Chapter 04

컴퓨터 구조

***학습목표**

- 컴퓨터 하드웨어의 전반적인 구성을 살펴본다.
- 중앙처리장치의 구성과 명령어 처리 단계를 살펴본다.
- 주기억장치의 역할과 동작 원리를 살펴본다.
- 컴퓨터 시스템에서 실행하는 프로그램 명령어의 형식과 동작 원리를 살펴본다.
- 프로그램이 어떻게 실행되는지를 예를 통해 살펴본다.

01. 컴퓨터 하드웨어의 구성

02. 프로그램 명령어

03. 프로그램 실행 동작

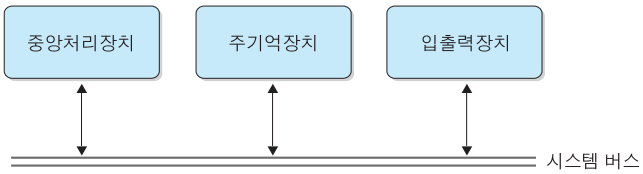
요약

연습문제

Section

1 컴퓨터 하드웨어의 구성

컴퓨터 하드웨어는 중앙처리장치(CPU : Central Processing Unit), 주기억장치, 입출력장치 그리고 이들을 연결해주는 시스템 버스로 구성된다.



[그림 4-1] 컴퓨터 하드웨어의 구성

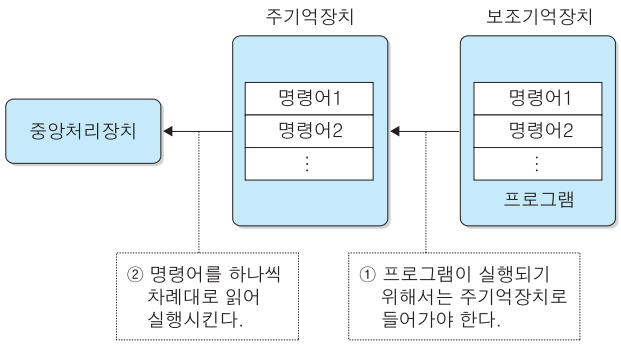
중앙처리장치는 프로그램 명령어를 실행하는 기능을 수행하고, 주기억장치는 실행 중인 프로그램과 프로그램 실행에 필요한 데이터를 일시적으로 저장하는 일을 담당한다. 그리고 입출력장치는 키보드, 모니터, 하드디스크 등으로 중앙처리장치나 주기억장치에 데이터를 입력하거나 출력하는 기능을 수행한다.

시스템 버스는 이런 장치들을 연결하는 일을 하는데, 주소 버스, 데이터 버스, 제어 버스로 나뉜다. 주소 버스를 통해서는 중앙처리장치가 주기억장치로부터 읽을 데이터가 저장된 주기억장치의 주소나 데이터를 쓸 주기억장치의 주소가 전달되고, 데이터 버스를 통해서는 중앙처리장치가 주기억장치로부터 읽거나 쓸 데이터가 전달된다. 그리고 제어 버스를 통해서는 읽을지 쓸지에 대한 제어 정보가 전달된다.

[표 4-1] 시스템 버스의 종류

종류	설명
주소 버스	주기억장치 주소가 전달된다.
데이터 버스	데이터가 전달된다.
제어 버스	제어 정보가 전달된다.

이런 구성에서 프로그램이 실행되는 과정은 [그림 4-2]와 같다. 평상시 대부분의 프로그램은 하드디스크와 같은 보조기억장치에 저장되어 있다가 프로그램이 실행되려 할 때 주기억장치로 들어가야 한다. 그리고 중앙처리장치에서 주기억장치에 들어온 프로그램의 명령어들을 하나씩 차례대로 읽어 실행시킨다. 자세한 실행 동작 과정은 앞으로 살펴볼 것이다.

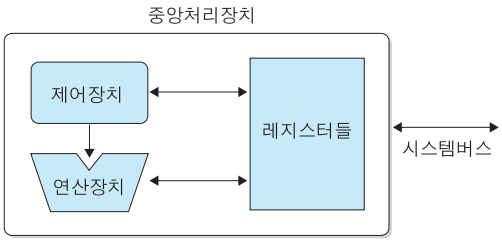


[그림 4-2] 프로그램의 실행 과정

그러면 중앙처리장치와 주기억장치에 대해 좀더 살펴보자.

1. 중앙처리장치

프로그램 명령어를 실행하는 일을 담당하는 중앙처리장치는 제어장치, 연산장치, 레지스터들의 세 부분으로 구성된다. 그리고 주기억장치를 비롯한 다른 장치들과는 시스템 버스로 연결되어 있다.



[그림 4-3] 중앙처리장치의 구성

[표 4-2] 중앙처리장치의 구성

구성	설명
제어장치	프로그램 명령어를 해석하고, 해석된 명령의 의미에 따라 연산장치, 주기억장치, 입출력장치 등에게 동작을 지시한다.
연산장치	덧셈, 뺄셈, 곱셈, 나눗셈의 산술 연산만이 아니라 AND, OR, NOT, XOR와 같은 논리 연산을 하는 장치로 제어장치의 지시에 따라 연산을 수행한다.
레지스터	주기억장치로부터 읽어온 명령어나 데이터를 저장하거나 연산된 결과를 저장하는 공간이다.

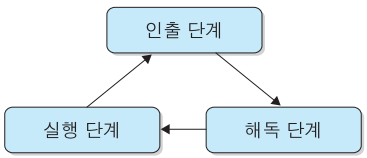
레지스터는 중앙처리장치에서 명령어를 실행하는 동안 필요한 정보들을 저장하는 기억 장소로, 범용 레지스터와 특수 목적 레지스터로 분류할 수 있다. 레지스터의 개수와 크기는 중앙처리장치의 종류에 따라 차이가 있으나, 이 책에서는 동작을 쉽게 설명하기 위해 [표 4-3]과 같은 최소한의 레지스터가 있다고 가정한다. 각 레지스터에 대한 상세한 설명은 뒤에서 다룰 것이다.

[표 4-3] 레지스터 종류

종류		설명
범용 레지스터		명령어 실행 중에 연산과 관련된 데이터를 저장한다.
특수 목적 레지스터	프로그램 카운터	다음에 실행될 명령어가 저장된 주기억장치의 주소를 저장한다.
	명령어 레지스터	현재 실행 중인 명령어를 저장한다.
	스택 포인터	주기억장치 스택의 데이터 삽입과 삭제가 이루어지는 주소를 저장한다.

중앙처리장치는 세 단계로 구성된 일련의 동작을 반복함으로써 명령어를 실행해 나가는데, 세 단계란 인출(fetch), 해독(decode), 실행(execute) 단계다. 인출 단계는 주기억장치에 저장된 명령어 하나를 읽어 오는 단계고, 해독 단계는 읽어 온 명령어를 제어 정보로 해독하는 단계다. 그리고 실행 단계는 해독된 명령을 실행하는 단계다.

한 명령어의 실행이 끝나면 다음 명령어에 대한 인출 단계를 시작한다.

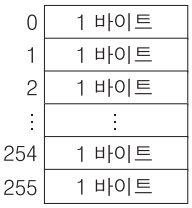


[그림 4-4] 중앙처리장치의 명령어 처리 단계

2. 주기억장치

주기억장치는 현재 실행 중에 있는 프로그램과 이 프로그램이 필요로 하는 데이터를 일시적으로 저장하는 장치다. 이런 주기억장치는 저장된 정보를 관리하기 편하고 각 위치를 구분하기 위해 바이트(byte) 또는 워드(word) 단위로 분할해 주소(address)를 할당한다. 이는 우리가 집을 구분하기 위해 주소를 사용하는 것과 유사한 개념이다.

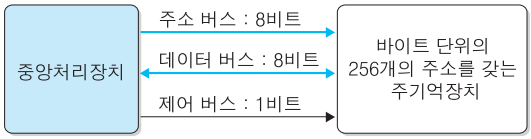
주기억장치는 바이트 단위로 나뉘며, [그림 4-5]와 같이 256개의 주소를 갖는다. 이 주기억장치에는 256바이트의 정보를 저장할 수 있으며, 주소는 0부터 시작된다.



[그림 4-5] 바이트 단위로 분할한 256바이트 크기의 주기억장치

주기억장치가 제공하는 동작은 중앙처리장치가 주기억장치에 데이터를 저장하는 '쓰기'와 주기억장치에 저장된 데이터를 중앙처리장치로 읽는 '읽기' 두 가지다. '쓰기'와 '읽기' 동작이 어떻게 이루어지는지를 [그림 4-5]와 같은 바이트 단위의 256개의 주소를 갖는 주기억장치를 통해 살펴보자.

주기억장치는 다음과 같이 주소 버스, 데이터 버스, 제어 버스와 연결되어 있다.



[그림 4-6] 주기억장치와 연결된 버스

■ 주소 버스

주소 버스를 통해 주기억장치의 어느 위치에서 데이터를 읽을지 또는 어느 위치에 데이터를 쓸지가 정해진다. 주기억장치의 주소는 0~255 사이므로 이 모든 주소를 나타내기 위해서는 주소 버스가 8비트가 되어야 한다. 8비트보다 작으면 모든 주소를 표현할 수 없고, 8비트보다 크면 사용하지 않는 비트의 낭비가 발생된다. 전송되는 정보는 중앙처리장치에서 주기억장치로만 전송되므로 단방향성이다.

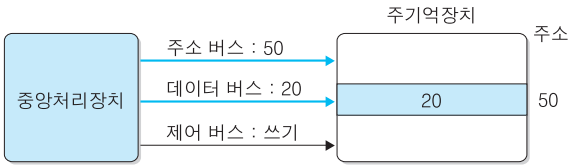
■ **데이터 버스**

데이터 버스를 통해 주기억장치로부터 읽거나 주기억장치에 써야 할 데이터가 전송된다. 데이터 버스의 크기는 중앙처리장치가 한 번에 전송할 수 있는 데이터의 크기와 같다. [그림 4-6]의 경우에는 8비트가 되지만, 중앙처리장치에 따라 데이터 버스의 크기는 달라진다. 데이터 버스를 통해 전송되는 데이터는 보낼 수도 받을 수도 있으므로 양방향성이다.

■ **제어 버스**

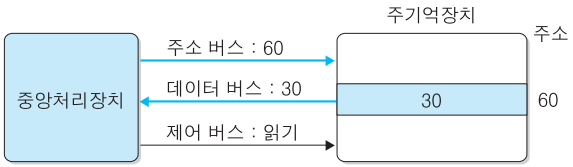
제어 버스를 통해 데이터를 주기억장치에 쓸지 읽을지를 결정하는 정보가 전송된다. 데이터를 주기억장치에 쓰려면 쓰기 제어 신호를, 주기억장치의 데이터를 읽으려면 읽기 제어 신호를 보낸다. 전송되는 정보는 중앙처리장치에서 주기억장치로만 전송되므로 단방향성이다.

그러면 주기억장치에 데이터를 쓰거나 읽는 동작 예를 살펴보자. 주기억장치 주소 50에 데이터 20을 쓰려면 주소 버스에 50을, 데이터 버스에 20을 전송한다. 그리고 제어 버스에 쓰기 제어 신호를 보내면 주기억장치 주소 50에 20이 저장된다.



[그림 4-7] 주기억장치 주소 50에 데이터 20 쓰기

그리고 주기억장치 주소 60의 데이터를 읽으려면 주소 버스에 60을 전송하고, 제어 버스에 읽기 제어 신호를 보낸다. 그러면 주기억장치 주소 60에 저장된 30이 데이터 버스를 통해 중앙처리장치로 전송된다.



[그림 4-8] 주기억장치 주소 60의 데이터 읽기

Section

02 프로그램 명령어

중앙처리장치는 프로그램을 구성하고 있는 명령어를 하나씩 읽어 실행한다고 앞에서 설명했다. 그러기 위해서는 중앙처리장치가 이러한 명령어들을 인식할 수 있어야 하며 이런 명령어를 실행하기 위한 회로를 갖추고 있어야 한다. 인식할 수 있는 명령어의 형식과 종류는 중앙처리장치에 따라 차이가 있다. 그러므로 프로그램은 해당 중앙처리장치가 인식할 수 있는 명령어들로 작성되어야 한다.

프로그램 명령어는 다음과 같이 연산 코드와 피연산자 부분으로 이루어진다.

[그림 4-9] 명령어 기본 형식

- 연산 코드 : 중앙처리장치가 실행해야 할 동작을 나타내는 부분으로, 더하기, 곱하기, AND, OR 등을 의미하는 내용이 위치한다.
- 피연산자 : 동작에 필요한 값 또는 저장 공간(레지스터 또는 주기억장치의 주소)을 나타내는데, 피연산자 개수는 중앙처리장치에 따라 차이가 있다.

여기에서는 피연산자의 개수가 다음과 같이 2개 있는 형식 위주로 살펴볼 것인데, 피연산자1과 피연산자2를 연산코드에 따라 연산한 결과를 피연산자1에 저장한다는 의미다.

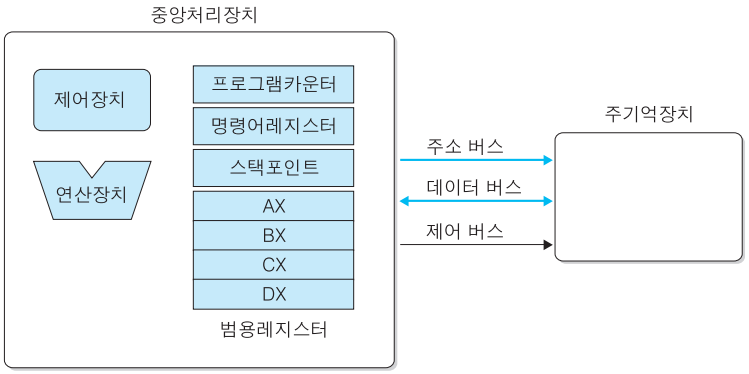
```
연산코드  피연산자1  피연산자2
```

대부분의 중앙처리장치가 제공하는 명령어는 크게 데이터 전송 명령어, 연산 명령어, 분기 명령어로 분류할 수 있다.

[표 4-4] 명령어의 분류

명령어의 분류	설명
데이터 전송 명령어	레지스터 또는 주기억장치의 데이터를 레지스터 또는 주기억장치로 이동하는 명령어와 입출력장치와 데이터를 주고받는 입출력 명령어가 있다.
연산 명령어	더하기, 빼기, 곱하기, 나누기 등의 산술 연산을 하는 명령어, AND, OR, NOT, XOR 등의 논리 연산 명령어, 레지스터에 저장된 비트들을 이동시키는 시프트 명령어가 있다.
분기 명령어	다음에 실행될 명령어를 새롭게 지정하는 명령어다.

명령어는 2진 표기법으로 표현되지만 이해를 돕기 위해 연상 기호 코드(mnemonic code)로 표현한다. 그리고 중앙처리장치의 구조가 다음과 같다는 가정을 하는데, 범용 레지스터는 4개로 각각의 이름은 AX, BX, CX, DX다.



[그림 4-10] 중앙처리장치의 구조

그러면 데이터 전송 명령어, 연산 명령어, 분기 명령어에 대해 좀더 상세하게 살펴보자.

1. 데이터 전송 명령어

데이터 전송 명령어는 다음과 같은 기능을 수행한다.

- 레지스터 또는 주기억장치에 저장된 값을 레지스터 또는 주기억장치로 전송
- 상수값을 레지스터 또는 주기억장치로 전송
- 스택에 저장된 값을 레지스터로 전송
- 레지스터에 저장된 값을 스택으로 전송

레지스터, 주기억장치, 상수값을 구분하기 위해 다음과 같이 가정한다.

- AX, BX, CX, DX는 레지스터를 의미한다.
- 30, 100과 같이 숫자로만 이루어진 값은 상수값을 의미한다.
- [10], [50]처럼 대괄호([,])로 감싸고 있는 것은 주기억장치를 의미하는데, [10]은 주기억장치의 주소가 10인 부분을 나타낸다.

다음 명령어는 B의 값을 A로 전송한다는 뜻으로, A는 주기억장치의 주소와 레지스터가 올 수 있고, B는 레지스터, 주기억장치의 주소, 상수값이 올 수 있다.

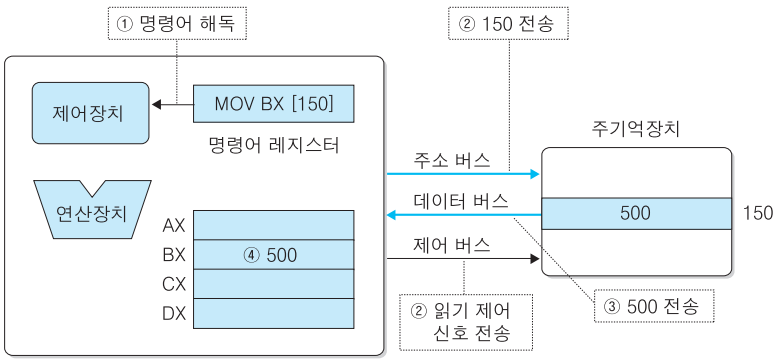
```
MOV A B
```

다음은 주기억장치 주소 150에 저장된 값을 읽어와 레지스터 BX에 저장한다는 뜻이다.

```
MOV BX [150]
```

주기억장치로부터 이 명령어가 읽혀져 명령어 레지스터에 저장되어 있는 상태의 동작 과정을 알아보자.

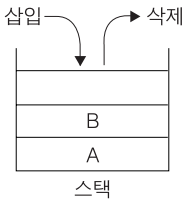
먼저 제어장치가 이 명령어를 해독한다. 그리고 주소 버스에 150을, 제어 버스에 읽기 제어 신호를 보낸다. 그러면 주기억장치 주소 150에 저장된 500을 데이터 버스를 통해 읽어와 레지스터 BX에 저장한다.



[그림 4-11] MOV BX [150]의 동작 과정

동작 과정은 다른 전송 명령어와 유사하지만, 스택을 이용한 전송 명령어는 그와는 약간 다르므로 하나씩 살펴보자.

주기억장치의 일부를 스택으로 사용하는데, 스택에서는 [그림 4-12]와 같이 데이터의 삽입과 삭제가 한쪽 방향에서만 일어난다. 그러므로 가장 최근에 삽입된 데이터가 가장 먼저 삭제되는데, 삽입과 삭제가 이루어지는 곳을 스택 포인터라는 레지스터가 가리키고 있다.

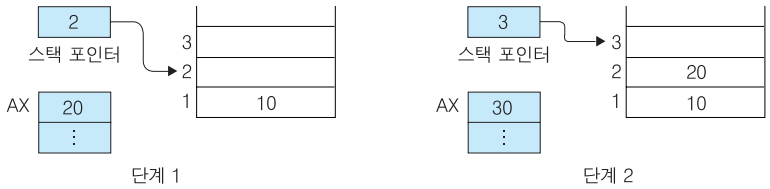


[그림 4-12] 스택의 동작

다음에 나오는 PUSH 명령어는 레지스터 AX에 저장된 값을 스택에 삽입한다.

```
PUSH AX
```

만약 스택 포인터에 2가 저장되어 있다면 스택 포인터가 가리키는 주소 2에 레지스터 AX에 저장된 값을 저장한다. 그리고 스택 포인터 값을 1 증가시켜 3을 가리킨다.

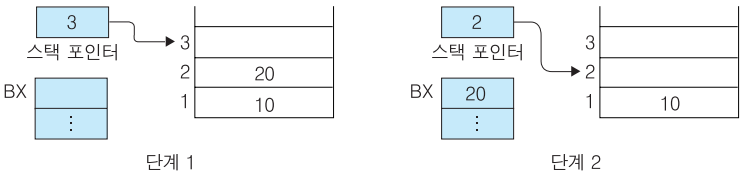


[그림 4-13] PUSH AX의 동작 과정

다음에 나오는 POP 명령어는 스택에서 데이터 하나를 삭제해 레지스터 BX에 저장한다.

```
POP BX
```

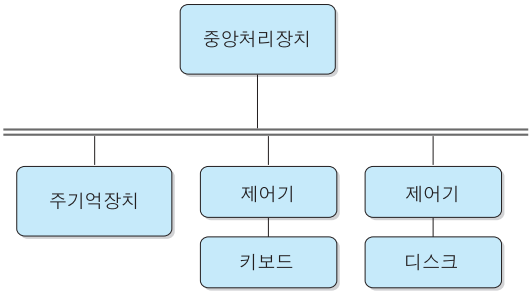
이 명령어는 스택 포인터 값을 1 감소시키고, 스택 포인터가 가리키는 주소 2의 데이터 20을 삭제해서 레지스터 BX에 저장한다.



[그림 4-14] POP BX의 동작 과정

다음은 입출력 명령어에 대해 살펴보자.

키보드, 디스크, 모니터 등의 입출력장치와 데이터를 주고받으려면 각 장치의 입출력을 제어하는 제어기(controller)를 통해야 하는 것 외에는 주기억장치의 읽기, 쓰기 동작과 거의 유사하다.



[그림 4-15] 입출력장치의 구성

각 제어기에는 포트(port)라는 유일한 번호가 부여되는데, 이 포트를 이용해서 입출력장치와 데이터를 주고받는다.

다음은 포트 B에서 데이터를 읽어 레지스터 AX에 저장하는 명령어다.

```
IN AX B
```

다음은 레지스터 AX의 데이터를 포트 B로 출력하는 명령어다.

```
OUT B AX
```

2. 연산 명령어

연산 명령어는 산술 연산 명령어, 논리 연산 명령어 그리고 시프트 명령어로 구분된다.

산술 연산 명령어

산술 연산 명령어의 종류로는 ADD, SUB, MUL, DIV가 있는데, 각각 더하기, 빼기, 곱하기, 나누기 연산을 의미한다.

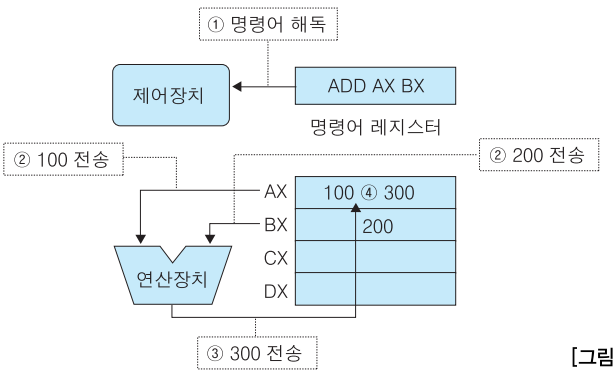
[표 4-5] 산술 연산 명령어

명령어의 분류	설명
ADD A B	A에 B를 더해서 A에 저장한다.
SUB A B	A에서 B를 빼서 A에 저장한다.
MUL A B	A에 B를 곱해서 A에 저장한다.
DIV A B	A를 B로 나누어서 A에 저장한다.

다음은 레지스터 AX와 BX에 저장된 값을 더해서 레지스터 AX에 저장하는 명령어다.

```
ADD AX BX
```

먼저 제어장치는 명령어 레지스터에 저장된 이 명령어를 해독하여 레지스터 AX에 저장된 100과 레지스터 BX에 저장된 200을 연산장치로 전송한다. 연산장치는 제어장치의 지시에 따라 이 두 값을 더하고, 그 결과값 300을 레지스터 AX에 저장한다.



[그림 4-16] ADD AX BX의 동작 과정

논리 연산 명령어

논리 연산 명령어의 종류로는 AND, OR, NOT, XOR 등이 있는데, 이들 연산은 대응되는 비트 간에 이루어진다.

[표 4-6] 논리 연산 명령어

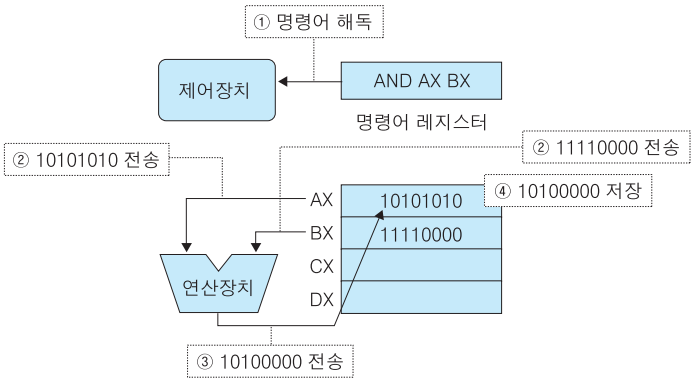
명령어	설명
AND A B	A와 B를 AND 연산해서 A에 저장한다.
OR A B	A와 B를 OR 연산해서 A에 저장한다.
NOT A	A의 1의 보수를 A에 저장한다.
XOR A B	A와 B를 XOR 연산해서 A에 저장한다.

AND, OR, NOT, XOR 연산이 어떻게 이루어지는지는 3장에서 살펴보았으므로 한 가지 예만 살펴보겠다.

다음은 레지스터 AX와 BX에 저장된 값을 AND 연산해서 결과를 AX에 저장하는 명령어다.

```
AND AX BX
```

먼저 제어장치는 명령어 레지스터에 저장된 이 명령어를 해독한다. 그리고 레지스터 AX에 저장된 10101010과 BX에 저장된 11110000을 연산장치로 전송한다. 연산장치는 제어장치의 지시에 따라 이 두 값을 AND 연산한 결과인 10100000을 레지스터 AX에 저장한다.



[그림 4-17] AND AX BX의 동작 과정

시프트 명령어

시프트 명령어는 레지스터에 저장된 데이터의 비트들을 이동시키는 데 사용된다. 논리 시프트인 SHR, SHL과 순환 시프트인 CIR, CIL이 있는데, 명령어의 가장 오른쪽의 R은 right를, L은 left를 의미한다.

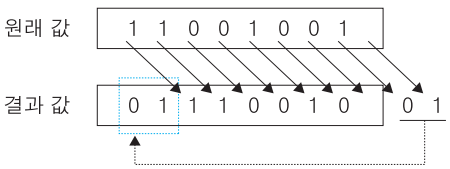
[표 4-7] 시프트 명령어

명령어		설명
논리시프트	SHR AX B	AX에 저장된 값을 오른쪽으로 B 비트만큼 이동시키고 왼쪽의 비는 곳에 0을 저장한다.
	SHL AX B	AX에 저장된 값을 왼쪽으로 B 비트만큼 이동시키고 오른쪽의 비는 곳에 0을 저장한다.
순환시프트	CIR AX B	AX에 저장된 값을 오른쪽으로 B 비트만큼 이동시키고 벗어나는 비트들을 왼쪽의 비는 곳에 저장한다.
	CIL AX B	AX에 저장된 값을 왼쪽으로 B 비트만큼 이동시키고 벗어나는 비트들을 오른쪽의 비는 곳에 저장한다.

동작이 거의 유사하므로 한 가지 예만 살펴보자. 다음은 레지스터 AX에 저장된 값을 오른쪽으로 2비트 순환 시프트하는 명령어다.

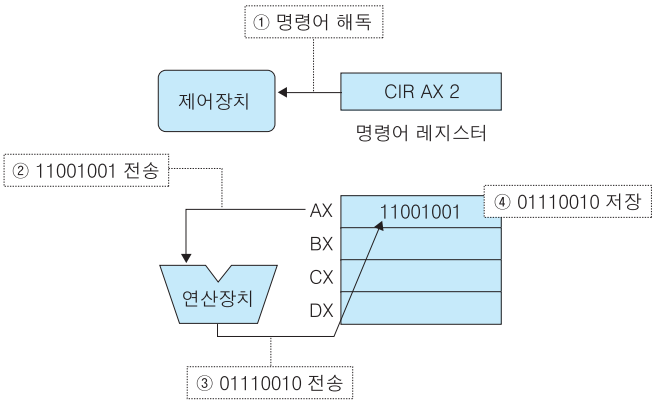
```
CIR AX 2
```

먼저 제어장치는 명령어 레지스터에 저장된 이 명령어를 해독하고 레지스터 AX에 저장된 11001001을 연산장치로 전송한다. 제어장치의 지시에 의해 연산장치는 이 값을 오른쪽으로 2비트 순환 시프트하는데, 동작은 다음과 같다.



[그림 4-18] 2비트 순환 시프트 동작

시프트 연산 결과인 01110010을 레지스터 AX에 저장한다.

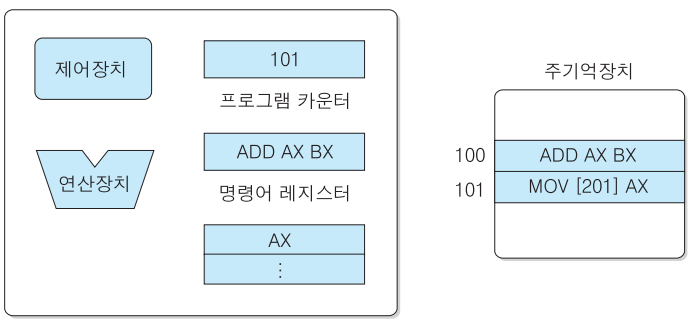


[그림 4-19] CIR AX 2의 동작 과정

3. 분기 명령어

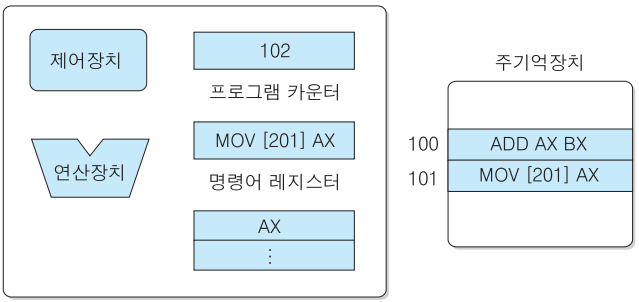
프로그램에서의 명령어들은 위에서부터 아래로 한 문장씩 순차적으로 실행된다. 그러나 JUMP나 CALL과 같은 분기 명령어를 이용하면 바로 아래에 위치한 명령어가 아닌 다른 위치의 명령어로 분기하게 된다. 분기 명령어에 대해 살펴보기 전에 먼저 프로그램 카운터라는 레지스터에 대해 알아보자.

[표 4-3]에서 프로그램 카운터란 다음에 실행될 명령어가 저장된 주기억장치 주소를 저장한다고 했다. 만약 중앙처리장치가 주기억장치 주소 100에 저장된 명령어인 ADD AX BX를 실행하고 있다면 프로그램 카운터에는 다음에 실행될 명령어가 저장된 주소인 101이 저장된다.



[그림 4-20] 프로그램 카운터에는 101이 저장

현재 명령어에 대한 실행을 마치면 프로그램 카운터에 저장된 주소 값인 101에 저장된 MOV [201] AX를 읽어와 명령어 레지스터에 저장한다. 현재 중앙처리장치는 주소 101에 저장된 명령어를 실행하는 중이므로 프로그램 카운터 값을 1 증가시켜 다음에 실행될 명령어가 저장된 주소인 102로 변경한다.

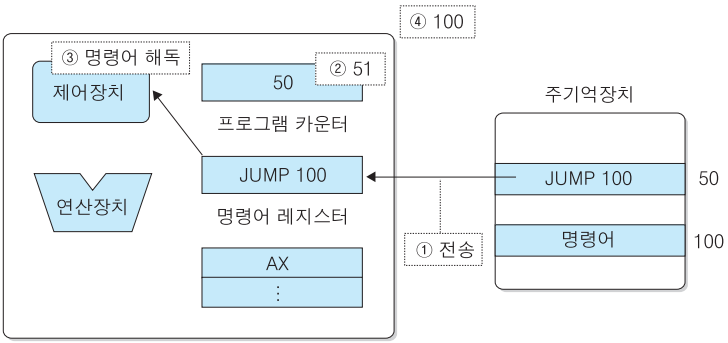


[그림 4-21] 프로그램 카운터 값을 1 증가시킴

프로그램 카운터에 대해 간단하게 알아보았으므로 분기 명령어를 살펴보자. 다음은 주기억장치 주소 100으로 분기하는 명령어다.

```
JUMP 100
```

주기억장치 주소 50으로부터 JUMP 100을 읽어 명령어 레지스터에 저장하면 프로그램 카운터 값은 1이 증가되어 51이 된다. 명령어 레지스터에 저장된 명령어인 JUMP 100을 해독하고 분기하고자 하는 주기억장치 주소 100을 프로그램 카운터에 저장하면 동작이 완료된다. 프로그램 카운터 값이 100이므로 다음에는 자연스럽게 주기억장치 주소 100에 저장된 명령어가 실행된다.

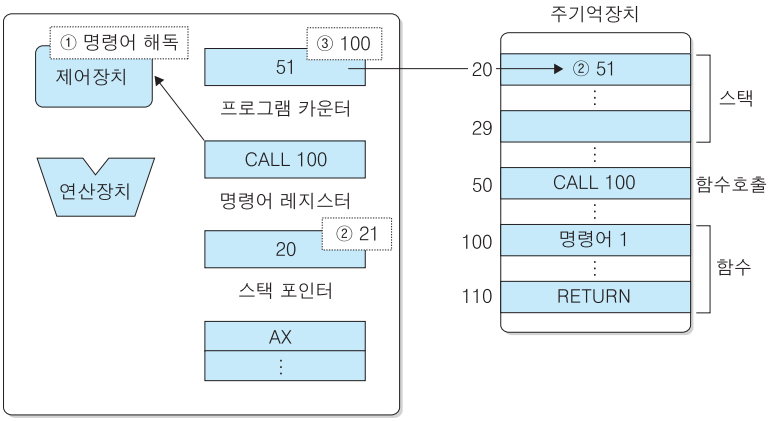


[그림 4-22] JUMP 100의 동작 과정

또 다른 분기 명령어인 CALL은 프로그램 카운터에 저장되어 있는 주소를 스택에 저장하고, 다른 위치로 분기하는 명령어로 함수를 호출할 때 사용한다. 다음은 현재 프로그램 카운터에 저장되어 있는 주소를 스택에 삽입하고, 주소 100으로 분기하는 명령어다.

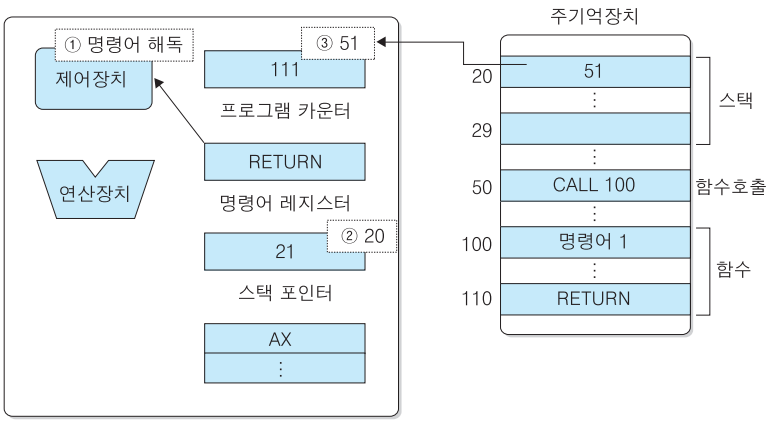
```
CALL 100
```

[그림 4-23]에서 주소 100~110에 함수가 저장되어 있고, 주소 20~29를 스택 영역이라 가정하고 동작을 살펴보자. 우선 제어장치가 명령어를 해독한다. 그리고 프로그램 카운터에 저장되어 있는 주소 51을 스택에 삽입하는데, 스택 포인터에 20이 저장되어 있으므로 주소 20에 저장된다. 그리고 스택 포인터는 1이 증가되어 21이 되고, CALL 100의 100을 프로그램 카운터에 저장하면 동작이 종료된다.



[그림 4-23] CALL 100의 동작 과정

그러면 함수의 첫 번째 명령어인 주소 100의 명령어부터 실행된다. 주소 100부터 109까지의 명령어들을 실행하고, 주소 110의 RETURN을 실행하게 되는데, RETURN은 함수의 끝을 의미하는 것으로, 함수 호출 명령인 CALL 100 다음 주소의 명령어가 실행되도록 한다. 그러므로 RETURN을 실행하면 스택 포인터 값을 1 감소시킨 주소 20에 저장된 51을 꺼내서 프로그램 카운터에 저장한다.



[그림 4-24] RETURN의 동작 과정

그러면 주소 51에 저장된 명령어를 실행하는데, 이 명령어는 함수 호출 명령어인 CALL 100 다음 주소의 명령어다.

Section

03 프로그램 실행 동작

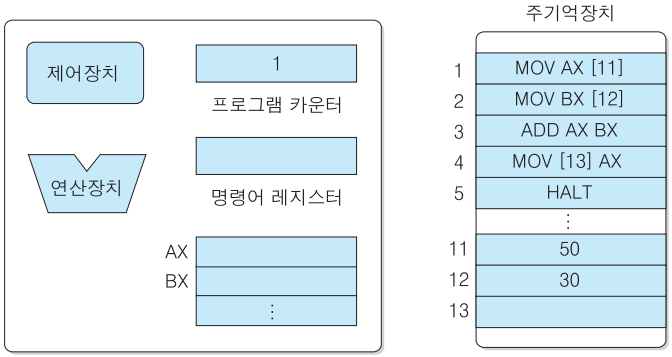
주기억장치로부터 두 개의 값을 읽어 와 레지스터에 저장하고, 이 값들을 더한 결과를 주기억장치에 저장하는 프로그램 실행 동작을 연상 기호 코드 형식과 2진 표기법 형식으로 살펴보자.

1 연상 기호 코드 형식

이런 동작을 하는 프로그램은 다음과 같이 나타낼 수 있다.

```
01 MOV AX [11] → 주기억장치 주소 11의 값을 레지스터 AX로 읽어옴
02 MOV BX [12] → 주기억장치 주소 12의 값을 레지스터 BX로 읽어옴
03 ADD AX BX   → 레지스터 AX와 BX에 저장된 값을 더한 결과를 AX에 저장
04 MOV [13] AX → AX의 값을 주기억장치 주소 13에 저장
05 HALT        → 프로그램 종료
```

이 프로그램이 실행되기 위해서는 주기억장치로 들어가야 하는데, [그림 4-25]처럼 주소 1~5까지 연속적으로 들어 있고, 주소 11에는 50이, 주소 12에는 30이 저장되어 있고, 프로그램 카운터에는 1이 저장되어 있다고 가정하고 동작 과정을 살펴보자.



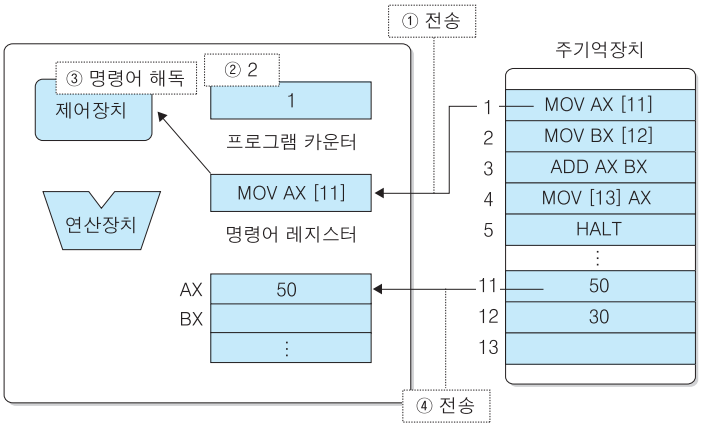
[그림 4-25] 시작 상태

① 주소 버스에 프로그램 카운터에 저장된 1을 보내고, 제어 버스에 읽기 제어 신호를 보내 주기억장치 주소 1의 MOV AX [11]을 읽어와 명령어 레지스터에 저장한다.

② 프로그램 카운터 값을 1 증가시켜 2가 된다. 다음에 실행될 명령어가 저장되어 있는 주기억장치의 주소는 2이기 때문이다.

③ 제어장치는 명령어 레지스터의 내용을 해독하여 주기억장치 주소 11로부터 데이터를 읽어 올 필요성을 느낀다.

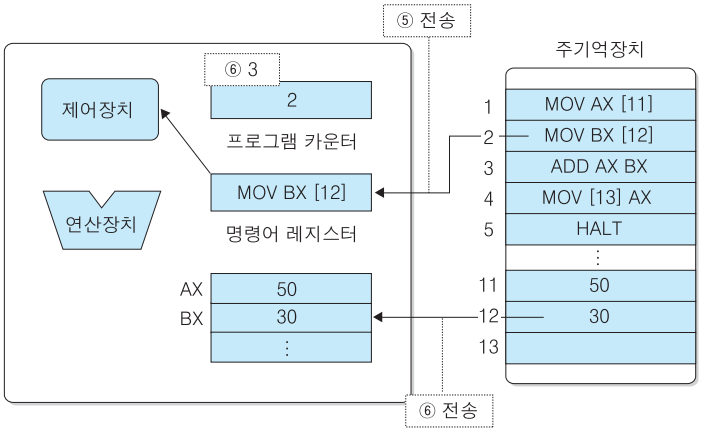
④ 주소 버스에 11을, 제어 버스에 읽기 제어 신호를 보내 주기억장치 주소 11에 저장된 50을 읽어와 레지스터 AX에 저장한다. 주소 1 명령어에 대한 실행이 모두 끝났다.



[그림 4-26] 동작 ①~④

⑤ 다음 명령어를 실행하기 위해 프로그램 카운터에 저장된 2를 주소 버스를 통해 보내고, 읽기 제어 신호를 보내 주소 2에 저장된 MOV BX [12]를 읽어와 명령어 레지스터에 저장한다.

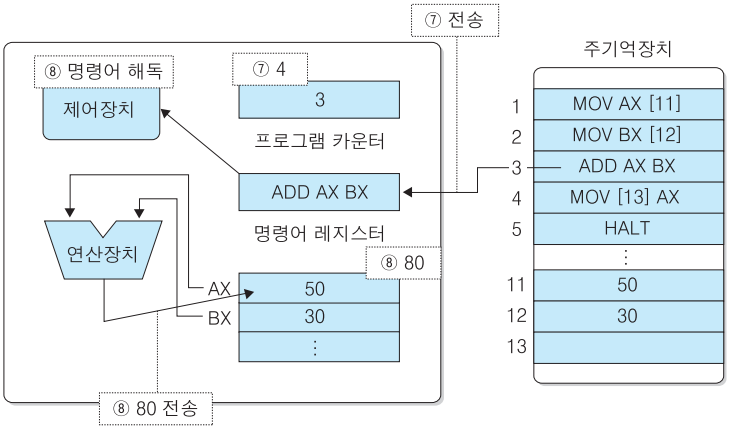
⑥ ②~④와 유사한 동작을 거쳐 프로그램 카운터는 3이 되고 레지스터 BX에는 30이 저장된다. 주소 2 명령어에 대한 실행이 끝났다.



[그림 4-27] 동작 ⑤~⑥

⑦ 다음 명령어인 주기억장치 주소 3에 저장된 ADD AX BX를 읽어와 명령어 레지스터에 저장한다. 그리고 프로그램 카운터 값은 1 증가하여 4가 된다.

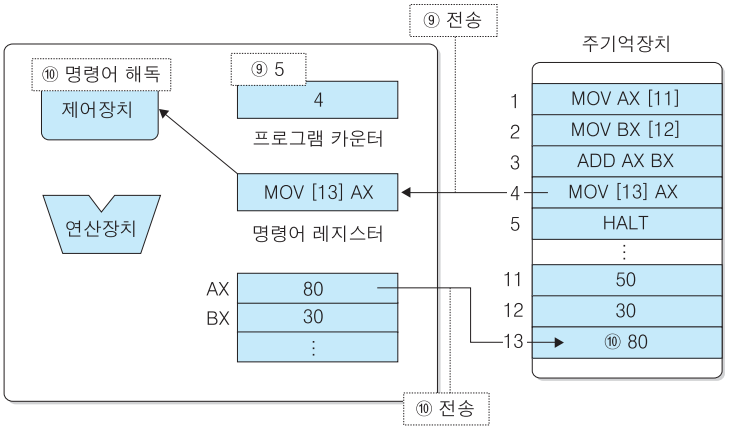
⑧ 제어장치는 명령어 레지스터에 저장된 명령어를 해독하여 레지스터 AX의 50과 레지스터 BX의 30을 연산장치로 전송하여 제어장치의 지시에 의해 연산장치는 이 두 값을 더한 결과인 80을 레지스터 AX에 저장한다. 주소 3 명령어에 대한 실행이 끝났다.



[그림 4-28] 동작 ⑦~⑧

⑨ 주소 4에 저장된 MOV [13] AX를 읽어와 명령어 레지스터에 저장하고, 프로그램 카운터 값은 1 증가하여 5가 된다.

⑩ 제어장치는 명령어를 해독하여 레지스터 AX에 저장된 80을 주기억장치 주소 13에 저장한다. 주소 4 명령어에 대한 실행이 끝났다.



[그림 4-29] 동작 ⑨~⑩

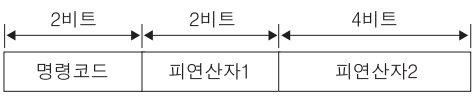
⑪ 다음 명령어인 주소 5에 저장된 HALT를 읽어와 명령어 레지스터에 저장하고, 프로그램 카운터 값은 1 증가하여 6이 된다.

⑫ 제어장치가 명령어를 해독하여 프로그램 실행을 종료한다.

2 2진 표기법 형식

2진 표기법 형식의 명령어를 살펴보기 위해 다음과 같은 가정을 한다.

- 주기억장치는 바이트 단위로 분할하며 0부터 15까지 16개의 주소를 갖는다.
- 범용 레지스터는 4개로 각 번호는 00, 01, 10, 11이다.
- 명령어의 길이는 8비트로 처음 2비트는 명령코드고, 다음 2비트는 피연산자1 부분, 나머지 4비트는 피연산자2 부분이다.



• 명령어로는 다음의 4가지가 있다.

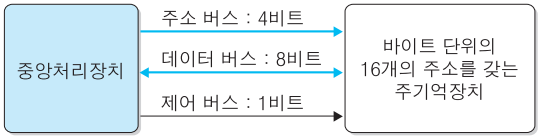
명령코드	피연산자1	피연산자2	설 명
00	범용 레지스터 번호	주기억장치 주소	주소가 피연산자2인 주기억장치 내용을 레지스터 피연산자1로 전송한다. 즉 00 01 1100 은 주기억장치 주소 12의 내용을 범용 레지스터 1로 전송하는 명령이다.
01	범용 레지스터 번호	주기억장치 주소	레지스터 피연산자1의 내용을 주소가 피연산자2인 주기억장치로 전송한다. 즉 01 10 1101 은 범용 레지스터 2의 내용을 주기억장치 주소 13으로 전송하는 명령이다.
10	범용 레지스터 번호	범용 레지스터 번호	레지스터 피연산자1의 값과 레지스터 피연산자2의 값을 더해 레지스터 피연산자1에 저장한다. 즉 10 01 0010 은 범용 레지스터 1의 값과 범용 레지스터 2의 값을 더해 결과를 범용 레지스터 1에 저장하는 명령이다.
11	00	0000	실행을 종료한다. 즉 11 00 0000 은 프로그램의 실행을 종료한다.

명령어가 4개로 이루어진 중앙처리장치는 거의 없다. 이해를 돕기 위해 최소한의 명령어를 이용한 것이다.

명령 코드 부분을 보면 2비트로 되어 있는데, 전체 명령어의 개수가 4개므로 모두 표현할 수 있다. 2비트로 표현할 수 있는 정보의 개수는 4개기 때문이다.

2비트 피연산자에 해당하는 부분에는 범용 레지스터 번호가 들어가는데, 범용 레지스터의 개수가 4개므로 모두 표현할 수 있다. 전송 명령어의 경우에 4비트 피연산자에 해당하는 부분에는 주기억장치 주소가 들어가는데, 주기억장치 주소가 0부터 15므로 4비트를 배정한 것이다.

앞에서 살펴봤던 예에서는 명령어의 길이를 언급하지 않았는데, 여기에서는 명령어의 길이를 8비트로 지정했다. 그러므로 데이터 버스가 8비트가 되어 주기억장치로부터 데이터를 읽거나 쓸 때 8비트 단위로 동작한다.



[그림 4-30] 8비트 크기의 데이터 버스

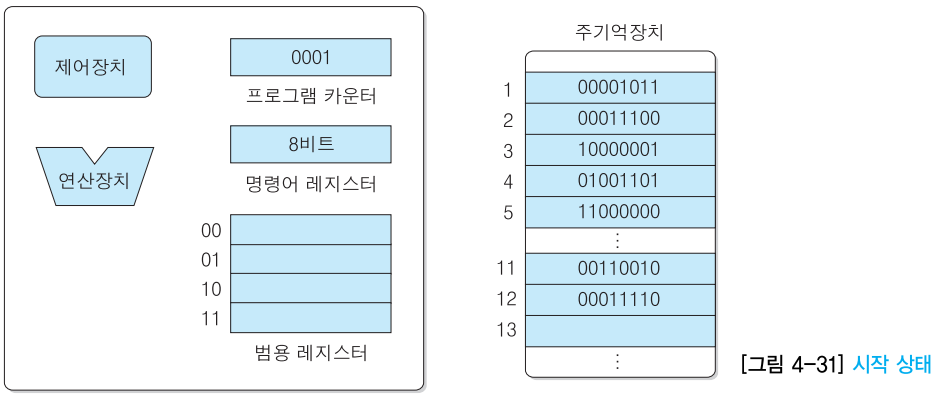
앞에서 살펴본 연상 기호 코드 형식의 프로그램을 2진 표기법으로 나타내면 다음과 같은데, 레지스터 AX를 0으로, BX를 1로 변경했다.

```
01 00001011
02 00011100
03 10000001
04 01001101
05 11000000
```

1행은 주기억장치 주소 11의 내용을 레지스터 0으로 읽어오는 명령어고, 2행은 주기억장치 주소 12의 내용을 레지스터 1로 읽어오는 명령어다. 3행은 레지스터 0과 1에 저장된 값을 더한 결과를 레지스터 0에 저장하는 명령어고, 4행은 레지스터 0의 내용을 주기억장치 주소 13에 저장하는 명령어다. 그리고 5행은 프로그램을 종료하는 명령어다.

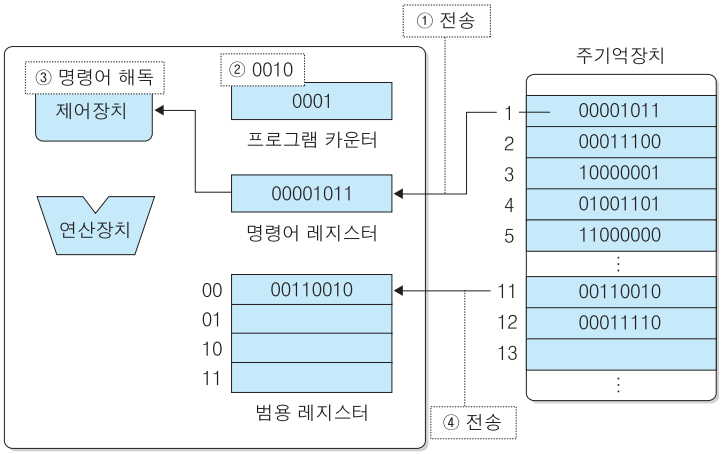
전반적인 실행 과정은 앞에서 살펴본 내용과 큰 차이가 없으므로 첫 번째 명령어를 실행하는 동작에 대해서만 살펴보자.

[그림 4-31]은 시작 상태인데, 프로그램 카운터에는 1이 저장되어 있다. 주기억장치 주소가 4비트 크기므로 프로그램 카운터의 크기는 4비트가 되고, 명령어의 길이가 8비트므로 명령어 레지스터의 크기는 8비트가 된다.



[그림 4-31] 시작 상태

주소 버스에 프로그램 카운터에 저장된 0001을 보내고 읽기 제어 신호를 보내면 주소 1에 저장된 내용이 데이터 버스를 통해 전송되어 명령어 레지스터에 저장된다. 그리고 프로그램 카운터가 1 증가하여 2가 된다. 그리고 제어장치가 명령어 레지스터에 저장된 명령어를 해독하여 주기억장치 주소 11의 내용을 읽어와 범용 레지스터 00에 저장한다. 그러면 주소 1 명령어에 대한 실행이 끝난다.

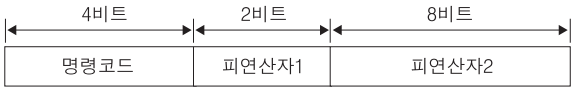


[그림 4-32] 주소 1 명령어에 대한 동작

다른 동작 과정은 앞에서 살펴본 내용과 큰 차이가 없으므로 생략한다.

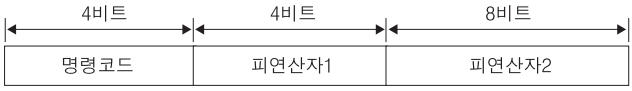
그런데 지금 살펴본 예는 비현실적이다. 명령어가 4개뿐인 중앙처리장치와 주소가 16개뿐인 주기억장치는 없을 것이다. 좀더 현실에 가깝게 명령어가 16개고, 주기억장치의 주소가 256개라고 확장해서 살펴보자.

명령어가 16개므로 명령 코드가 4비트는 되어야 한다. 그리고 주기억장치의 주소가 0부터 255의 256개므로 모든 주소를 표현하기 위해서는 명령어의 피연산자2 부분이 8비트는 되어야 한다. 이런 사항을 반영하면 명령어의 길이는 다음과 같이 14비트가 될 것이다.



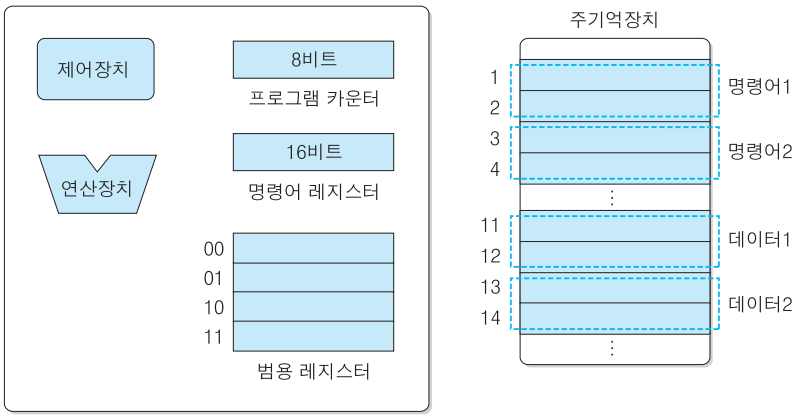
[그림 4-33] 명령어 길이 14비트로 확장

그러나 주기억장치는 바이트 단위로 동작하기 때문에 명령어의 길이를 16비트로 해주는 것이 바람직하다. 피연산자2는 1바이트 크기므로 그대로 두고, 명령코드 부분이나 피연산자1 부분의 크기를 2비트 키워서 명령어의 길이를 16비트로 만드는 것이 바람직하다. 다음은 피연산자2 부분을 4비트로 한 경우다.



[그림 4-34] 바람직한 명령어 길이 16비트

명령어의 크기가 16비트고 데이터의 크기도 16비트라면, 각 명령어와 데이터는 주기억장치 내 두 개의 주소 영역에 저장되고, 명령어 레지스터와 범용 레지스터의 크기도 16비트가 된다. [그림 4-35]는 이런 구조를 나타낸 것인데, 프로그램 카운터가 8비트인 이유는 주기억장치의 주소가 0부터 255까지기 때문이다.



[그림 4-35] 명령어와 데이터가 16비트인 경우의 구조

요약

1 컴퓨터 하드웨어는 중앙처리장치, 주기억장치, 입출력장치 그리고 이들을 연결해주는 시스템 버스로 구성된다.

구성요소	설명
중앙처리장치	프로그램 명령어를 실행하는 기능을 수행
주기억장치	실행 중인 프로그램과 프로그램 실행에 필요한 데이터를 일시적으로 저장
입출력장치	키보드, 모니터, 하드디스크 등으로 중앙처리장치나 주기억장치에 데이터를 입력하거나 출력

2 시스템 버스는 중앙처리장치, 주기억장치, 입출력장치들을 연결하는 일을 하는데, 주소 버스, 데이터 버스, 제어 버스의 세 가지가 있다.

구성요소	설명
주소 버스	중앙처리장치가 주기억장치로부터 읽을 데이터가 저장된 주기억장치의 주소나 데이터를 쓸 주기억장치의 주소가 전달
데이터 버스	중앙처리장치가 주기억장치로부터 읽거나 쓸 데이터가 전달
제어 버스	읽을지 쓸지에 대한 제어 정보가 전달

3 중앙처리장치는 인출(fetch), 해독(decode), 실행(execute) 단계로 구성된 일련의 동작을 반복함으로써 명령어를 실행해 나간다.

단계	설명
인출	주기억장치에 저장된 명령어 하나를 읽어 오는 단계
해독	읽어 온 명령어를 제어 정보로 해독하는 단계
실행	해독된 명령을 실행하는 단계

4 주기억장치는 저장된 정보를 관리하기 편하고 각 위치를 구분하기 위해 바이트(byte) 또는 워드(word) 단위로 분할해 주소(address)를 할당한다.

5 프로그램 명령어는 연산 코드와 피연산자 부분으로 이루어진다.

요약

6 중앙처리장치가 제공하는 명령어는 크게 데이터 전송 명령어, 연산 명령어, 분기 명령어로 분류할 수 있다.

구성요소	설명
데이터 전송 명령어	레지스터 또는 주기억장치의 데이터를 레지스터 또는 주기억장치로 이동하는 명령어와 입출력장치와 데이터를 주고받는 입출력 명령어
연산 명령어	더하기, 빼기, 곱하기, 나누기 등의 산술 연산을 하는 명령어로, AND, OR, NOT, XOR 등의 논리 연산 명령어, 레지스터에 저장된 비트들을 이동시키는 시프트 명령어
분기 명령어	다음에 실행될 명령어를 새롭게 지정하는 명령어

연습문제

1 시스템 버스의 역할과 종류에 대해 설명하여라.

2 프로그램이 실행되는 과정을 개략적으로 나타내어라.

3 중앙처리장치를 구성하는 세 가지 장치에 대해 설명하여라.

4 중앙처리장치가 한 명령어를 실행하는 세 단계에 대해 설명하여라.

5 주기억장치가 제공하는 두 가지 동작은 무엇인가?

6 프로그램 명령어의 기본 형식을 나타내어라.

7 대부분의 중앙처리장치가 제공하는 세 가지 명령어는 무엇인가?

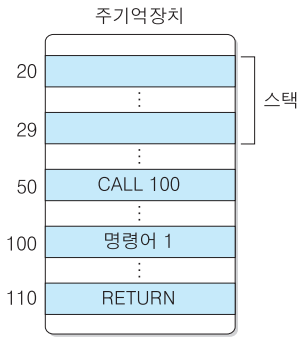
8 다음 명령어를 실행하는 과정을 나타내어라.

```
MOV BX [150]
```

9 다음 명령어를 실행하는 과정을 나타내어라.

```
ADD AX BX
```

10 50번지 명령어에 의해 함수가 호출되고 함수 실행이 종료되기까지의 동작 과정을 나타내어라.



연습문제

11 다음 연상 기호 코드 형식의 프로그램 실행 동작을 나타내어라.

```
MOV AX [11]
MOV BX [12]
ADD AX BX
MOV [13] AX
HALT
```

12 다음 2진 표기법 형식의 프로그램 실행 동작을 나타내어라.

```
00001011
00011100
10000001
01001101
11000000
```