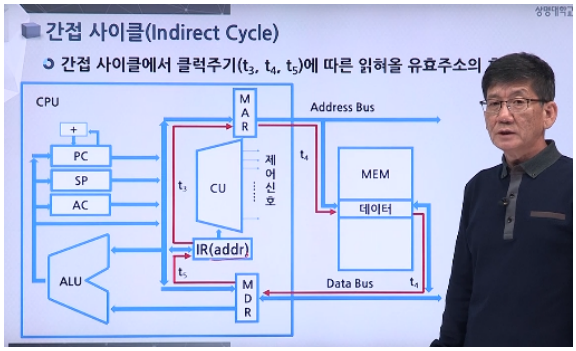
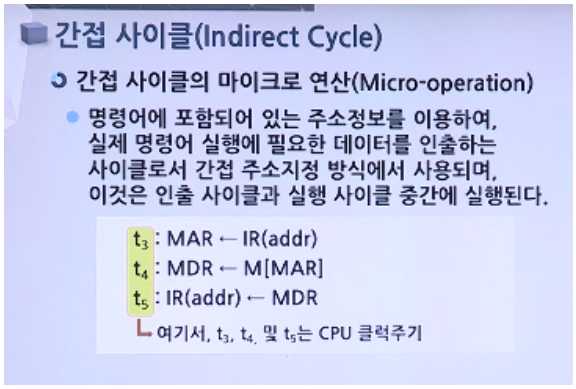
15강_간접사이클 및 종합실행 예제

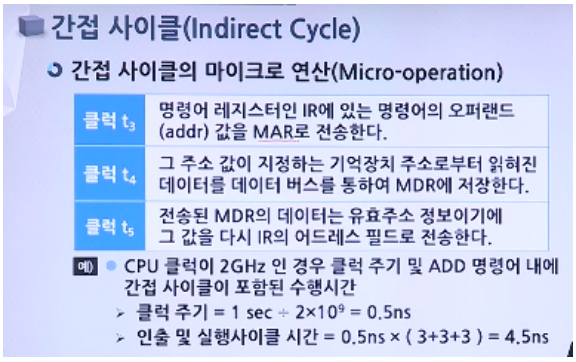
Created	@Aug 11, 2020 5:23 PM
Tags	

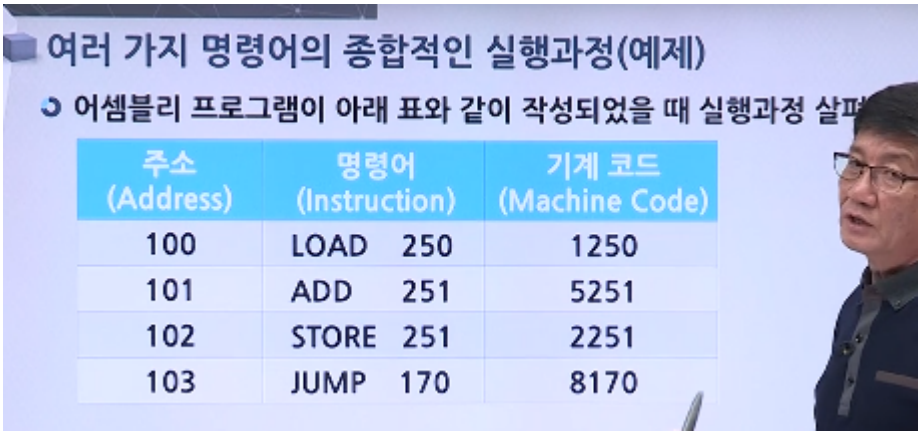


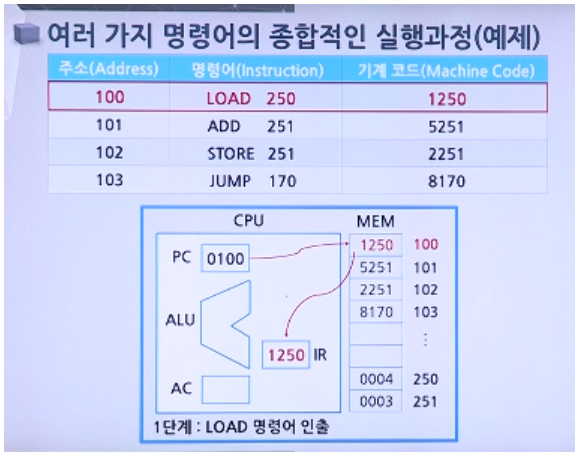
명령어 내에 있는 간접 사이클이 체크 되었을 때 이 간접 사이클을 어떻게 하는지 보여주는 그림(~유효주소의 흐름도)

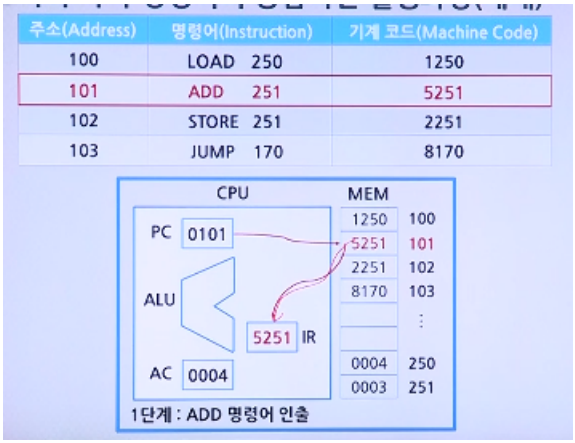
EX) LOAD명령 인출, addr 정보를 따라갔을 때 데이터가 아니라 또다른 addr를 가지고 있어 추가되는 사이클(간접사이클)일 때 어떻게 하느냐
: 첫 사이클을 했을 때 데이터가 아니기 때문에 AC에 바로 가지 않고, addr의 과정을 한 번 더 거친다

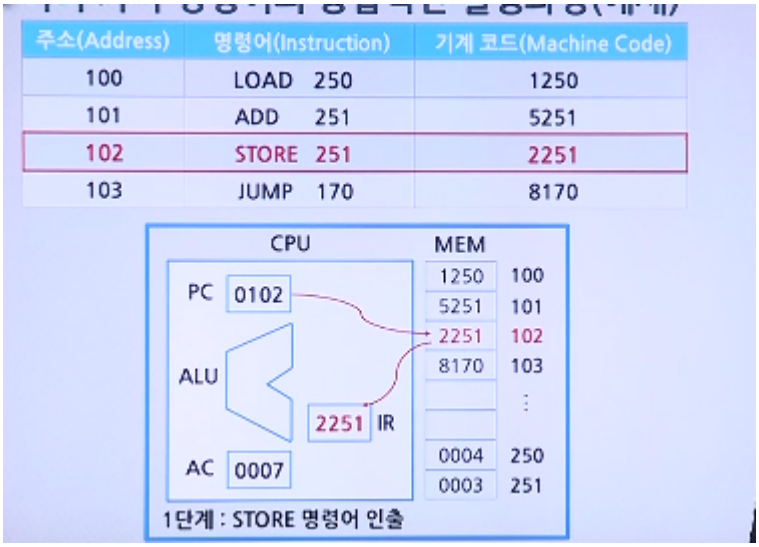





여러 가지 명령어의 종합적인 실행과정(예제)
어셈블리 프로그램이 아래 표와 같이 작성되었을 때 실행과정 살펴
주소 (Address)
명령어 (Instruction)
기계 코드 (Machine Code)
100 LOAD 250 1250
101 ADD 251 5251
102 STORE 251 2251
103 JUMP 170 8170


여러 가지 명령어의 종합적인 실행과정(예제)
주소(Address) 명령어(Instruction) 기계 코드(Machine Code)
100 LOAD 250 1250
101 ADD 251 5251
102 STORE 251 2251
103 JUMP 170 8170
CPU
MEM
PC 0100
ALU
1250 IR
AC
1250 100
5251 101
2251 102
8170 103
0004 250
0003 251
1단계 : LOAD 명령어 인출


주소(Address) 명령어(Instruction) 기계 코드(Machine Code)
100 LOAD 250 1250
101 ADD 251 5251
102 STORE 251 2251
103 JUMP 170 8170
CPU
MEM
PC 0101
ALU
5251 IR
AC 0004
1250 100
5251 101
2251 102
8170 103
0004 250
0003 251
1단계 : ADD 명령어 인출

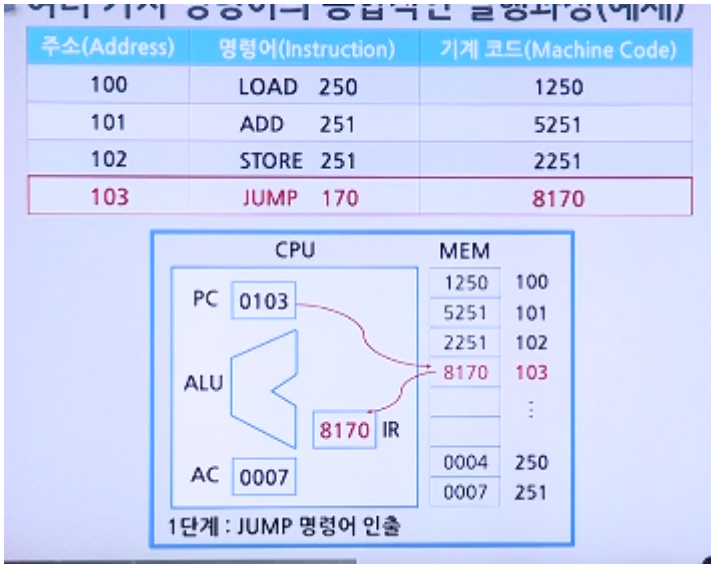

주소(Address) 명령어(Instruction) 기계 코드(Machine Code)
100 LOAD 250 1250
101 ADD 251 5251
102 STORE 251 2251
103 JUMP 170 8170
CPU
MEM
PC 0102
ALU
2251 IR
AC 0007
1250 100
5251 101
2251 102
8170 103
0004 250
0003 251
1단계 : STORE 명령어 인출

▼ 설명

1. 그러면 이 하나씩 한번 실행될 때 그림이 어떻게 변하는지 살펴보도록 하죠. 먼저 100 번지에 있는 LOAD 250. 즉, Machine Code 로는 1250 이렇게 되있죠. 이것이 수행이 되면, 제일 먼저 이제 PC 가 100 번지. 즉, 100 번지를 가리키는 명령어를 가져와라. 이런 뜻이 되겠죠. 물론 여기다가 MAR, MDR 은 표시 안했습니다.
2. 그럼 PC 가 100 번지를 가리키면 여기에 명령어가 이 Machine Code 가 들어가 있는 것입니다. 1250 이 들어가 있고. 그러니까 인출을 하게 되니까, 이 명령어가 메모리에서 CPU 내로 오면서 IR 레지스터. Instruction Register 로 들어오게 되죠. 그 과정이 이제 LOAD 명령어를 인출하는 첫 번째 단계 입니다.
3. 다음 단계는, 이 LOAD 명령이 실행단계가 되겠죠. 그 실행에서 그, 이미 그 단계 1 에서 PC 의 값은 사실은 자동 증가해서, 즉 PC 에다 플러스가 되서 101 이 됐고, 그리고 이제 명령어 1250 이란 LOAD 란 명령이 들어가 있는데 얘를 해석해 보니까. 아 250 번지에 있는 내용을 Accumulator 거기다 저장을 해라 이런 뜻이니까, 250 번지에 있는 이 내용. 즉, 0004 라는 4죠. 4 값을 Accumulator 에 가지고 왔습니다. 이것이 이제 LOAD 명령어의 끝이죠. 그러므로써 이제 이 명령어는 일단락 된거죠.
4. 그 다음에 이제 프로그램 카운터가 하나 증가 했으니까 101 이 되었죠. 다음은 이제 101 번지에 있는 ADD 명령을 수행하게 되는 것이죠. 그래서 ADD 명령은 역시 ADD가 이 명령어를 가져오기 전에는 이게 무슨 명령어 인지 모르죠. 그래서 101 번지에 있는 명령을 이렇게 끌고 나와서 보니까. 어디로? Instruction Register 로. 여기서 해독을 여기서 디코딩을 해보니까. 아 이게 ADD 라는걸 이제 알게 된거죠. 1 단계에서.
5. 그럼 1 단계에서 가져오면, 또 이 프로그램 카운터는 하나 증가가 되겠죠. 이 값이. 그래서 그 다음 단계로 넘어가면, 프로그램 카운터는 또 자동으로 증가하고 해서 102 가 되고, 그 다음에 앞전에 이 IR 에 들어있는 이 명령어를 보니까 ADD 다. 그럼 ADD 가 뭔가? 아, 251 번지에 있는 내용을 가져와서. 기존에 LOAD 에서 250 번지에 있던 값. 그걸 가져와서 Accumulator 에 두었으니까, 그럼 251 번지 하고 250 번지에 있는 내용하고 서로 더하면 되겠다. 그렇게 되는것이죠.
6. 그래서 이 ALU 이걸 플러스 하는. 이 ALU 를 통해서 이 Accumulator 값은 기존의 4 값과 그 다음에 현재 가져온 3 값. 이거를 더해서 7을 만들어 놓은 것입니다. 그래서 여기 설명이 이렇게 붙어있죠. 이 의미는. 그것이 이제 ADD 명령이 끝나는거 입니다. 그래서 여기까지 현재 끝난 것이고.
7. 그 다음에 이제 STORE 명령은 어떻게 또 하나 보시죠. STORE 명령도 그 다음 프로그램 카운터가 증가해서 102 번지이니까 얘를 먼지 모르고 그냥 가져오는 겁

니다. 이게. 그럼 이때 까지도 이게 무슨 명령인지 모르는거죠. 여기까지 이제 1 단계에서 인출이 되는 것이고, 그 다음 단계로 이제 넘어가면, 프로그램 카운터가 하나 증가해서 얘는 이미 103 번지로 가있고. 그 다음에 이 IR 에 있는 명령을 해독하니까, 그때서야 이제 STORE 라는 걸 안거죠.

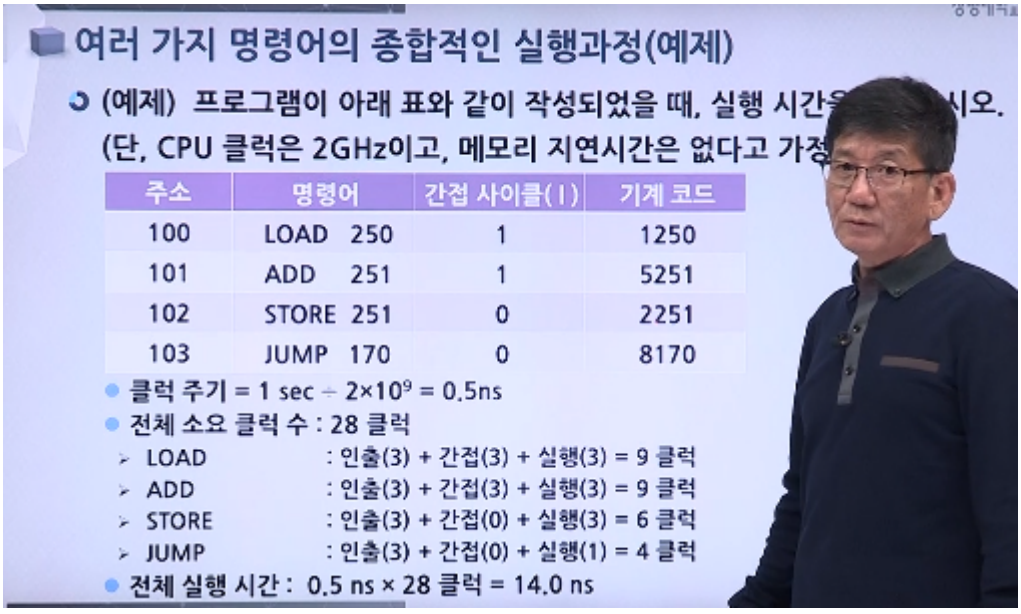
8. 그럼 STORE 란 뭐냐. 보니까. 251 번지에다가 저장해라. 이런 뜻이 된거죠. 그래서 251 번지에다가 뭐를. Accumulator 에 있는 내용을. 이렇게 되는 거죠. 그래서 Accumulator 에 우리가 계산된 7 이라는 값을 251 번지. 여기다가 저장을 하는. 이 과정이 바로 이제 STORE 과정인거죠. 이렇게 하므로써 이제 또 3 번째인 STORE 명령이 끝이 난겁니다.



▼ 설명

1. 다음은 이제 일반 명령어하고. 즉, 산술,연산 뭐 이런 명령어하고 다르게 이건 프로그램 제어 명령어입니다. 그래서 JUMP 는 어떻게 되나 살펴보도록 하죠.
2. 그래서 이미 앞 전에서 프로그램 카운터가 하나 증가해서 103 번지를 유지하고, 여기있는 그 Instruction 을. Instruction Register 로 이제 가져오게 되는데 그래서 이 명령어. 즉 이게 여기에 와있게 되는데, 이때까지도 이게 JUMP 인지 모르죠. 이제 여기서 해독하면서, 바로 JUMP 다 라는 걸 알 수 있는 것이죠.
3. 그리고 또 프로그램 카운터. JUMP 명령이니까 프로그램 카운터는 아까 103 에서 104 로 이미 증가가 되있었는데 JUMP 를 보니까, 이 명령어에서 170 번지로 JUMP 해라 이런 걸 이제 알게 된거죠. 그럼 170 번지에 있는 걸 어떻게 꺼내올 건가. 그냥 바로 지정하는 방법이 없는거죠.

4. 그래서 기존에 있던 방법을 사용하면, 아 170 번지에 있던 것을 꺼내 올려면 프로그램 카운터를 바꿔주면 되겠다. 이제 이런 생각이 드는 것입니다. 그래서 이 IR 에 있는 170 이란 값을 어디다? 프로그램 카운터에다가 줘서 그 값을 기존에 104 번지라고 되어 있던 그 값을 지워버리고 170 이라는 값을 여기다가 쓰게 되는 것이죠.
5. 그래서 이 JUMP 명령은 그 다음 실행이 없습니다. 끝난거죠 이게. JUMP 가. 그럼 이 다음에 또 다른 명령어들이 쭉 내려오면 그 다음부터 이제 170 번지 부터 그 명령어를 실행하게 되는 그런 형태가 되는 것입니다.



▼ 설명

1. 이제 마지막으로 이런 명령어들의 어떤 종합적인 실행을 한 번. 시간은 어떻게 걸리고 지금까지 했던 것들이. 계산을 한 번 해보도록 하면. 즉, 프로그램이 아래 표와 같이 작성되었을 때, 실행 시간을 계산하시오. 그래서 CPU 클럭은 역시 2GHz. 메모리 지연시간은 없다. 라고 가정을 하죠. 여기서. 그러면 앞에서. 표에서는 이 명령을 실행했는데 제가 여기다가 간접 사이클을 넣었습니다. 여기다가. 그러니까 실행 시간이 달라지게 되겠죠.
2. 그래서 LOAD 명령은 원래 인출 실행에서 6개의 클럭. ADD, STORE 다 같이 6개의 클럭씩 걸리지만, JUMP 는 인출하는데 3개 실행하는데 1 클럭. 그래서 4개. 즉, 이렇게 계산을 하면 되지만, 지금 여기 간접이라는 걸 제가 넣었기 때문에 간접 사이클 3개가 더 들어가게 되죠. 그래서 LOAD 는 인출, 간접 3개, 실행 3개 해서 9개의 클럭.

3. ADD 도 마찬가지로 인출, 간접, 실행. 그런데 STORE 는 간접이 없거든요. 그러니까 그냥 STORE 는 인출과 실행만, 간접은 0 이 되는 것이고. 그 다음에 JUMP 도 마찬가지로 간접이라는 개념이 없죠. 인출하고 실행하는데 1개. 그래서 이걸 모두 더하면 28 개의 클럭이 계산이 되서. 곱하면. 14 나노세크. 즉, 이 4개의 프로그램을 수행하는데 걸리는 시간이 14 나노세크 걸린다. 이렇게 얘기를 하는 것이죠.
4. 그래서 실질적으로 여러분들이 컴퓨터 프로그램을 하게 되면, 앞에서 했던 C 는 A 더하기 B 다. 이런 Statement 를 우리가 C 에서 작성했을 때, 과연 이것들이 얼마나 많은 명령어가 생기느냐. 이러한 명령어들을 계속 반복적으로 수행함으로써 그런 시간이 다 더해지고 하다보면 그것이 바로 CPU 가 실행하는데 걸리는 시간. 이런 것들을 계산할 수 있는 것 입니다.