렇게 들어가 있다. 그럼 이 비트를 어디로? 일로 집어넣어라 그래서.. 0101. 이렇게 쓴다.. 이렇게 얘기 한다는 겁니다.

49. 그럼 이 말은 뭐냐? 즉, CAR의 값이 결정이 된 겁니다 이게. 즉, 항상 여기는 1이고, 1이다 이 얘기는 이 자리가 64다 이런 뜻이죠. 2의 6승이니까.. 64. 그 다음엔 여긴 00 이니까 이 안에 있는.. 지금 여기 표시된 이값, 이값, 이값, 이값. 이 네 비트. 이것이 00 이면, 이 값은 항상 64가 되겠죠. 64.

50. 그 다음에 0001 이면.. 이게 하나 증가하면, 2의 2승. 4가 증가하니까 68이 되겠고. 그 다음에 4씩 증가하게 되있죠. 72. 이렇게 증가하게 되겠죠. 그런 경우에 얘는 얼마입니까? 여기 4가 증가했고, 이거는.. 16 이니까, 64에다가 16, 4.. 20 그러니까 84가 되겠죠. 이게.. 2의 4승이니까 16. 얘가 4. 그래서 이걸 다 더하면, 여기 20하니까 84가 되겠죠. 즉 이 op-code가 0101 이다 그러면 84라는 값을 얻게 되는 거죠. 84. 그러니까 84가 뭐냐?

51. 즉, 이 매핑(MAP) 라는 뜻은 이렇게 연산을 해보니까 결과가 CAR 값에 84가 들어가는 결과. 즉, CAR에 84가 들어갔다는 얘기는 무슨뜻이죠? 이게 프로그램 카운터 역할을 하니까, 84번지로 가라.. 이 소리 입니다. 이게. 그런 역할을 하는 게 바로 이 매핑이라는 거죠. 매핑. 그러니까 여기 JUMP나 CALL. 이거하고는 좀 다르죠. 리턴도 좀 다른 형태죠? 무조건 SBR을 일로 주는 것이..

52. 결국은 지금 여기 나와있는 이 설명에 되어 있는 이 내용은 결국 뭐냐? CAR 에다가 어떤 걸 할꺼냐? 이걸 결정하는 겁니다. 지금. 즉, 00일때, 01일때, 10일때, 11일때 각각 다 무조건 CAR 값을 뭘로 할까.. 이거를. 얘를 결정해주는 방법이 이거다. 이렇게 보시면 되겠습니다.

53. 그래서 저희가 이제 이걸 마이크로프로그램이라는 걸 한번 실질적으로 작성을 해보면, 인출사이클. 제일 처음에 제가 그 컴퓨터구조 그림 드리고, 거기다가 명령어를 가져오는 과정이 어떻게 되나요. 컴퓨터 시작하는 것이 바로 이 인출 사이클라는 것..

54. 지금 여기까지 왔습니다. 인출사이클을 저희가 뭘로 표현했죠? 마이크로오퍼레이션으로 표현했었죠. 여기 지금 익숙한 MAR 에다가 PC 값 주고, MDR 에다가 메모리 지정하는 값 주고 PC 하나 증가시키고, IR 에다가 MDR 주고.. 이걸 여태까지 했습니다. 이제 이거를 프로그램 하자. 이 말이에요. 이거를..

55. 그러면, MAR 에다가 PC값을 준다. 앞에서 제가 연산1 필드에서 그랬죠. PC TO MAR (PCTAR) 이라는 기호.. 이거 있었잖아요 이거..

56. 그 다음에 여기는 아무것도 없어요. 여기는.. 그러니까 NONE. 이렇게 되있는 것이죠. 그 다음에 여기도 보면 여기. MAR값.. MDR로 이걸 보내주는 거니까 뭐에요 이거. READ 죠. 그래서 READ. 여긴 있네요? 프로그램 카운터 하나 증가해서 여기.. 이렇게 기호 썼죠?

57. 그 다음에 MDR 값을 IR로.. 여기 있네요. 그러니까 여긴 없죠. 이렇게 해서 NONE 쓴거고. 이게 바로 이제.. 이게 바로 연산필드 1이고, 이게 연산필드2가 되는 것이죠. 이렇게 기호로 표현을 한겁니다.

58. 그리고 이제 조건을 보니까 여기가 U 가 되있어요. U. U 는 뭐냐? 즉, PC 값을 MAR로 주고 나서 뭘하죠? 없어요 할께.. MAR에 직접 내용을 가져오는 거 밖에 없어요. 그 바로 밑으로 그냥 가는 거에요. 그러니까 무조건 JUMP를 하는데.. 어디로? NEXT 로.. 가자 이거에요. NEXT 로..

59. 그럼 NEXT가 어디에요? 이 다음이죠. 다음.. 그러니까 얘는 어디로 가냐면, 욜로 가는 거에요. 그래서 연산하고 또 여기도 무조건 JUMP 해. 어디로? NEXT 로. 그런 뜻입니다. 이게.

60. 여기도 마찬가지로 MDR. 이거 주고, PC값 증가시켜. 그 다음에 뭐 할께 없어요. 왜? 인출사이클 이니까. 명령어만 가져다가 어디까지? 인스트럭션 레지스터에 갖다 주면 됩니다. 그게 목적이니까.

61. 그래서 무조건 JUMP JUMP NEXT 로 가는 것이죠. 그래서 마지막에 딱 갔어요. MDR을 IR로. 얘는 이제 인출사이클 끝난거죠. 근데 여기는 맵(MAP)이라는 게 있어요 맵. 분기하는데 무조건 MAP 을 하는데, MAP 뭐냐? 아까 제가 말씀드렸죠. 이 MAP 이.. 뭐죠? CAR 값을 결정하는 거에요. 결정하는.

62. CAR 값이 뭐냐? 여기다 뭘 줄꺼냐 결정하는 과정이죠.. 그래서 7 bits 로 구성이 되있는데, 제일 마지막.. 제일 위에 MSB 죠. 거기는 1을 주고, 마지막 두 비트는 0을 주고, 네 비트는 뭘로 채울꺼냐? 그걸 이제 이 인출사이클에서 명령어를 가져왔죠. 명령어를 가져와서 지금 여기 IR 에다 갖다 놨잖아요? IR. IR에 갖다 놨어요.. 갖다 놓고 보니까 아.. 여기 op-code 부분이 보여. 네 비트. 그거를 아까 CAR 조합한 MAP 라는거. 그 위치에다 갖다 집어 넣는거죠.

63. 그러면 이 MAP 은 명령어를 가져오고 나서 MAP 하니까.. 아.. 얘가 어디 있구나..를 찾아서 CAR 값에다 주니까 그쪽으로 가는 거에요. 이해가 되시는지 잘 모르겠습니다만, 어쨌든 그런 형태입니다.

64. 그리고 이제 여기 FETCH 라고 되있는 거 이거는 이제, 루틴의 이름이죠. 즉, FETCH에 관한 루틴이다. 인출사이클 루틴이다. 이런 개념인거고.

65. 그 다음에 여기 나왔는 이 어드레스 이 부분.. 이 부분은 ROM의 주소값 입니다. 이거 실제로 코드가 들어가는 게 아니라, 이건 ROM의 주소 번지죠. 즉, 마이크로 프로그램 코드는 이거죠 이거. 이거.. 이게 하나, 즉 0번지에 이 마이크로 프로그

램이 들어가 있는거고, 그 다음 번지 1번지에 이게 들어가 있는 거고, 그다음 번지에 이게 들어가는 겁니다. 그러니까 인출사이클은 보시면 알겠지만 0번지, 1번지, 2번지.. 해서 3개의 번지를 차지하고 프로그램이 된 상황인거죠.

66. 그 다음에 여기 NEXT,NEXT 이 말은, 이걸 실행하면 그 다음에 NEXT로 가니까 ADF 에다가 1번지를 써주니까 바로.. 이거 NEXT가 이거죠? 1번지 써주고.. 그 다음에 이걸 실행하면 또 NEXT 니까, 2번지. 그래서 여기다 2번지.. 여기다 2번지를 써 준거고. 그 다음에 여기는 이제 MAP 에요. MAP이 뭐에요? 분기쪽에 11 그러면 MAP이 거든요. 이게 MAP 에요. MAP. 표에서 보시면..

67. MAP는 어디로 분기해요? 이거하고 상관이 없어요. MAP 보시면, ADF 하고는 상관이 없어요. 그러니까 이건 그냥 0으로 채워 준거에요. 이건 의미가 없습니다. 이거는.. 아시겠죠? 그래서 MAP는 어쨌든, CAR을 결정하는 그런 형태의 분기다. 이렇게 보시면 되겠습니다.

68. 그 다음에 이제 간접사이클 루틴. 이것도 이제 실행사이클이 아니라 간접은 제가 앞에서 말씀드렸듯이, 뭐.. LOAD나 STORE 명령에서 그 메모리에 있는 값을 가져오려고 봤더니.. 그 주소에 갔더니 데이터가 아니고, 주소값이더라. 그러면 한번 더 해야 되는 그런 형태로 제가 간접사이클을 설명 드렸습니다. 그래서 명령어를 가져온 형태에서.. 인출한 형태에서 IR에 있는 오퍼랜드. 그 필드에 어드레스 값을 MAR에 받고 그것이 가리키는 주소번지에 가서 MDR에 가져와서 다시 MDR 을 IR로 가져가는 형태. 그것이 바로 이제 간접사이클 루틴인거죠.

69. 그래서 얘도 마찬가지고. 간접사이클 루틴은, 그.. 필요한 경우에 LOAD나 STORE에 필요한 경우에 일로 가게 되는데 이쪽으로 오면 이것 실행하고 이거 실행하고 그죠.. 그래서 JUMP도 JUMP NEXT, JUMP NEXT 무조건 이렇게 되있죠. 그래서 실행이 끝나면, 무조건 어떻게 합니까? 리턴하는 겁니다. 서브루틴이 실행해서 얘는 끝났습니다. 그러니까 원래로 가야죠. 그래서 여기 이제 리턴이다. 그러니까 리턴은 뭐였죠? 제가 앞에서 설명드린 대로 그 SBR에 있는 값을, 이거 이 리턴은 CAR 에다가 뭐를..? SBR 값을 주는.. 이런 형태죠. 복원시키는 거죠.

70. 그럼 여기도 마찬가지로, 마이크로 오퍼레이션을 하면 IR에 어드레스 값을 주니까 요렇게. 그 다음엔 여긴 없어요. 다.. 여긴 다 없으니까, 이 안에는 다.. 연산2는 다.. NONE 이 되는 것이고. 그 다음에 이거는 메모리를 읽어오니까 READ 고, MDR 꺼를 IR로 주니까 기호로 다 이렇게 씁니다. 그러니까 이거에 해당하는 그.. 비트가 이거. 이것에 해당하는 비트가 이거. 이것에 해당하는 비트가 이거. 이렇게 이제 작성이 되는거죠.

71. 그 담에 이쪽은 다 NONE 이니까 다.. 제로고. 여기 다.. 제로고.. 컨디션도 무조건 이니까 다 제로고.. 이렇게 제로 제로. 그 담에 브랜치도 JUMP니까 다 제로 제로. 제로 제로. 근데 여기만 리턴이죠. 리턴은 10 이죠. 그래서 여기가 10이 들어갑니다.

72. 그 담에 이 주소값도 NEXT 니까, 요번지가 설정된게 지금 여기 ORG 4로 되어 있습니다. 즉, 4번지부터 시작한다.. 그래서 여기가 4번지, 5번지, 6번지 이렇게 되있는것이죠. 4,5,6. 그래서 NEXT 이니까 이 값을 5로 표현이 된것.. 바로 일로 가는거죠? 얘도 NEXT 니까 6. 6번지값 일로 가는것이죠. 그 다음에 여기도 리턴하니까 이게 뭐에요. 이거 의미없는 없는겁니다. 리턴이니까. 리턴은 뭐에요? 이거니까.. ADF 하고 상관이 없죠. 어드레스 필드하고는 상관이 없다.. 그랬습니다. 그래서 이거는 의미가 없다. 이렇게 보시면 되겠습니다.

73. 그래서 이거는 뭐냐? 아까 인출사이클 루틴을 0번지부터 0,1,2 뭐 이렇게 했으면.. 얘는 그 다음에 ORG 4.. 4번지부터 또 작성이 되있는 것이죠. 이런게 하나의 루틴으로 작성이 되있습니다. 이런것들이..

74. 그래서 앞에서 말씀드린 것처럼 ROM이 이렇게 있으면, 저희가 보통은 이렇게 반 잘라서, 이쪽은 이제 공통적으로 실행되는 루틴들을 주로 이쪽에다 작성을 하고.. 그 다음에 아까 64되어 있는 이거.. 이 이후에는.. 이제 실행 루틴. 그러니까 개별적.. LOAD 인지 ADD 인지 SUB 이런 것들을 작성하는.. 개별적으로 여기에 다 작성을 해놓는 겁니다.

75. 그래서 앞에서 말씀드린 매핑이라는 거는, 인출한 다음에 처음에 인출하고 여기서 찾게되죠. 여기에 실행사이클들이 쭉 많은데, 어느건가 해당해서 찾아가는.. 이것이 바로 매핑(MAP) 인거죠. 제가 매핑은 계속 설명을 드릴겁니다.

76. 예, 이제부터 매핑에 대해서 제가 설명을 드릴께요. 실행 사이클의 마이크로서브루틴을 찾기 위한 사상(Mapping). 매핑 방법이다. 그래서 여기서 이제 앞에는 인출사이클이었죠. 이제 실행입니다. 실행.

77. 즉, 명령어 내의 op-code 가 지정하는 연산을 실행하기 위하여.. 이말은 명령어 내의 op-code 가 뭐.. ADD면, 이거를 하기 위해서 연산을 실행하면서 제어기억장치 내에 실행 사이클의, 즉 실행 사이클의 마이크로서브루틴 프로그램 되어 있는. 여길 찾아가는 것입니다. 찾아가기 위해서는 거기의 주소를 알아야 되겠죠. 그래서 시작 주소를 찾아가는 방법이다.. 이렇게 얘기할 수 있는 겁니다.

78. 명령어 op-code의 비트를 사상함수의 특정비트 패턴과 조합하는 방법이다. 이 말은 아까 이렇게 일곱 비트가 CAR이 일곱비트가 있었으면, 제가 반복해서 말씀드리는데, 여기는 1, 여기 두개는 0, 조합한다. op-code를 여기다가 집어넣는다. 이런말이죠. 해당되는거를.. 그래서 전체적으로 이 CAR 값이 얼마인가를 보는것이죠. 그런식으로 찾아가는 게 바로 매핑입니다.

79. 이걸 보시면 이제 쉽게 이해가 되실겁니다. Instruction register. 즉, 명령어가 들어왔죠. 명령어 인출하고, 그러면 앞에 네 비트가 있고 이 뒤에는 뭐에요? 오퍼랜드 뭐.. 간접비트도 있고, 오퍼랜드.. 여기서는 중요한 게 이거죠. 이 op-code

를 가지고 옵니다. 그래서 Mapping function 에다 이제 집어 넣는거죠. 이 네 비트를 이렇게 집어넣으니까, 이거는 어떻게 변하냐면, 요렇게 변한다.

80. 그러니까 처음엔 1, 그 다음에 X, X, X, X. 이거는 뭐에요. 여기 들어오는 값이 그대로 바뀐다는 뜻이죠. 끝에 두 비트가 00. 그래서 이건 2진 표현인데, 이걸 저희가 계산하면 1 곱하기 2의 6승 플러스 여기 X 곱하기 2의 5승. 이렇게 되겠죠? 쭉~ 해서, 결과값을 10진수로 이제 찾아내는 거죠. 그래서 최종적으로 보면, 여기 비트.. 이 네 비트가 뭐에 해당하느냐에 따라서 64에다가 더해진 값이 바로.. 그 실행 사이클루틴의 시작 주소가 되는것입니다. 이런 방식이 이제 매핑 방식인거죠.

81. 그래서 저희가 이걸 한번 넣어보면, 아무것도 안한 노. 노오퍼레이션(NOP). 이거는 op-code로 이렇게.. 그래서 여기다 00 집어 넣으면 얘는 64. 아.. 그러니까, 아무것도 안하는 노오퍼레이션은 64번지에서 시작을 하네.. 이렇게 알 수 있는 거죠?

82. 그담에 LOAD라는거는 0001. 집어 넣으면 계산하면 68. 아.. LOAD 명령은 68에 있다. 이렇게 쭉~ 결과를 저희가 뽑아낼 수 있는 거죠. STORE, ADD, SUB, JUMP. 다 마찬가지로 68, 72, 76, 80, 84 이렇게.

83. 그러니까 ADD 라는 명령을 만나면, 인출해서 앞에서 MAP 이라는 그, 그.. 브랜치가 있었죠? 그 MAP을 만나면, 걔는 어디로 가냐면 계산을 해서 CAR값에다 뭘 주냐? ADD를 만나면, 즉 76 이라는 값을 CAR에 준다.. 이런 의미가 되는 것입니다.

84. 마이크로프로그램. 이제 실행사이클 한번.. 실질적으로 어떻게 되어 있나 한번 하나씩 살펴보기로 하죠. 노퍼레이션(NOP). 아무것도 안하는 실행 사이클입니다. 그래서 여기 ORG 64. 즉, 64번지부터 시작한다. 그래서 어드레스 값이 여기. 64. 여기 2진수를 10진수로 바꾸면 64가 되죠.

85. 그러니까 아무것도 안하니까 NONE. 여기.. 연산필드도 아무것도 안해야 되는데 봤더니..그러면 뭐가 안되죠? 그래서.. 일단 얘는, 명령어를 가져와야 되니까 프로그램카운터는 증가시켜야 된다. 그건 하나 해야 되겠죠. 그래서 이렇게 들어가는 거죠. 즉, PC값을 인크리먼트 시켜라.. 그거죠. 그담에 무조건 JUMP 해라. 무조건. 00 이죠. JUMP. 00. 어디로? 어디로 JUMP 하나요? 어디로 JUMP 하나요? 갈 데가 없죠. 이 놉(NOP)이라는 오퍼레이션은 실행사이클., 이걸로 끝난겁니다.

86. 그럼 끝나면 어디로 가죠? ADD 라는 명령을 다 실행했어요. ADD 라는 명령을 생각해보시면, 처음에 명령어 인출해서 가져왔어요. 근데 봤더니 ADD 에요. 그럼 ADD에 실행.. 뭘 하겠죠? 끝나면 어디로 가요? 또 명령을 가져와야죠. 그래서 끝나면 여기 그.. FETCH 로 가고, FETCH. 여기 FETCH 로 되있죠. FETCH는

ROM 에서 가장 위에. 0번지에 구성이 되니까 그리고 다시 가는 겁니다. 왜? 새로운 명령을 불러오기 위해서. 아시겠죠?

87. 그 다음에 이제 LOAD 라는 명령. LOAD 라는 명령은 68. 68에 되있다. 그래서 여기도 보니까, 근데 좀 다른게.. 저희가 LOAD 라는 명령. 이거만 했어요. IR에 있는 값을 MAR, MAR이 지정하는곳을 MDR, 그 MDR을 AC로.. 이거만 했죠. 이건 없었습니다. 이거는.

88. 그래서 제가 설명을 드리면 이건 뭐냐면, I 라는것 I. 즉, 한 비트를 저희가 간접비트라고 그러죠. 간접비트. 즉, 간접 사이클 루틴을 갈꺼냐 말꺼냐를 결정하는 겁니다. 그러니까 그런 경우가 있죠. LOAD 에서 주소값을 받아 가지고 이제 거기 있는 값을 가져오려고 봤더니, 그게 또 주소값이에요. 그 경우는 뭐에요. 간접사이클에서 얘가 세팅 된거죠. 1로 세팅된 거죠.

89. 그러면 가서 주소값을 간접사이클을 통해서 다른 주소값 정보를 받아서 거기에 있는 값을 가져와야 되니까.. 얘는 인다이렉트(INDRT)라는것을 부르게 됩니다. 이게. 제가 앞에서 설명드렸죠? 간접사이클 루틴. 거길 부르면, 어떻게? CALL.. 일로 가야죠? 그래서 얘는.. 보세요. 아무것도 안해요. 여기서 연산에서는. 단지 얘만 체크해요. 그래서 I를 봐서 I가 조건이 맞나? 아.. 봤더니, 간접.. 조건이 맞아요. 맞으면 어떻게 해? CALL 을 해. 뭐를. 얘를.. 여기가 어디에요? 간접사이클 루틴이 있는곳. 그리로 CALL을 하겠죠. 얘가.

90. 그럼 갈 때 어떻게 합니까? 간접사이클이 들어오면 거기 아까 보셨지만, 거기에 리턴되있죠. 리턴을 만나면 어떻게 하라고? SBR을 CAR로. 줘야 되니까.. 그때 하기 위해서 여기서 바로 뭐냐면, SBR 에다가 CAR 플러스 1을 해주고 가야죠. 자기만 가면 안되죠. 돌아오질 못하니까.

91. 그러니까 갈 때는 어떻게 되죠? 조건이 맞으면 CAR에다 뭘 주죠? 여기 INDRT라는 이 주소값이 있는 곳. 그리로 가라는 것이죠. 그 값을 여기다가 줘요. 그게 아마 4번지가 되는데, 그 값을 여기다가 주고 간 다음에 돌아올 때는 이걸 반드시 해야 되는거죠. 그래야 돌아올 수 있으니까.

92. 그러고 오면, 다시 왔죠? 그러면 거기와서 돌아오니까 이 리턴할때 SBR. 그 반대로 여기 올때는 CAR 에다가 뭘 주죠? 리턴을 주게 되죠.. 아니, 저 SBR 값을 주게 되죠. 그럼 바로 이 SBR 값이 뭔가요? 요기죠 요기. 욜로 돌아오는 겁니다. 그래서 여기부터 실행을 하는거죠. 그래서 이거 한다음에, 이거 한다음에, 그래서.. JUMP NEXT, JUMP NEXT. 나머지는 뭐 이거는, 이제 여러분 말씀 드렸으니까 이거는 제가 굳이 설명을 안 드려도 이해를 하실겁니다.

93. 그래서 JUMP NEXT, JUMP NEXT, LOAD가 이제 끝나가요. 그러니까 이거 하면 다 끝나죠? 끝나면 JUMP 어디로? 다른 명령을 또 가져와야 되니까 역시 또 FETCH로 간다. 이렇게 되는거죠.

94. 그래서 이거를 전부 주소값. 68번지 이니까 여기가 68이 되겠죠. 여기가 68. 68에다가 연산, 연산, 하나쓰고. 이 조건이 맞는.. 그대로 이제 이 기호를 비트로 바꿔주면 되는 겁니다.

95. 그러면 얘는 68, 69, 70, 71. 이 네 개의 이렇게 프로그램이 하나,둘,셋,네 개. 이렇게 작성이 되는거죠. 그러니까 이게 뭡니까? 여기 LOAD 라고 되있는.. 여기다가 이제 이름을 붙이면 예를 들면, LAOD 실행 루틴이다. 이렇게 되는 것이죠. 이렇게 그룹을 만드는 거예요. 이렇게..

96. STORE ? 마찬가지죠. STORE 도 여기 보면 아무것도 않아죠. 저흰 이거만 했었죠. STORE 는.. STORE 는 내용이 뭡니까? AC 에 있는 내용을 메모리에다 저장하는 관계죠. 그 과정에서 그 IR. 이 주소값 주면, AC 값이 MDR로 갔다가 MDR을 메모리에 저장하는.. 이런 형태죠.

97. 이 세 개만 했는데, 이제 이것도 간접인지를 봐야되니까 여기도 마찬가지 NONE NONE 하고, 간접이냐 물어보는 거죠. 간접이면 CALL 해서 간접사이클 루틴으로 가자. 관계없으면 안가요. 안가면 바로 내려오면 되니까. 그죠?

98. 그래서 여기 내려오면 JUMP NEXT, JUMP NEXT. 마찬가지에요. STORE 다 끝나면 어디로 갑니까? 또 명령을 가져와야 되니까 이제 FETCH 로 가는거죠. 그 다음에 여기 ORG 72 되있으니까 72. 여기가.. 72. 73, 74, 75. 각 여기에 이렇게 프로그램이 작성이 된거죠. 이렇게. 하나씩. 이렇게. 네 개. 그러면 또 하나 만들었으니까 뭐라 그럴까요? 이거를 STORE 이렇게 얘길 하죠. 그럼 또 실행 루틴 하나 만들어 진거죠.

99. 그래서 ROM에 이제 64번지 이후로 이런 것들을 쭉~ 만들어 가는 겁니다. 이렇게.

100. 그 다음에 이제 ADD 명령. 제가 뭐 많이 설명.. ADD는 설명 안 드려도 이거는 간접사이클 그게 없죠. 이거는. AC 하고 뭐하고 더하는 겁니까? MDR 하고 더하는 거니까 상관이 없는거죠. 일단 가져올 때, 저장할 때 문제지. AC는 CPU 내에서 ALU 연산하는 과정이니까.. 그죠?

101. 그래서 얘는 그.. 마찬가지로 조건.. 무조건 JUMP, 무조건 JUMP, 무조건 JUMP. 어디로? FETCH. 이렇게 되는 겁니다. 특이사항이 이거는 없는 거죠. 이거 마이크로 오퍼레이션 해당하는 거. 이거를 여긴 아무것도 없죠? 그러니까 연산2는 다 제로. 이거에 해당하는 거 표에서 찾아서 비트로 작성해주시면 되고.. 76번지니까 여기가 76. 그렇죠? 그래서 이것도 3개. 해서 마이크로 프로그램이 작성이 된거죠 이게.

102. 그 담에 SUB 명령? 마찬가지죠. 이거만 하나 바뀐거죠. 이거. 플러스에서 마이너스로. 마찬가지로 얘도 무조건 JUMP, 무조건 JUMP 인데 다.. NEXT, NEXT, FETCH. SUB 끝나면 명령은 무조건! 실행 루틴은 끝나면 무조건 FETCH. 왜? 명

령어 실행이 다 끝났으니까 다음 명령을 가져와야 되죠. 그런 관계입니다. ORG 80 이니까, 여기가 80. 81,82 이렇게 되는거죠. 그래서 이것도 마찬가지로 하나, 둘, 세 개. 작성이 된겁니다.

103. 마지막으로 JUMP 명령어-실행 사이클의 마이크로서브루틴 보겠습니다. ORG 84 니까 이게 84. 이거. 됐고, 여기도 아무것도 안하죠? JUMP는.. 즉, PC값을 IR에 있는 어드레스 값으로 바꿔주는 역할 밖에 안하니까, 첫번째 NONE 니까 여긴 제로 제로. 그 다음에 IR에 있는 값을 PC로.. 그래서 이렇게 바꿔주고, 무조건 JUMP. 무조건 JUMP. 표시하고, 그다음에 어디로 가나요?

104. 이건 NEXT 가 아니죠! 끝나니까 이제 다시 명령어 인출로 가야 되죠. 그래서 FETCH. 그래서 여기가 000.. 이렇게 되는거죠. 이상이 이제 실행 마이크로 그.. 서브루틴들의 종류를 제가 한번.. 다는 아니지만, 앞에서 나왔던것들을 한번 쭉~ 마이크로. 작성을 해봤습니다.

105.