3장 / 프로세서

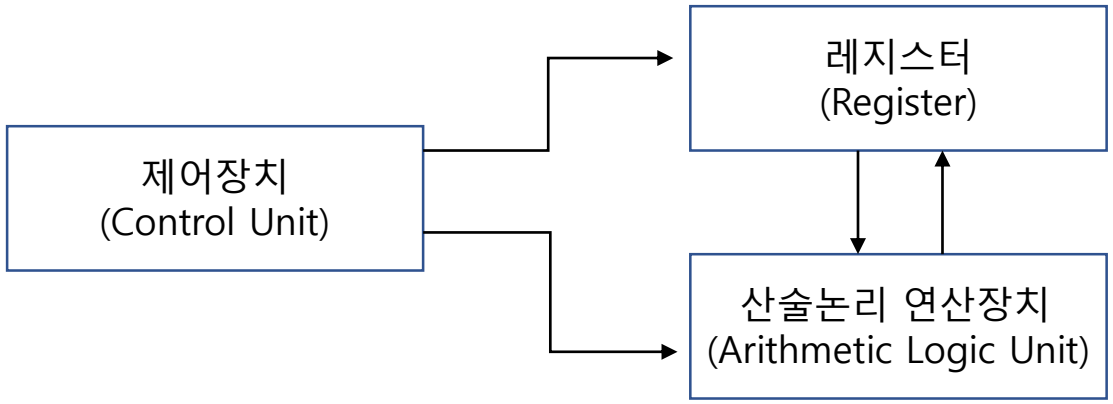
정보처리산업기사

SECTION 50

중앙처리장치

중앙 처리 장치

- 중앙처리장치는 사람의 두뇌와 같이 컴퓨터 시스템에 부착된 모든 장치의 동작을 제어하고 명령을 실행하는 장치이다.
- 중앙처리장치는 제어장치, 연산장치, 레지스터 그리고 이들을 연결하여 데이터를 전달하는 버스로 구성되어 있다.



https://www.youtube.com/watch?v=cNN_tTXABUA

제어 장치

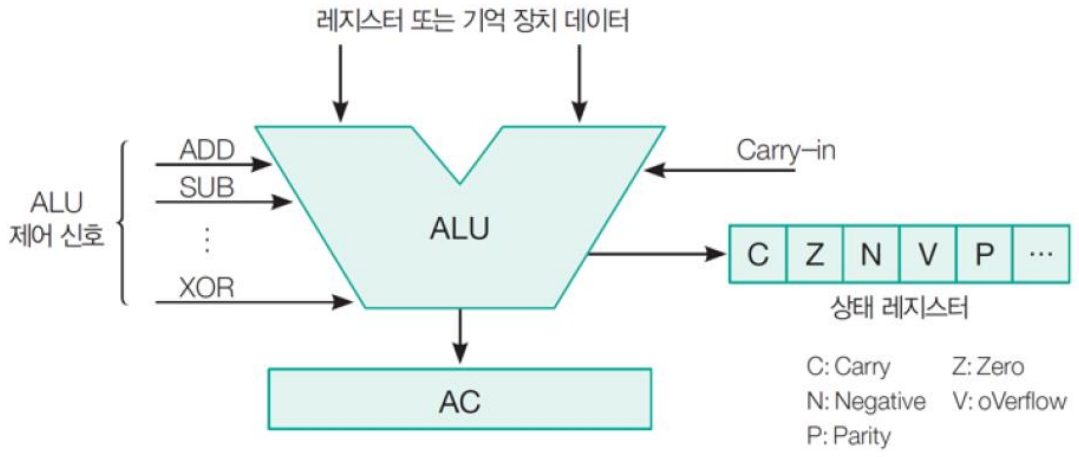
- 제어장치는 컴퓨터에 있는 모든 장치들의 동작을 지시하고 제어하는 장치이다.
- 제어장치는 명령 레지스터에서 읽어 들인 명령어를 해독하여 해당하는 장치에게 제어 신호를 보내 정확하게 수행하도록 지시한다.

제어장치의 구성 요소

- 명령 레지스터 : 현재 실행중인 명령어의 내용을 기억하고 있다.
- 명령 해독기(Decoder) : 명령 레지스터에 있는 명령어를 해독하는 회로이다.
- 제어신호 발생기, 부호기(Encoder) : 해독된 명령에 따라 각 장치로 보낼 제어 신호를 생성하는 회로이다.
- 제어 주소 레지스터(CAR) : 다음에 실행할 마이크로명령어의 주소를 저장하는 레지스터로, Mapping의 결과값, 주소 필드, 서브루틴 레지스터의 내용들이 적재되어 있다.
- 제어 버퍼 레지스터(CBR) : 제어기억장치로부터 읽혀진 마이크로명령어 비트들을 일시적으로 저장하는 레지스터이다.
- 제어 기억 장치 : 마이크로명령어들로 이루어진 마이크로 프로그램을 저장하는 내부 기억 장치이다.
- 순서 제어 모듈 : 마이크로명령어들의 실행 순서를 결정하는 회로들의 집합이다.
- 순차 카운터 : 디코더에 의해 선택된 번호에 해당하는 타이밍 신호를 생성한다.

연산 장치(ALU)

- 연산장치(ALU, Arithmetic & Logic Unit)는 제어장치의 명령에 따라 실제로 연산을 수행하는 장치이다.
- 연산장치가 수행하는 연산에는 산술 연산, 논리 연산, 관계 연산, 이동 등이 있다.
- 연산장치는 가산기, 누산기, 보수기, 데이터 레지스터, 오버플로 검출기, 시프트 레지스터 등으로 구성되어 있다.

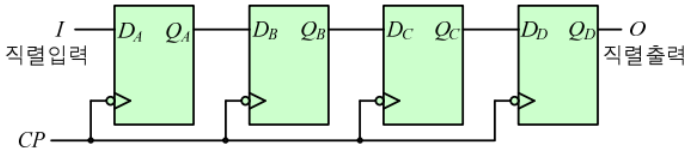


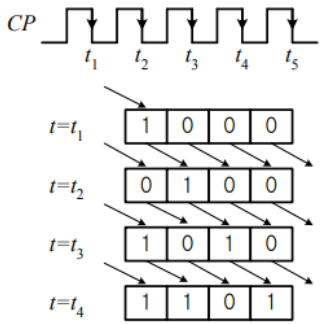
레지스터(register)

- 레지스터는 CPU 내부에서 처리할 명령어나 연산의 중간 결과값 등을 일시적으로 기억하는 임시 기억장소이다.
- 레지스터는 플립플롭이나 래치(Latch)들을 병렬로 연결하여 구성한다,
- 레지스터는 메모리 중에서 속도가 가장 빠르다.
- 레지스터의 크기는 워드를 구성하는 비트 개수만큼의 플립플롭으로 구성되며, 여러 개의 플립플롭은 공통 클록의 입력에 의해 동시에 여러 비트의 자료가 저장 된다.
- 레지스터를 구성하는 플립플롭은 저장하는 값을 임의로 설정하기 위해 별도의 입력 단자를 추가할 수 있으며, 저장값을 0으로 하는 것을 설정해제(CLR)라 한다.

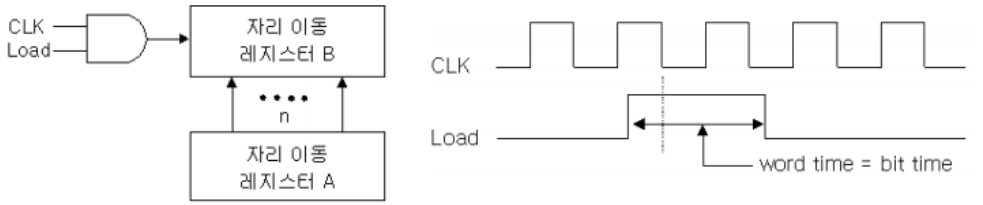
레지스터 간의 자료 전송

- 직렬 전송 : 직렬 시프트 마이크로 오퍼레이션을 뜻하며, 병렬 전송에 비해 전송 속도가 느리다.
- 병렬 전송 : 하나의 클록 펄스 동안에 레지스터 내의 모든 비트, 즉 워드가 동시에 전송되는 전송 방식이다.
- 버스 전송 : 모든 레지스터들이 공통으로 이용하는 경로로, 병렬 전송에 비해 결선의 수를 줄일 수 있다는 장점이 있다.

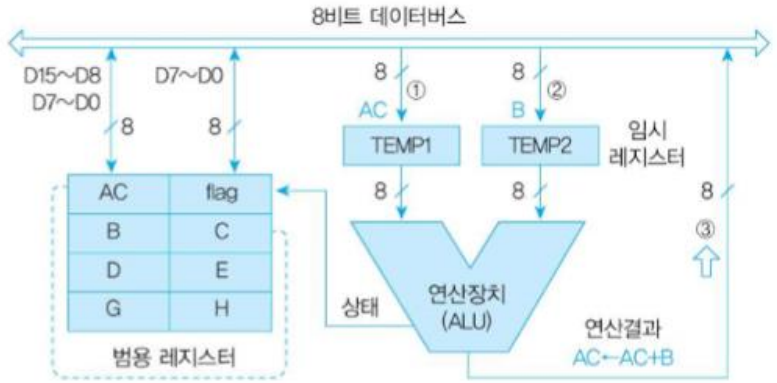




직렬전송



병렬전송



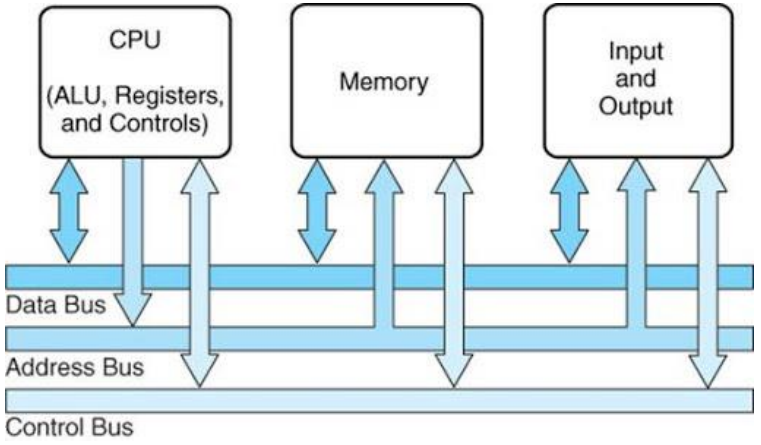
버스전송

레지스터의 종류 및 기능에 대한 설명 (1/3)

레지스터	기능
프로그램 카운터, 프로그램 계수기 (PC : Program Counter)	-다음 번에 실행할 명령어의 번지를 기억하는 레지스터 -분기 명령이 실행되는 경우 그 목적지 주소로 갱신됨.
명령 레지스터 (IR : Instruction Register)	현재 실행중인 명령의 내용을 기억하는 레지스터
누산기 (AC : Accumulator)	연산된 결과를 일시적으로 저장하는 레지스터로 연산의 중심
-상태 레지스터 (Status Register) -PSWR (Program Status Word Register) -플래그 레지스터	-시스템 내부의 순간순간의 상태가 기록된 정보를 PSW라고 함 -오버플로, 언더플로, 자리올림, 계산상태 (0,-,+), 인터럽트 등의 PSW를 저장하고 있는 레지스터 -프로그램 제어와 밀접한 관계를 가짐
메모리 주소 레지스터 (MAR : Memory Address Register)	-기억장치를 출입하는 데이터의 번지를 기억하는 레지스터
메모리 버퍼 레지스터 (MBR : Memory Buffer Register)	-기억 장치를 출입하는 데이터가 잠시 기억되는 레지스터로 CPU가 데이터를 처리하기 위해서는 반드시 거쳐야 함
베이스 레지스터 (Base Register)	명령이 시작되는 시작 번지를 기억하고 있는 레지스터
인덱스 레지스터 (Index Register)	-주소의 변경, 서브루틴 연결 및 프로그램에서의 반복 연산의 횟수를 세는 레지스터 -프로그래머가 내용을 변경할 수 있음
데이터 레지스터 (Data Register)	연산에 사용될 데이터를 기억하는 레지스터
시프트 레지스터 (Shift Register)	-저장된 값을 왼쪽 또는 오른쪽으로 1Bit씩 자리를 이동시키는 레지스터 -2배 길이 레지스터 라고도 함
메이저 스테이터스 레지스터 (Major Status Register)	CPU의 메이저 상태를 저장하고 있는 레지스터

버스

- 버스는 CPU, 메모리, I/O 장치 등과 상호 필요한 정보를 교환하기 위해 연결하는 공통의 전송선이다.
- 컴퓨터 내부 회로에서 버스 선을 사용하는 목적은 결선의 수를 줄이기 위해서이다.
- 메모리나 입출력 장치가 제대로 동작하려면 버스를 통해 전달되는 제어 신호, 어드레스 신호 및 데이터 신호의 상호 시간적 관계가 잘 유지되어야 한다.



전송하는 정보에 따른 분류

- 전송하는 정보에 따른 분류

-번지 버스(Address Bus) : CPU가 메모리나 입출력기기의 번지를 지정할 때 사용하는 단방향 전송선

-자료 버스(Data Bus) : CPU와 메모리 또는 입출력기기 사이에서 데이터를 전송하는 양방향 전송선

-제어 버스(Control Bus) : CPU의 현재 상태나 상태 변경을 메모리 또는 입출력 장치에 알리는 제어신호를 전송하는 데 사용하는 양방향 전송선

- 버스 위치에 따른 분류

-내부 버스 : CPU 및 메모리 내에 구성된 BUS

-외부 버스 : 주변 입 . 출력장치에 구성된 BUS

기출문제

SECTION 51

명령어

(Instruction)

명령어의 구성

- 컴퓨터에서 실행되는 명령어는 크게 연산자가 표시되는 연산자(Operation Code)부와 연산의 수행에 필요한 자료의 정보가 표시되는 자료부(Operand)로 구성된다.

연산자(Operation Code)부	모드(Mode)부	자료(Operand)부

- 연산자부
  - 수행해야 할 동작에 맞는 연산자를 표시하며 흔히 OP-Code부 라고 한다.
  - 연산자부의 크기(비트 수)는 표현하라 수 있는 명령의 종류를 나타내는 것으로, nBit 일 때 최대 2^n개의 명령어를 사용할 수 있다.
- 모드부
  - 주소부의 유효 주소가 결정되는 방법을 지정한다.
  - 모드 비트가 0이면 직접, 1이면 간접 주소이다.
- 자료부
  - 실제 데이터에 대한 정보를 표시하는 부분으로 주소 필드라고도 한다.
  - 기억 장소의 주소, 레지스터 번호, 사용할 데이터 등을 표시한다.
  - 자료부의 크기는 메모리의 용량과 관계가 있다.

명령어 설계 시 고려사항

- 연산자의 종류 : 해당 컴퓨터 시스템에서 처리할 기능에 맞게 연산자의 종류를 결정한다.
- 명령어 형식 : 명령어의 길이, 오퍼랜드 필드의 개수와 길이 등을 결정한다.
- 주소지정방식 : 명령어가 사용할 자료의 위치를 표현하기 위한 방법을 결정한다.
- 데이터 구조 : 해당 컴퓨터 시스템의 워드 크기 등 데이터 구조에 맞게 명령어를 설계한다.
- 인스트럭션 세트의 효율성을 높이기 위하여 고려할 사항 : 기억 공간, 사용 빈도, 주소 지정 방식, 주기억장치 밴드폭 이용

연산자(Operation Code)의 기능

- 연산자의 기능에는 함수 연산, 자료 전달, 제어, 입출력 기능이 있다.

- 함수 연산 기능

- 함수 연산은 수치적인 산술 연산과 비수치적인 논리 연산이 있다.
- 산술 연산 : ADD, SUB, MUL, DIV, 산술 Shift 등
- 논리 연산 : NOT, AND, OR, XOR, 논리적 Shift, Rotate, Complement, Clear 등

- 자료 전달 기능

- 자료 전달 기능은 CPU와 기억장치 사이에서 정보를 교환하는 기능이다.
- Load : 기억 장치에 기억되어 있는 정보를 CPU로 꺼내오는 명령
- Store : CPU에 있는 정보를 기억장치에 기억시키는 명령
- Move : 레지스터 간에 자료를 전달하는 명령
- Push : 스택에 자료를 저장하는 명령
- Pop : 스택에서 자료를 꺼내오는 명령

연산자(Operation Code)의 기능

- 제어 기능

- 제어기능은 명령의 실행 순서를 변경시킬 때 사용하는 명령이다.
- 무조건 분기 명령 : GOTO, Jump 등
- 조건 분기 명령 : IF 조건, SPA, SNA, SZA 등
- CALL : 부 프로그램 호출
- Return : 부 프로그램에서 주 프로그램으로 복귀

- 입 출력 기능

- 입출력 기능은 CPU와 I/O 장치, 또는 메모리와 I/O 장치 사이에서 자료를 전달하는 기능이다.
- INPUT : 입출력 장치의 자료를 주 기억장치로 입력하는 명령이다.
- OUTPUT : 주 기억장치의 자료를 입출력장치로 출력하는 명령이다

기출문제

SECTION 52

연산

(Operation)

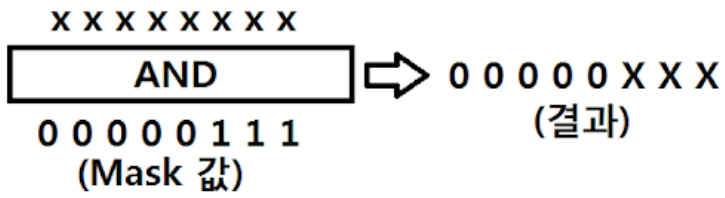
연산

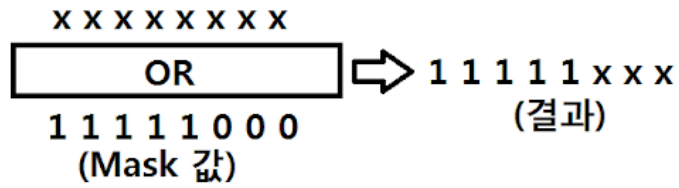
- AND(Masking Operation)

- AND 연산은 특정 문자 또는 특정 비트를 삭제 (Clear) 시키는 연산으로 , Masking 연산 이라고도 한다.
- AND 연산은 삭제할 부분의 비트를 0과 AND 시켜서 삭제하는데, 대응시키는 0인 비트를 Mask Bit라 한다.

- OR(Selective-Set)

- OR 연산은 특정 문자를 삽입하거나 특정 비트에 1을 세트 시키는 연산으로, Selective Set 연산 이라고도 한다.
- 삽입하거나 세트 시킬 비트에 삽입할 문자 코드 또는 1을 OR 연산 시킨다.

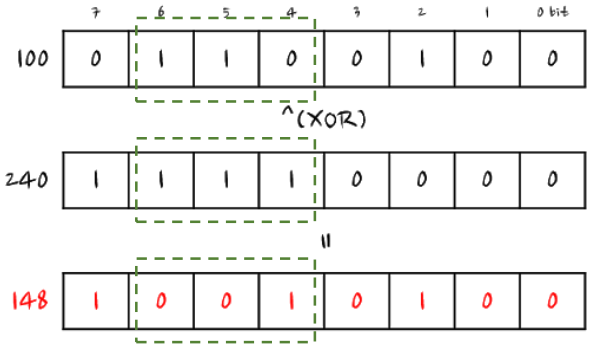


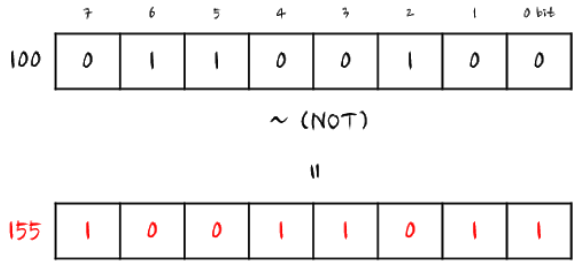


연산

- XOR 연산

- XOR 연산은 두 개의 데이터를 비교하거나 특정 비트를 발전시킬 때 사용한다.
- 두 개의 데이터를 XOR 연산하여 결과에 1Bit라도 1이 있으면 서로 다른 데이터이다.
- 반전시킬 때는 반전시킬 비트와 1을 XOR 시킨다.





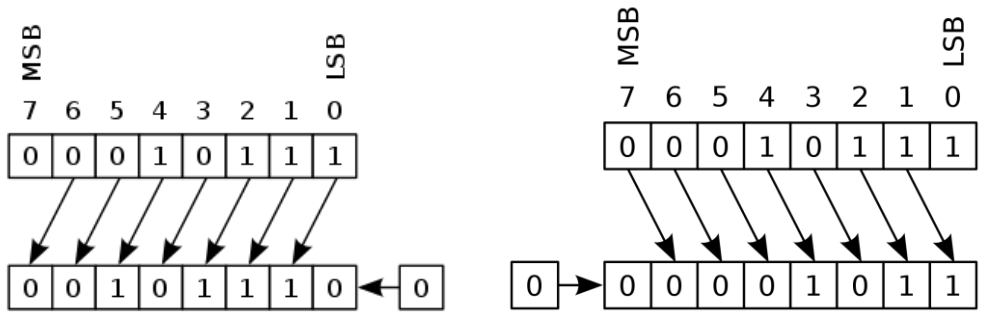
- NOT 연산

- NOT 연산은 각 비트의 값을 발전시키는 연산으로, 보수를 구할 때 사용한다.

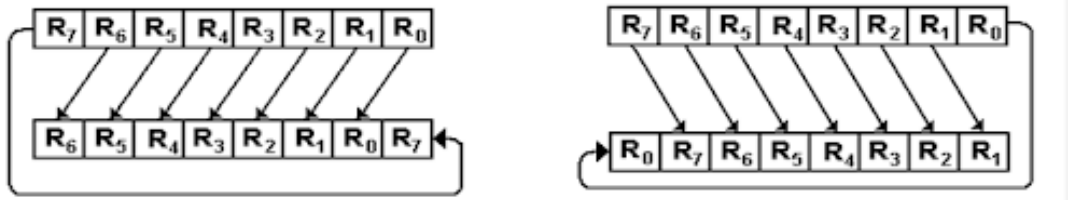
연산

- 논리 Shift

- 논리 Shift는 왼쪽 또는 오른쪽으로 1Bit씩 자리를 이동시키는 연산으로, 데이터의 직렬 전송(Serial Transfer)에 사용한다.
- 삽입되는 자리는 무조건 0이다.



논리 Shift



Rotate

- Rotate

- Rotate는 Shift에서 밀려 나가는 비트의 값을 반대편 값으로 입력하는 연산이다.
- 문자 위치를 변환할 때 사용한다.

산술 Shift (1/2)

- 산술 Shift는 부호를 고려하여 자리를 이동시키는 연산으로, 2^n으로 곱하거나 나눌 때 사용한다.
- 왼쪽으로 n Bit Shift 하면 원래 자료에 2^n을 곱한 값과 같다.
- 오른쪽으로 n Bit Shift 하면 원래 자료를 2^n으로 나눈 값과 같다.
- 홀수를 오른쪽으로 한 번 Shift하면 0.5의 오차가 발생한다.

Shift	수치 표현법	-43	+43
Shif Left	부호화 절대치	Padding Bit : 0 10101011 → 11010110 -43 x 2^1, 즉 -86이 된다.	양수는 모두 같다 Padding Bit : 0 00101011 → 01010110 -43 x 2^1, 즉 -86이 된다.
	1의 보수법	Padding Bit : 1 11010100 → 10101001 -43 x 2^1, 즉 -86이 된다.	
	2의 보수법	Padding Bit :0 11010101 → 10101010 -43 x 2^1, 즉 -86이 된다.	

산술 Shift (2/2)

Shift	수치 표현법	-43	+43
Shift Right	부호화 절대치	Padding Bit : 0 오차발생 : 0.5 증가 10101011 → 10010101 $-43 \div 2^1 \rightarrow 21.5 \rightarrow 1$	양수는 모두 같다 Padding Bit : 0 00101011 → 00010101 → 21 $-43 \div 2^1$, 즉 -21.5가 되어야 하지만 오차가 발생하여 0.5가 감소한다.
	1의 보수법	Padding Bit : 1 오차발생 : 0.5 증가 11010100 → 11101010 $-43 \div 2^1 \rightarrow 21.5 \rightarrow 1$	
	2의 보수법	Padding Bit : 0 오차발생 : 0.5 증가 11010101 → 11101010 $-43 \div 2^1 \rightarrow 21.5 \rightarrow 1$	

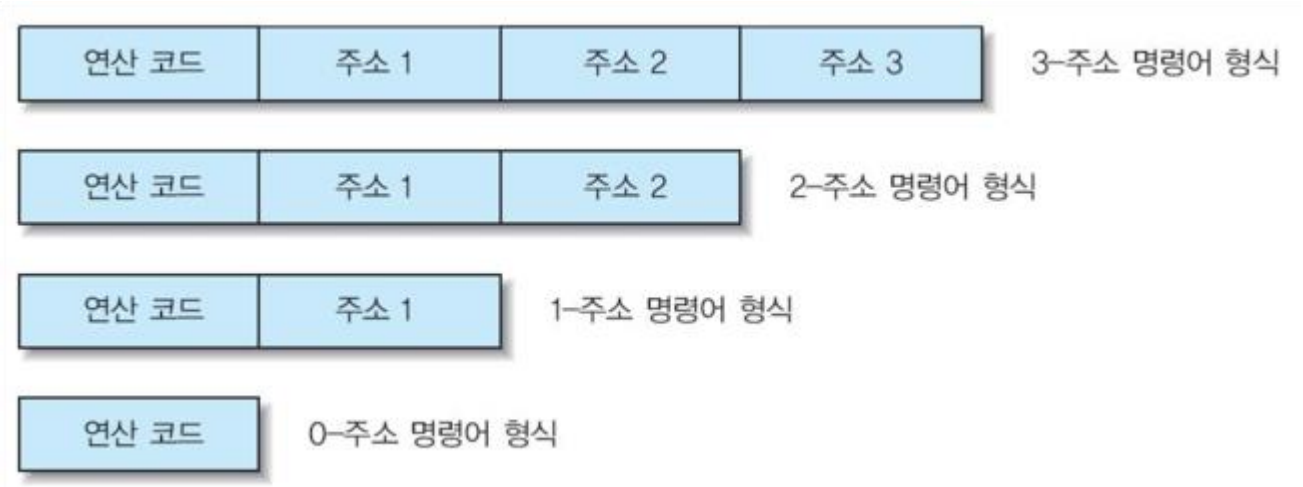
기출문제

SECTION 53

명령어 형식

명령어 형식

- 명령어는 크게 OP–Code부(명령부)와 Operand부(자료부)로 구성되는데, Operand부의 개수에 따라 다음의 네 가지 명령어 형식이 있다.



3주소 명령어

- 3주소 명령어는 Operand부가 세 개로 구성되는 명령어 형식으로 여러 개의 범용 레지스터(GPR)를 가진 컴퓨터에서 사용한다.
- 연산의 결과는 주로 Operand 1에 기록된다.

OP-Code	Operand 1	Operand 2	Operand 3

- 장점

- 연산 시 원래의 자료를 파괴하지 않는다.
- 다른 형식의 명령어를 이용하는 것보다 프로그램 전체의 길이를 짧게 할 수 있다.
- 전체 프로그램 실행 시 명령 인출을 위하여 주기억장치를 접근하는 횟수가 줄어 든다.

- 단점

- 명령어 한 개의 길이가 너무 길어진다.
- 하나의 명령을 수행하기 위해서 최소한 4번 기억 장소에 접근해야 하므로 수행 시간이 길어진다.

2주소 명령어

- 2주소 명령어는 Operand부 두 개로 구성되는, 가장 일반적으로 사용되는 명령어 형식이다.
- 여러 개의 범용 레지스터를 가진 컴퓨터에서 사용한다.

OP-Code	Operand 1	Operand 2

- 장점

- 실행 속도가 빠르고 기억 장소를 많이 차지하지 않는다.
- 3주소 명령에 비해 명령어의 길이가 짧다.
- 계산 결과가 기억장치에 기억되고 중앙처리장치에도 남아 있어서 계산 결과를 시험할 필요가 있을 때 시간이 절약된다.

- 단점

- 연산의 결과는 주로 Operand 1에 저장되므로 Operand 1에 있던 원래의 자료가 파괴된다.
- 전체 프로그램의 길이가 길어진다

2주소 명령어

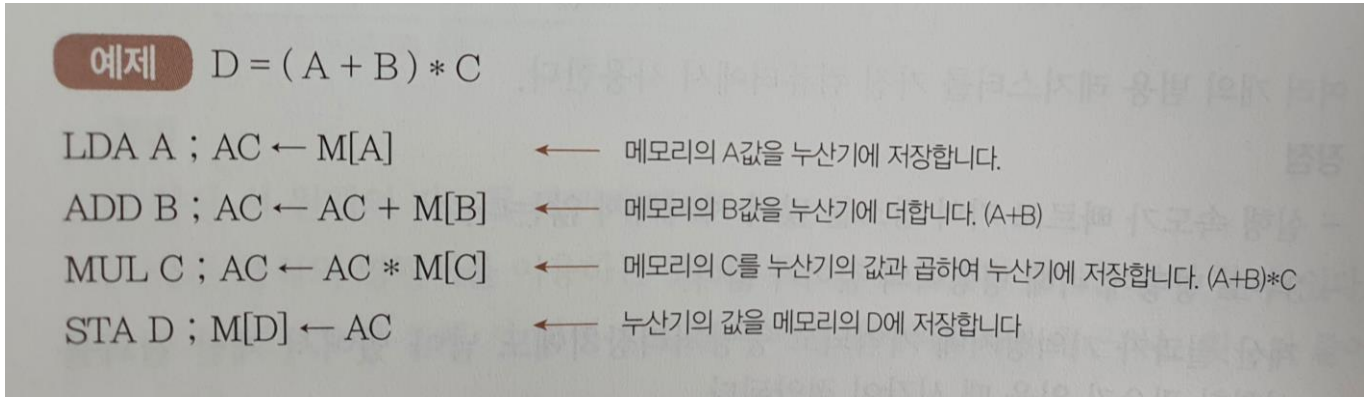
예제 Y = (A + B) * (C + D)

```
MOV  R1 A;   R1←M[A]        ←  메모리의 A 값을 R1에 저장합니다.
ADD  R1 B;   R1←R1+M[B]     ←  메모리의 B 값을 R1에 더합니다. (A+B)
MOV  R2 C;   R2←M[C]        ←  메모리의 C 값을 R2에 저장합니다.
ADD  R2 D;   R2←R2+M[D]     ←  메모리의 D 값을 R2에 더합니다. (C+D)
MUL  R1 R2;  R1←R1*R2       ←  R1과 R2를 곱하여 R1에 저장합니다. (A+B)*(C+D)
MOV  Y  R1;  M[Y]←R1        ←  R1의 값을 메모리의 Y에 저장합니다.
```

1주소 명령어

- 1주소 명령어는 Operand부가 한 개로 구성되어 있다.
- 1주소 명령어 형식의 컴퓨터는 누산기(AC, Accumulator)를 이용하여 명령어를 처리하므로 결과도 누산기에 저장된다.

OP-Code	Operand 1



0주소 명령어

- 0주소 명령어는 자료의 주소를 지정하는 Operand부 없이 OP-Code부만으로 구성되어 있다.

OP-Code

- 0주소 명령어의 모든 연산은 스택에 있는 자료를 이용하여 수행하기 때문에 스택 머신 이라고도 한다.
- 스택에 기억된 데이터만을 이용하여 연산하므로 인스트럭션 수행시간이 짧다.
- 피 연산자를 나타내지 않기 때문에 인스트럭션의 길이가 짧아서 기억 공간의 이용이 효율적이다.
- 스택을 사용한 컴퓨터에서 수식을 계산하기 위해서는 우선 수식을 PostFix 형태로 변경해야 한다.
- 0주소 명령어는 주소의 사용 없이 스택에 연산자와 피 연산자를 넣었다 꺼내어 연산한 후 결과를 다시 스택에 넣으면서 연산하기 때문에 원래의 자료가 남지 않는다.

0주소 명령어

예제 D = (A + B) * C

PUSH A ← 메모리의 A 값을 스택의 최상위에 저장합니다.

PUSH B ← 메모리의 B 값을 스택의 최상위에 저장합니다.

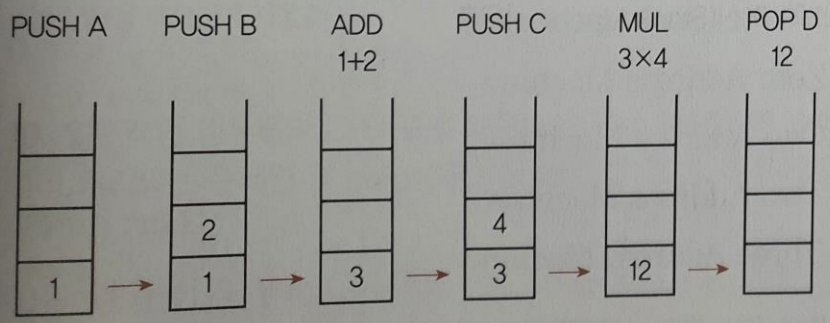
ADD ← 스택의 최상위에서 차례대로 2개의 피연산자 B와 A를 꺼내 덧셈을 수행한 다음 스택의 최상위에 저장합니다.

PUSH C ← 메모리의 C 값을 스택의 최상위에 저장합니다.

MUL ← 스택의 최상위에서 차례대로 2개의 피연산자 C와 (A+B)를 꺼내 곱셈을 수행한 다음 스택의 최상위에 저장합니다.

POP D ← 스택의 최상위에서 (A+B)*C를 꺼내 메모리의 D에 저장합니다.

※ A = 1, B = 2, C = 4 이라면 결과는 다음과 같다.



기출문제

SECTION 54

주소지정박식

(Addressing Mode)

주소 지정 방식 (Addressing Mode)

- 주소 지정 방식이란 프로그램이 수행되는 동안 사용될 데이터의 위치를 지정하는 방법이다.

- 주소 설계시 고려 사항

- 표현의 효율성 : 빠르게 접근하고 주소 지정에 적은 비트 수를 사용할 수 있도록 다양한 어드레스 모드를 사용할 수 있어야 한다.
- 사용의 편리성 : 다양하고 융통성 있는 프로그램 작업을 위해 포인터, 프로그램 리로케이션 등의 편의를 제공하여야 한다.
- 주소 공간과 기억 공간의 독립성 : 프로그램 상에서 사용한 주소를 변경 없이 실제 기억 공간 내의 주소로 재배치할 수 있도록 서로 독립적이어야 한다.

주소 지정 방식의 종류

- 주소 지정 방식은 암시적, 즉시적, 직접, 간접 그리고 계산에 의한 주소 지정 방식이 있다.

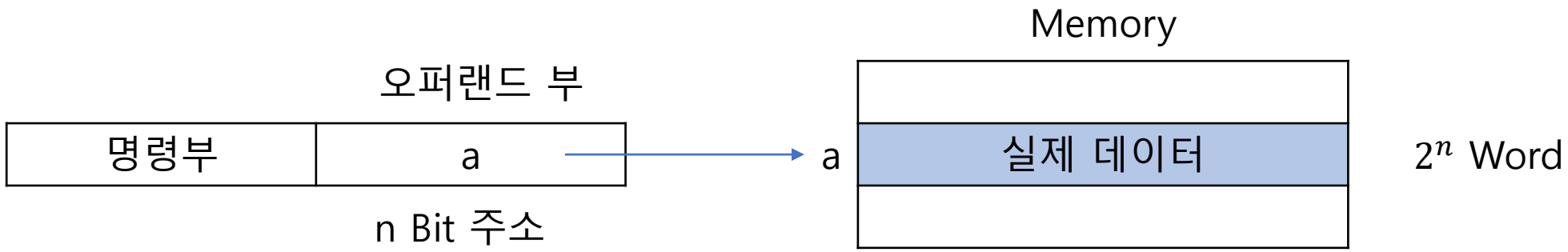
- 암시적(묵시적) 주소 지정 방식(Implied)
  - 명령 실행에 필요한 데이터의 위치를 지정하지 않고 누산기나 스택의 데이터를 묵시적으로 지정하여 사용한다.
  - 오퍼랜드가 없는 명령이나 PUSH R1처럼 오퍼랜드가 1개인 명령어 형식에 사용된다.

- 즉치적 주소지정방식(Immediate Mode)
  - 명령어 자체에 오퍼랜드(실제 데이터)를 가지고 있는 방식이다.

  - 별도의 기억 장소를 엑세스하지 않고 CPU에서 곧바로 자료를 이용할 수 있어서 실행 속도가 빠르다는 장점이 있다.
  - 명령어의 길이에 영향을 받으므로 표현할 수 있는 데이터 값의 범위가 제한적이다.

직접 주소 지정 방식(Direct Mode)

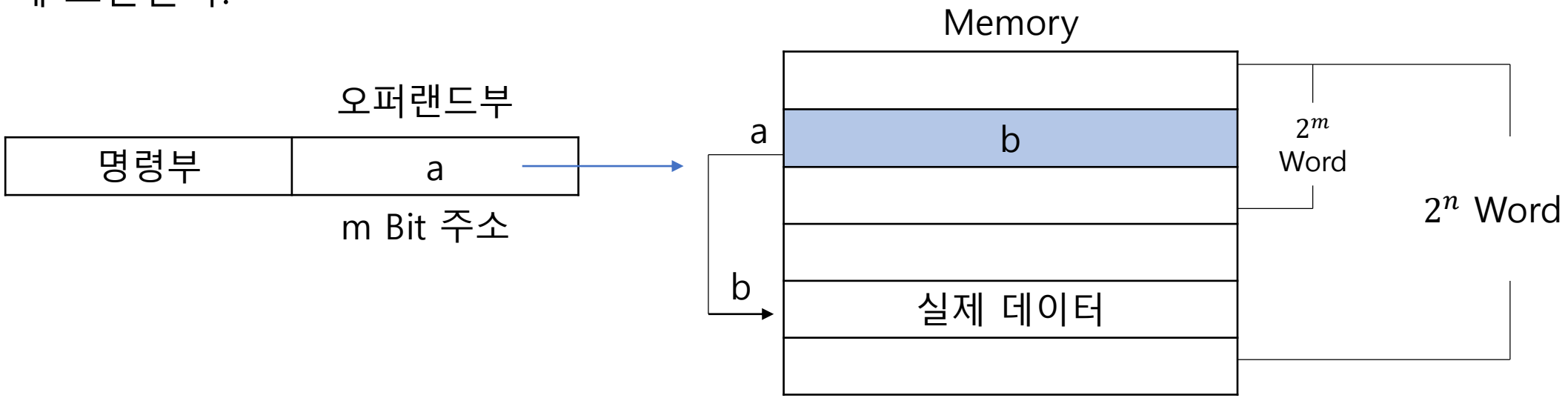
- 직접 주소지정방식은 명령의 주소부 (Operand)가 사용할 자료의 번지를 표현하고 있는 방식이다.



- 명령의 Operand부에 표현된 주소를 이용하여 실제 데이터가 기억된 기억 장소에 직접 사상시킬 수 있다.
- 주소 부분에 실제 사용할 데이터의 유효 주소를 적기 때문에 주소 길이에 제약을 받는다.
- 기억 용량이 2^n개의 Word인 메모리 시스템에서 주소를 표현하려면 nBit의 Operand부가 필요하다.
- 직접 주소 지정 방식에서 명령의 Operand부에 데이터를 가지고 있는 레지스터의 번호를 지정하면 레지스터 모드라고 한다.

간접 주소 지정 방식(Indirect Mode)

- 명령어 내의 Operand 부에 실제 데이터의 주소가 아니고, 실제 데이터의 주소가 저장된 곳의 주소를 표현하므로, 최소한 주 기억 장치를 두 번 이상 접근하여 데이터가 있는 기억 장소에 도달한다.



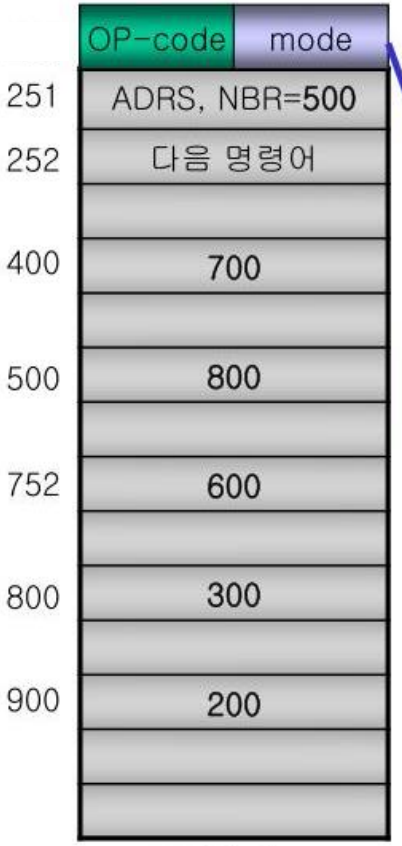
- 명령어에 나타낼 주소가 명령어 내에서 데이터를 지정하기 위해 할당된 비트수로 나타낼 수 없을 때 사용하는 방식이다.
- 명령의 길이가 짧고 제한되어 있어도 긴 주소에 접근 가능한 방식이다.
- 간접 주소 지정 방식에서 명령의 Operand부에 데이터의 주소를 가지고 있는 레지스터의 번호를 지정하면 레지스터 간접 모드라고 한다.

계산에 의한 주소 지정 방식

- 계산에 의한 주소 지정 방식은 Operand부와 CPU 특정 레지스터의 값이 더해져서 유효 주소를 계산하는 방식이다.
- 사용하는 레지스터의 종류에 따라 상대, 베이스, 인덱스 주소 지정 방식으로 구분한다.

주소 지정 방식	설명
상대 주소 (Relative Mode)	유효 주소 : 명령어의 주소 부분 + PC 명령어 자신의 기억 장소를 기준으로 하여 데이터의 우치를 지정하는 방식 이다.
베이스 레지스터 (Base Register Mode)	유효 주소 : 명령어의 주소 부분 + Base Register 프로그램을 재배치 할 때 이용한다.
인덱스 레지스터 (Index Register)	명령어의 주소 부분 + Index Register

주소 지정 방식의 요약



PC = 252

R1 = 400

AC

연산코드 : AC에 적재하라

주소지정	기호표기	레지스터 전송	주소	AC
즉치	LDA #NBR	AC ← NBR	251	500
직접	LDA ADRS	AC ← M[ADRS]	500	800
간접	LDA [ADRS]	AC ← M[M[ADRS]]	800	300
상대	LDA $ADRS	AC ← M[ADRS+PC]	752	600
인덱스	LDA ADRS(R1)	AC ← M[ADRS+R1]	900	200
레지스터	LDA R1	AC ← R1	-	400
레지스터간접	LDA (R1)	AC ← M[R1]	400	700

기출문제