제3장 컴퓨터 명령어

컴퓨터 명령어

명령어 집합

- 컴퓨터 명령어(instruction)
  - 컴퓨터가 수행해야 하는 일을 나타내기 위한 비트들의 집합
  - 일정한 형식을 가짐

- 컴퓨터에서 사용할 수 있는 명령어의 세트(set)
  - 모든 컴퓨터는 자신의 명령어 집합을 가지고 있다.
  - 명령어 집합은 그 컴퓨터의 구조적인 특성을 나타내는 가장 중요한 정보이다.
  - 동일 계열의 컴퓨터는 같은 명령어 집합이 사용된다.
  - 따라서 명령어 집합을 이용하여 컴퓨터 시스템의 구조를 살펴볼 수 있다.

명령어 집합

컴퓨터 명령어의 수행 기능

▶ 함수연산기능

- 덧셈, 시프트, 보수 등의 산술연산과 AND, OR, NOT 등의 논리연산 수행 기능

▶ 전달기능

- 레지스터들 사이의 정보전달 기능과 중앙처리장치와 주기억장치 사이의 정보전달 기능

▶ 제어기능

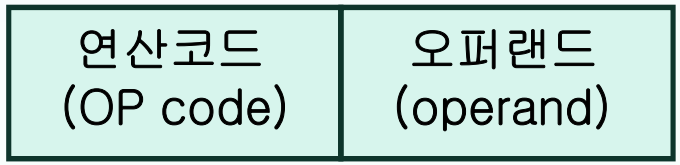
- 조건 분기와 무조건 분기 등을 통해 명령어의 수행 순서를 제어하는 기능

▶ 입출력 기능

- 주기억장치와 입출력장치 사이의 정보 이동 기능

명령어 구성형태

- 명령어는 필드(field)라는 비트그룹으로 이루어지며, 연산코드와 오퍼랜드 필드로 나뉘어진다.
  - ▶ 연산코드 : 처리해야 할 연산의 종류
  - ▶ 오퍼랜드 : 처리할 대상 데이터 또는 데이터의 주소



<명령어의 구성형태>

명령어 형식

□ 명령어를 구성하는 필드들의 수와 배치 방식 및 각 필드들의 비트 수를 말한다.

▶ 명령어는 컴퓨터의 내부구조에 따라 여러 가지 형식이 있다.

□ 명령어 형식의 분류

▶ 기억장소에 따른 명령어 형식

▶ 오퍼랜드의 수에 따른 명령어 형식

명령어 형식

□ 기억장소에 따른 명령어 형식

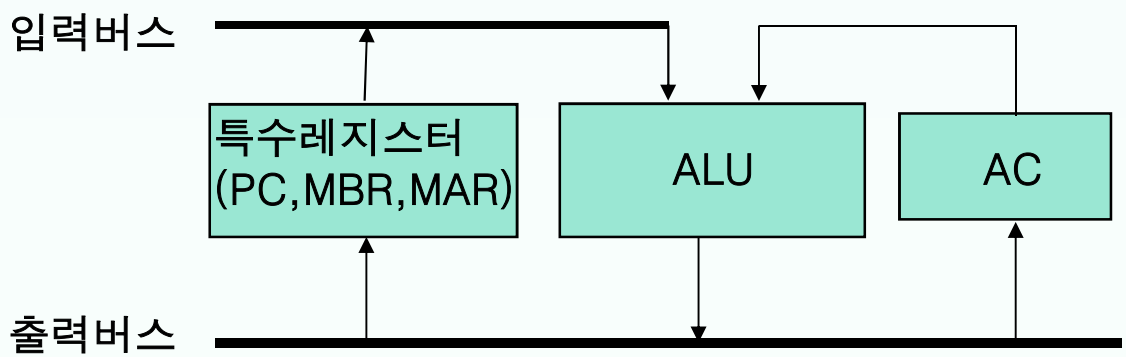
▶ 오퍼랜드가 기억되는 장소에 따라

1. 단일 누산기를 이용하는 명령어 형식
2. 다중 레지스터를 이용하는 명령어 형식
3. 스택 구조를 이용하는 명령어 형식

명령어 형식(기억장소에 따른)

1. 단일 누산기를 이용하는 명령어 형식

- ▶ **누산기를 가진 컴퓨터구조에서 사용되는 형식**
- ▶ **누산기 (AC: accumulator)**
  - 누산기를 가진 컴퓨터구조에서 중앙처리장치에 있는 유일한 데이터 레지스터로서 명령어가 수행될 때 오퍼랜드를 기억시키는 레지스터

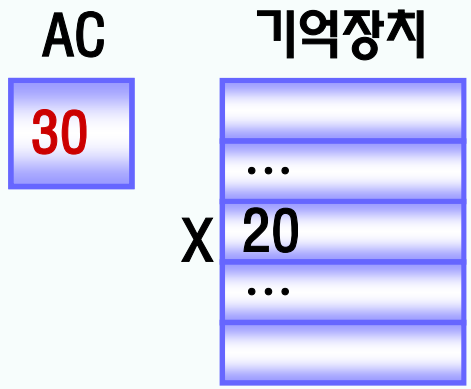


<누산기를 가진 컴퓨터 구조>

명령어 형식(기억장소에 따른)

▶ 단일 누산기를 이용하는 명령어 형식의 예

ADD X ; AC←AC+M[X] 예) 10 + 20 = 30



<의미>

'누산기(AC)에 있는 내용과 기억장치 X번지에 있는 내용을 더해서 누산기(AC)로 전송하라'

명령어 형식(기억장소에 따른)

▶ **단일 누산기를 이용하는 명령어 형식의 예**

〈예2〉 LOAD X ; $AC \leftarrow M[X]$

〈의미〉 ' 기억장치 X번지에 있는 내용을 누산기로 적재하라 '

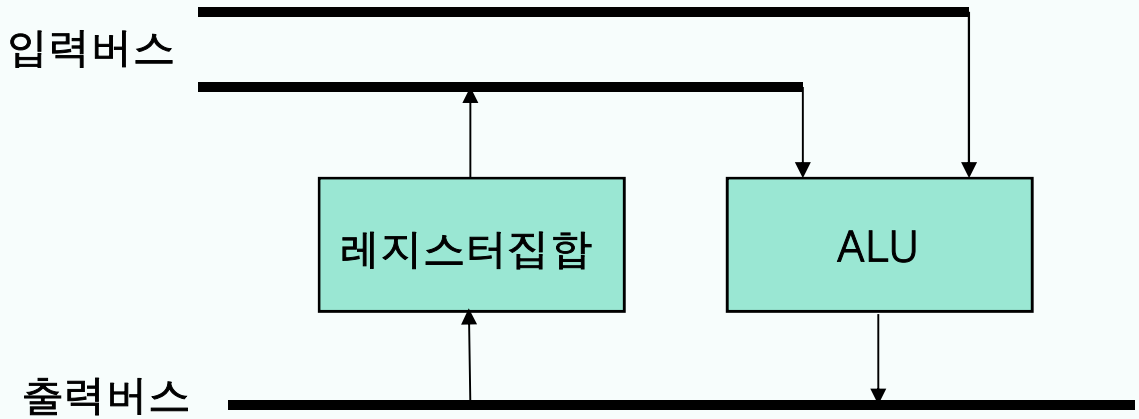
〈예3〉 STORE X ; $M[X] \leftarrow AC$

〈의미〉 ' 누산기의 내용을 기억장치 X번지에 저장하라 '

명령어 형식(기억장소에 따른)

2. 다중 레지스터를 이용하는 명령어 형식

- **다중 레지스터를 가진 컴퓨터 구조는 중앙처리장치 내에 여러 개의 레지스터를 가지고 있는 컴퓨터이다.**



<다중 레지스터를 가진 컴퓨터 구조>

명령어 형식(기억장소에 따른)

▶ 다중 레지스터를 이용하는 명령어 형식의 예

- 세 개의 레지스터를 사용하는 경우

ADD R1, R2, R3 ; R3←R1+R2

<의미>

'레지스터 R1의 내용과 레지스터 R2의 내용을 더해서 레지스터 R3로 전송하라'

- 두 개의 레지스터를 사용하는 경우

ADD R1, R2 ; R2←R1+R2

<의미>

'레지스터 R1의 내용과 레지스터 R2의 내용을 더해서 레지스터 R2로 전송하라'

명령어 형식(기억장소에 따른)

▶ **다중 레지스터를 이용하는 명령어 형식의 예**

〈예3〉 전달기능을 가진 명령어인 경우

MOVE R1, R2 ; *R2 ← R1*

〈의미〉 ' 레지스터 R1의 내용을 레지스터 R2로 전송하라 '

〈예4〉 주소필드 중 하나가 기억장치 주소필드인 경우

1) LOAD X, R1 ; *R1 ← M[X]*

2) STORE R1, X ; *M[X] ← R1*

〈의미 1〉 ' 기억장치 X번지의 내용을 레지스터 R1에 적재하라 '

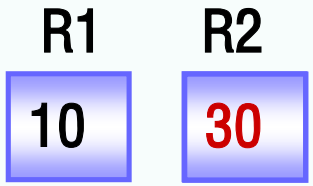
〈의미 2〉 ' 레지스터 R1의 내용을 기억장치 X번지에 저장하라 '

명령어 형식(기억장소에 따른)

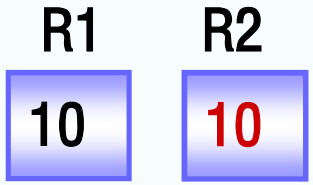
▶ 다중 레지스터를 이용하는 명령어 형식의 예

예) 10 + 20 = 30

ADD R1, R2, R3 ; R3 ← R1 + R2



ADD R1, R2 ; R2 ← R1 + R2

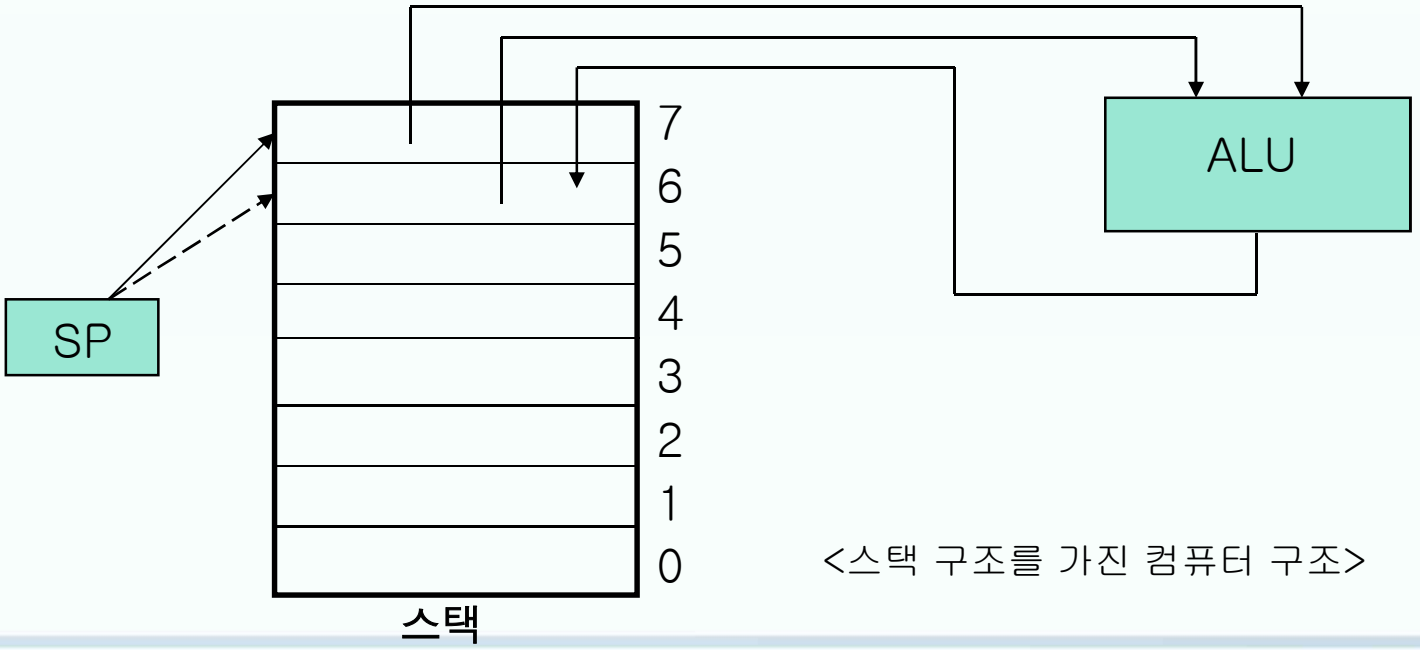


MOVE R1, R2 ; R2 ← R1

명령어 형식(기억장소에 따른)

3. 스택구조를 이용하는 명령어 형식

▶ 스택구조 컴퓨터는 연산에 필요한 오퍼랜드들을 기억장치 스택에 기억시켜야 하고, 연산의 결과도 스택에 기억시키는 구조이다.



<스택 구조를 가진 컴퓨터 구조>

명령어 형식(기억장소에 따른)

▶ 스택구조를 이용하는 명령어 형식의 예

- 주소필드를 사용하지 않는 경우

ADD ; TOS←TOS+TOS-1

<의미> '기억장치 스택의 최상위(TOS)의 내용과 그 아래(TOS-1)의 내용을 더해서 스택의 최상위(TOS)로 전송하라'

- 주소필드를 사용하는 경우

PUSH X ; TOS←M[X]

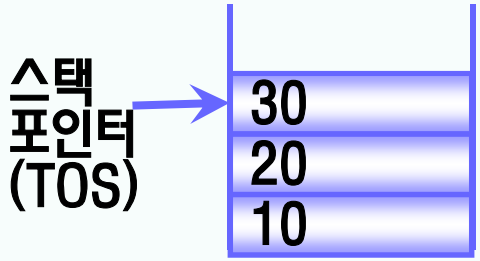
<의미> '기억장치 주소 X의 내용을 기억장치 스택의 최상위(TOS)로 전송하라'

POP X ; M[X]←TOS

<의미> '기억장치 스택의 최상위(TOS)에 있는 내용을 기억장치 주소 X 로 전송하라'

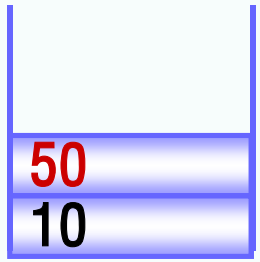
명령어 형식(기억장소에 따른)

▶ 스택구조를 이용하는 명령어 형식의 예



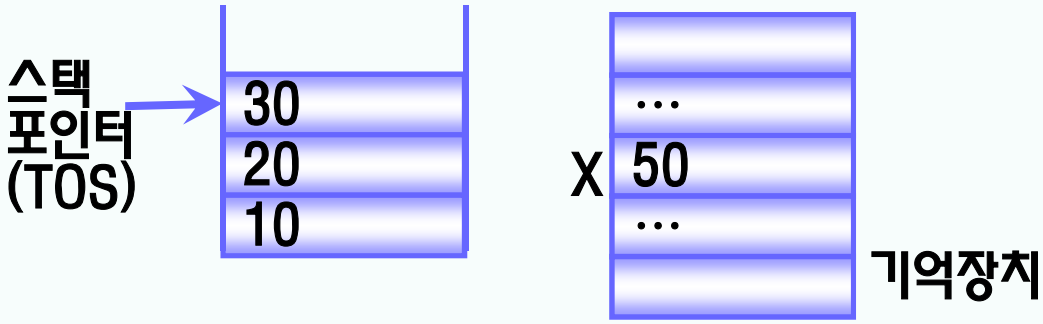
예) 30 + 20 = 50

ADD

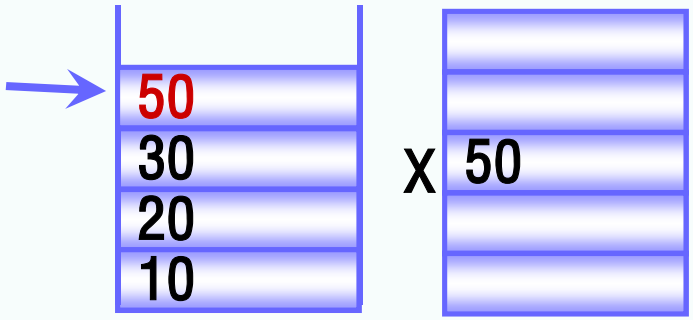


명령어 형식(기억장소에 따른)

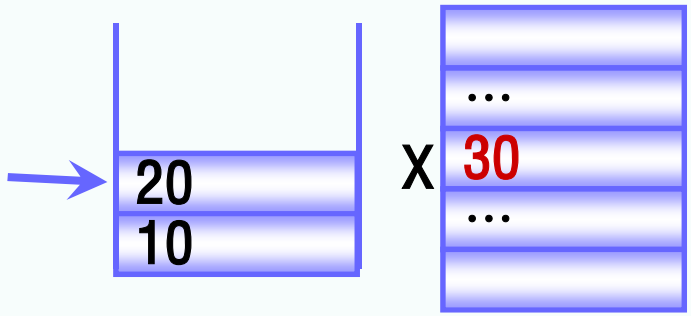
▶ 스택구조를 이용하는 명령어 형식의 예



PUSH X ; TOS ← M[X]



POP X ; M[X] ← TOS



명령어 형식

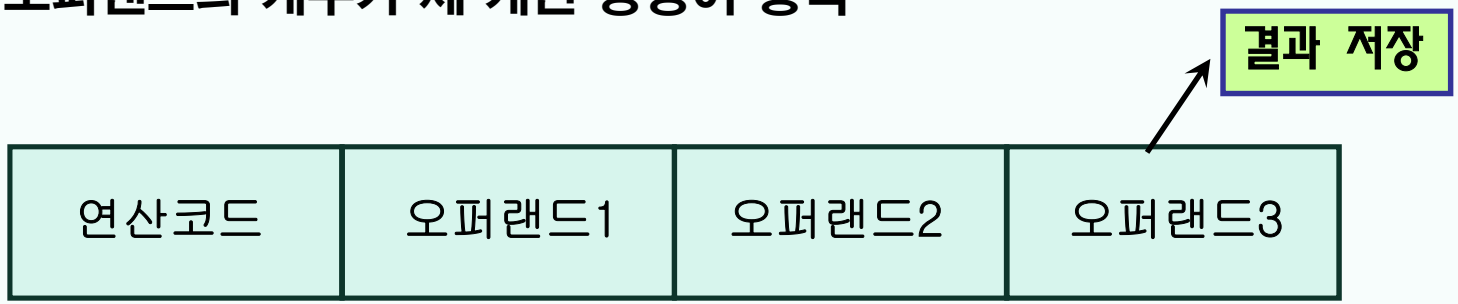
□ 오퍼랜드의 수에 따른 명령어 형식

1. 3-주소 명령어(three-address instruction)
2. 2-주소 명령어(two-address instruction)
3. 1-주소 명령어(one-address instruction)
4. 0-주소 명령어(zero-address instruction)

명령어 형식(오퍼랜드 수에 따른)

1. 3-주소 명령어

▶ **오퍼랜드의 개수가 세 개인 명령어 형식**



<3-주소 명령어의 형식>

명령어 형식(오퍼랜드 수에 따른)

▶ **3-주소 명령어의 예**

산술식 X = (A+B)×C 에 대해 3-주소 명령어를 이용한 프로그램

ADD *A, B, R1* ; *R1 ← M(A) + M(B)*

MUL *R1, C, X* ; *M(X) ← R1 × M(C)*

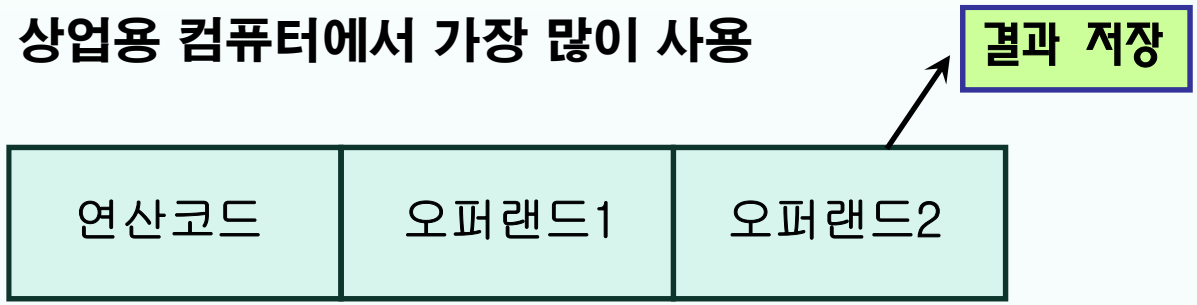
장점: 산술식을 프로그램화하는데 있어서 프로그램의 길이가 짧아짐.

단점: 3-주소명령어를 2진 코드화 했을 때 세 개의 오퍼랜드를 나타내기 위한 비트 수가 다른 주소 명령어 형식보다 많이 필요하다.

명령어 형식(오퍼랜드의 수에 따른)

2. 2-주소 명령어

- 오퍼랜드의 개수가 두 개인 명령어 형식
- 상업용 컴퓨터에서 가장 많이 사용



<2-주소 명령어의 형식>

명령어 형식(오퍼랜드의 수에 따른)

▶ **2-주소 명령어의 예**

산술식 X=(A+B)×C 에 대해 2-주소 명령어를 이용한 프로그램

```
LOAD    A, R1    ;   R1 ← M(A)
ADD     B, R1    ;   R1 ← R1 + M(B)
MUL     C, R1    ;   R1 ← R1 × M(C)
STORE   R1, X    ;   M(X) ← R1
```

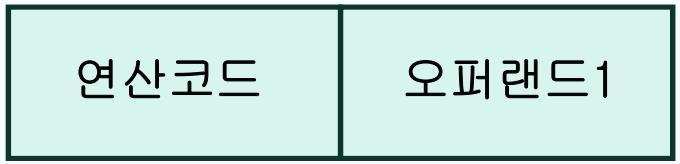
장점: 3-주소 명령어에 비해 명령어의 길이는 짧아짐.

단점: 같은 내용을 수행하기 위해 수행해야 하는 명령어의 수는 증가됨

명령어 형식(오퍼랜드의 수에 따른)

3. 1-주소 명령어

- 오퍼랜드의 개수가 하나인 명령어 형식
- 기억장치로부터 오퍼랜드를 가져오거나 연산결과를 저장하기 위한 임시적인 장소로 누산기 레지스터를 사용한다.



<1-주소 명령어의 형식>

명령어 형식(오퍼랜드의 수에 따른)

▶ **1-주소 명령어의 예**

산술식 X = (A+B)×C 에 대해 1-주소 명령어를 이용한 프로그램

```
LOAD    A    ;    AC ← M(A)
ADD     B    ;    AC ← AC + M(B)
STORE   X    ;    M(X) ← AC
LOAD    C    ;    AC ← M(C)
MUL     X    ;    AC ← AC  × M(X)
STORE   X    ;    M(X) ← AC
```

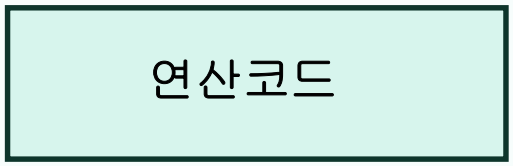
장점: 모든 연산은 누산기 레지스터와 기억장치에 저장된 오퍼랜드를 대상으로 수행.

단점: 프로그램을 수행하기 위해 사용되는 명령어의 수는 더 증가.

명령어 형식(오퍼랜드의 수에 따른)

4. 0-주소 명령어

- ▶ 스택구조에서 사용되는 형식
- ▶ 주소필드를 사용하지 않는다.



<0-주소 명령어의 형식>

명령어 형식(오퍼랜드의 수에 따른)

▶ **0-주소 명령어의 예**

산술식 X = (A+B)×C 에 대해 0-주소 명령어를 이용한 프로그램

```
PUSH   A    ;   TOS ← M(A)
PUSH   B    ;   TOS ← M(B)
ADD         ;   TOS ← TOS + TOS_1
PUSH   C    ;   TOS ← M(C)
MUL         ;   TOS ← TOS × TOS_1
POP    X    ;   M(X) ← TOS
```

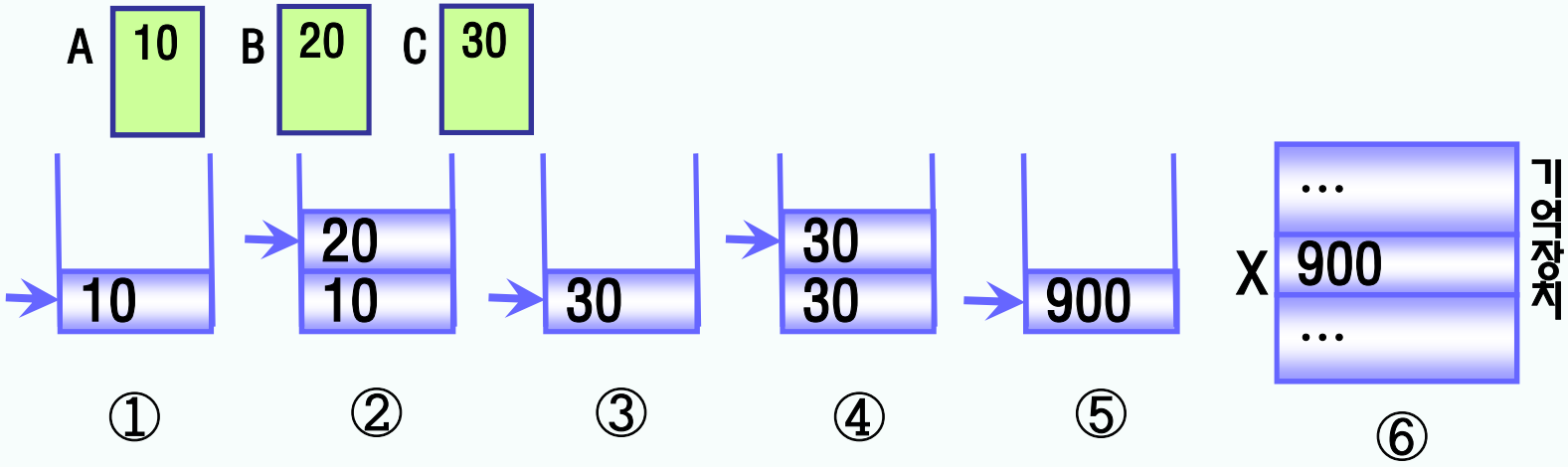
장점: 명령어의 길이가 매우 짧아서 기억공간을 적게 차지.

단점: 특수한 경우를 제외하고는 많은 양의 정보가 스택과 기억장치 사이를 이동하게 되어 비효율적.

명령어 형식(오퍼랜드의 수에 따른)

▶ 0-주소 명령어의 예

산술식 X = (A+B)×C



명령어 주소지정방식

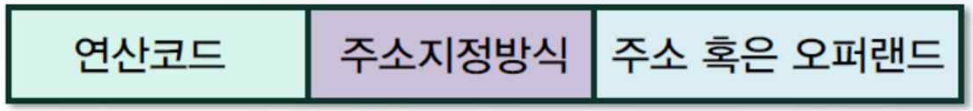
- **명령어 주소지정방식**
  - 프로그램 수행 시 오퍼랜드를 지정하는 방식.
  - 명령어의 주소 필드를 변경하거나 해석하는 규칙을 지정하는 형식.
  - 주소지정방식을 사용하면 명령어의 수를 줄일 수 있는 효과적인 프로그래밍 가능

- **유효주소**

  주소지정방식의 각 규칙에 의해 정해지는 오퍼랜드의 실제 주소.

명령어 주소지정방식

❖ 별도의 주소지정방식 필드를 가진 명령어 형식



- 연산코드 필드
  - 수행할 연산의 종류를 지정
- 주소지정방식 필드
  - 연산에 필요한 오퍼랜드의 주소를 알아내는 데 사용
- 주소 혹은 오퍼랜드 필드
  - 기억장치주소 혹은 레지스터를 나타낸다.

명령어 주소지정방식

- **주소지정방식의 종류**

1. 의미 주소지정방식
2. 즉치 주소지정방식
3. 직접 주소지정방식
4. 간접 주소지정방식
5. 레지스터 주소지정방식
6. 레지스터 간접 주소지정방식
7. 상대 주소지정방식
8. 인덱스된 주소지정방식

명령어 주소지정방식

1. 의미 주소지정방식

- 명령어형식에서 주소 필드를 필요로 하지 않는 방식.
- 연산코드 필드에 지정된 묵시적 의미의 오퍼랜드를 지정.

EX)　ADD ; TOS←TOS+TOS-1

▪ 기억장치 스택에서 ADD와 같은 명령어는 스택의 최상위 항목과 그 아래 항목을 더하여 스택의 최상위에 저장하는 명령어로서 오퍼랜드가 스택의 최상위에 있다는 것을 묵시적으로 가정함.

명령어 주소지정방식

2. 즉치 주소지정방식

- 명령어 자체 내에 오퍼랜드를 지정하고 있는 방식.
- 오퍼랜드 필드의 내용이 실제 사용될 데이터.
- 레지스터나 변수의 초기화에 유용.

EX) LDI 100, R1 ; R1←100

명령어 주소지정방식

3. 직접 주소지정방식

- 명령어의 주소필드에 직접 오퍼랜드의 주소를 저장시키는 방식
- 기억장치에의 접근이 한번에 이루어짐.

EX)　LDA ADRS ; AC ←M [ADRS]

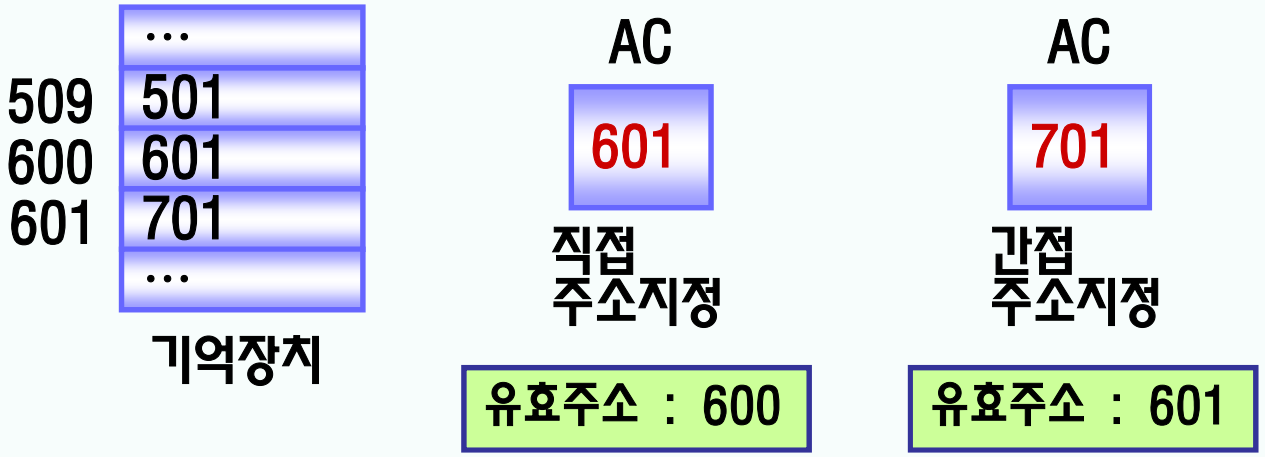
4. 간접 주소지정방식

- 명령어의 주소필드에 유효주소가 저장되어있는 기억장치 주소를 기억시키는 방식.

EX)　LDA[ADRS] ; AC ←M [M [ADRS]]

명령어 주소지정방식

직접/간접 주소지정방식 예)



명령어 주소지정방식

5. 레지스터 주소지정방식

- 오퍼랜드 필드에 레지스터 번호가 기억되는 방식
- 레지스터에 오퍼랜드가 들어있음.

EX)　LDA R1　;　AC ←R1

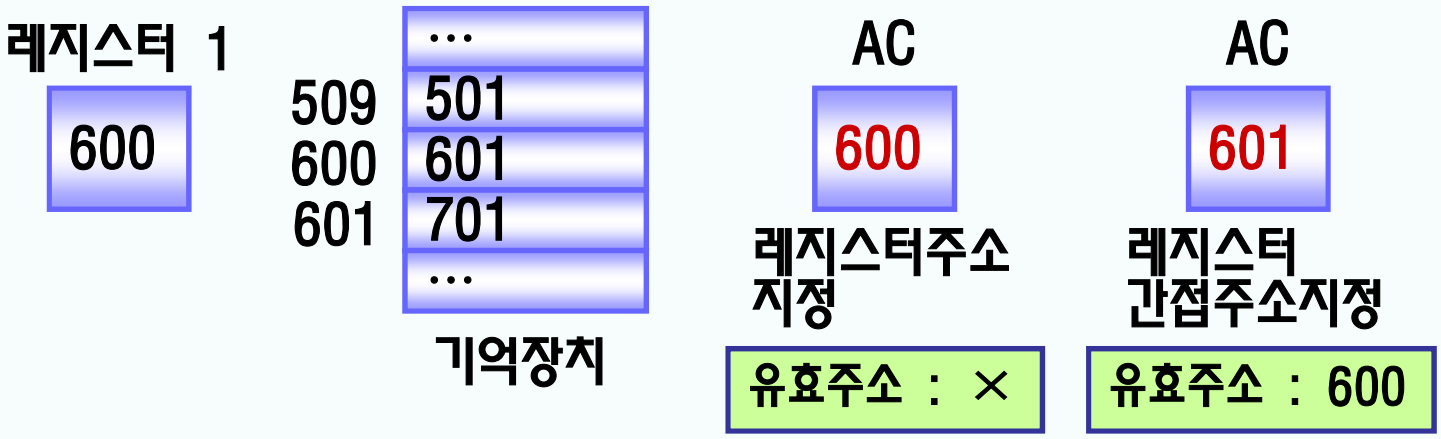
6. 레지스터 간접 주소지정방식

- 레지스터가 실제 오퍼랜드가 저장된 기억장치의 주소 값을 갖고 있는 방식.

EX)　LDA (R1)　;　AC ←M [R1]

명령어 주소지정방식

레지스터 (간접) 주소지정방식 예)



명령어 주소지정방식

7. 상대 주소지정방식

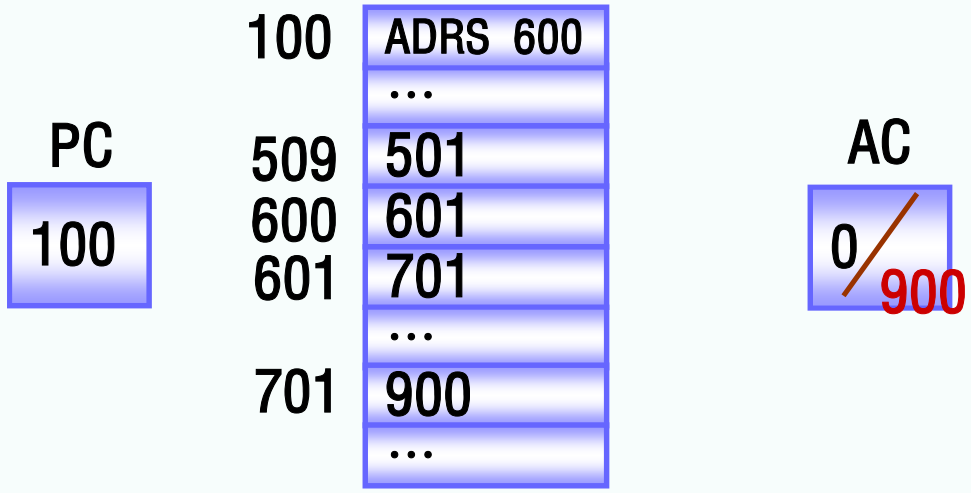
- 유효주소를 계산하기 위해 처리장치 내에 있는 특정 레지스터의 내용에 명령어 주소필드 값을 더하는 방식.
- 특정 레지스터로 프로그램카운터(PC)가 주로 사용.

유효주소 = 명령어 주소부분의 내용 + PC의 내용

EX) LDA $ADRS ; AC ←M [ADRS+PC]

명령어 주소지정방식

상대 주소지정방식 예)



유효주소는?

101 + 600 = 701

명령어 주소지정방식

8. 인덱스된 주소지정방식

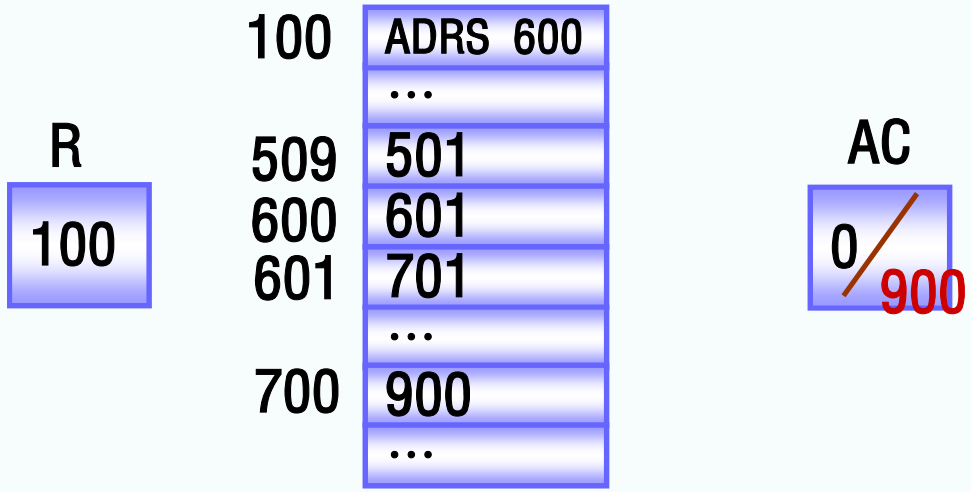
- 인덱스 레지스터의 내용을 명령어 주소 부분에 더해서 유효주소를 얻는 방식.

유효주소 = 명령어 주소부분의 내용 + 인덱스 레지스터의 내용

EX)　LDA ADRS(R1) ;　AC ←M [ADRS+R1]

명령어 주소지정방식

인덱스된 주소지정방식 예)



유효주소는?

100 + 600 = 700

주소지정방식의 요약

PC=350

R1=300

AC

연산코드 :

AC에 적재하라

기억장치

	연산코드	주소지정방식
350	ADRS, NBR=600	
351	다음 명령어	
352		
300	700	
600	900	
700	600	
900	950	
952	200	

즉치 주소
LDA #NBR ; *AC←NBR*

직접 주소
LDA ADRS ; *AC←M[ADRS]*

간접 주소
LDA [ADRS] ; *AC←M[[ADRS]]*

상대 주소
LDA $ADRS ; *AC←M[ADRS+PC]*

인덱스 주소
LDA ADRS(R1) ; *AC←M[ADRS+R1]*

레지스터 주소
LDA R1 ; *AC←R1*

레지스터 간접주소
LDA (R1) ; *AC←M[R1]*

주소지정방식의 요약

방식	기호표기	전송문	유효 주소	AC 내용
즉치주소	LDA #NBR	*AC←NBR*	351	600
직접주소	LDA ADRS	*AC←M[ADRS]*	600	900
간접주소	LDA[ADRS]	*AC←M[M[ADRS]]*	900	950
상대주소	LDA $ADRS	*AC←M[ADRS+PC]*	952	200
인덱스주소	LDA ADRS(R1)	*AC←M[ADRS+R1]*	900	950
레지스터주소	LDA R1	*AC←R1*		300
레지스터 간접	LDA (R1)	*AC←M[R1]*	300	700

명령어의 종류

명령어의 종류

- 데이터 전송명령어
- 데이터 처리명령어
- 프로그램 제어명령어

명령어의 종류

- **데이터 전송명령어**
  - 한 장소에서 다른 장소로 단지 데이터를 전송하는 명령어.
  - 레지스터와 레지스터 사이, 레지스터와 기억장치 사이, 또는 기억장치와 기억장치 사이에 데이터를 이동하는 기능.
  - 입출력 명령어가 포함.

명령어의 종류(데이터 전송명령어)

전송명령어	니모닉	기능
Load	LD	기억장치로부터 레지스터로의 전송
Store	ST	레지스터로부터 기억장치로의 전송
Move	MOVE	레지스터로부터 다른 레지스터로의 전송
Exchange	XCH	두 레지스터 간 또는 레지스터와 기억장치간의 데이터 교환
Push	PUSH	기억장치의 스택과 레지스터간의 데이터 전송
Pop	POP	
Input	IN	레지스터와 입출력장치 간의 데이터 전송
Output	OUT	

명령어의 종류

- 데이터 처리명령어

  – 데이터에 대한 연산을 실행하고 컴퓨터에 계산능력을 제공.

    1. 산술 명령어

    2. 논리와 비트 처리 명령어

    3. 쉬프트 명령어

명령어의 종류(데이터 처리명령어)

1. 산술명령어

- 사칙연산에 대한 명령어

산술명령어	니모닉	기 능
Increment	INC	1증가
Decrement	DEC	1 감소
Add	ADD	덧셈
Subtract	SUB	뺄셈
Multiply	MUL	곱셈
Divide	DIV	나눗셈
Add with carry	ADDC	캐리를 포함한 덧셈
Subtract with borrow	SUBB	빌림을 포함한 뺄셈
Negate	NEG	2의 보수

명령어의 종류(데이터 처리명령어)

2. 논리 및 비트 처리명령어

- 레지스터나 기억장치에 저장된 단어에 대한 2진 연산.
- 주로 2진 부호화 정보를 표현하는 비트 그룹이나 개별 비트를 처리하는데 사용.
- 비트 값을 0으로 만들거나, 기억장치 레지스터에 저장된 오퍼랜드에 새로운 비트 값을 삽입하는 것 등이 가능.

명령어의 종류(데이터 처리명령어)

논리 명령어	니모닉	기 능
Clear	CLR	모든 비트를 0으로 리셋
Set	SET	모든 비트를 1로 셋
Complement	COM	모든 비트를 반전
AND	AND	비트별 AND 연산
OR	OR	비트별 OR 연산
Exclusive-OR	XOR	비트별 XOR 연산
Clear carry	CLRC	캐리 비트의 리셋
Set carry	SETC	캐리 비트의 셋
Complement carry	COMC	(반전)보수

명령어의 종류(데이터 처리명령어)

3. 쉬프트 명령어

- 오퍼랜드의 비트를 왼쪽이나 오른쪽으로 이동시키는 명령어.
- 논리적 쉬프트와 산술적 쉬프트, 회전형 쉬프트 연산 등이 있음.

명령어의 종류(데이터 처리명령어)

쉬프트 명령어	니모닉	기 능
Logical shift right	SHR	오른쪽 쉬프트(왼쪽의 남는 비트는 0으로 채움)
Logical shift left	SHL	왼쪽 쉬프트(오른쪽의 남는 비트는 0으로 채움)
Arithmetic shift right	SHRA	부호비트는 고정(왼쪽의 남는 비트는 부호비트로 채움)
Arithmetic shift left	SHLA	부호비트는 고정(오른쪽의 남는 비트는 0으로 채움)
Rotate right	ROR	오른쪽으로 순환(버려지는 비트는 다시 왼쪽비트로)
Rotate left	ROL	왼쪽으로 순환(버려지는 비트는 다시 오른쪽비트로)
Rotate right with carry	RORC	캐리를 포함한 오른쪽 순환
Rotate left with carry	ROLC	캐리를 포함한 왼쪽 순환

명령어의 종류

- 프로그램 제어명령어
  - 프로그램 수행의 흐름을 제어
  - 다른 프로그램 세그먼트 (segment)로 분기

제어 명령어	니모닉	기 능
Branch	BR	조건 혹은 무조건적으로 유효주소로 분기
Jump	JMP	
Skip next instruction	SKP	조건이 만족되면 다음 명령어를 수행하지 않고 넘어감
Call procedure	CALL	서브루틴 호출
Return from procedure	RET	서브루틴 실행 후 복귀
Compare(by subtraction)	CMP	두 오퍼랜드의 뺄셈을 통해 상태 레지스터의 값을 변환
Test (by ANDing)	TEST	논리 AND 연산만 구현