컴퓨터 명령어

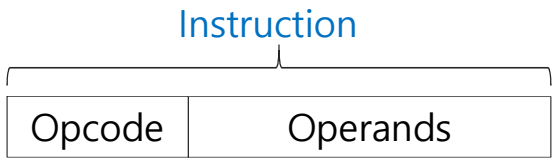
1. 컴퓨터 명령어 개요

- **컴퓨터 명령어(Computer Instruction)**
  - 컴퓨터의 동작을 제어하는 언어.
  - 명령어를 사용하여 컴퓨터 동작을 제어.

- **명령어 세트 (Instruction Set)**
  - 컴퓨터에서 실행할 수 있는 명령어들의 집합
    - CPU 유형에 따라 명령어 세트가 다름.
  - 명령어 세트 구조(ISA : Instruction Set Architecture)
    - 명령어 세트의 구조적 측면.
    - 컴퓨터 구조를 결정하는 핵심요소.

1.1 명령어 구성요소

- **명령어는 명령어 유형(Operation Code)과 명령어 실행에 필요한 데이터(Operands), 2 가지 요소로 구성된다.**



- **명령어 유형**
  - CPU가 실행할 작업을 결정.
  - 동작 코드 (Operation Code 또는 Opcode)로 정의
  - 바이너리로 인코딩

- **명령어 실행에 필요한 데이터 (Operand)**
  - 소스 오퍼랜드(Source Operand) : 명령어 실행시 입력으로 사용되는 하나 이상의 오퍼랜드
  - 목적 오퍼랜드(Destination Operand) : 명령어 실행 결과
  - Operand 개수, 용도, 표현방법은 연산유형에 따라 다양하게 결정됨.

1.2 명령어 실행순서

- **명령어는 메모리에 저장된 순서에 따라 순서적으로 실행된다.**
  - 묵시적 (implicit) 지정
    - 다음에 실행할 명령어 위치 미지정.
    - 실행중인 명령어의 바로 다음 위치
  - 명령어 실행중, 다음에 실행할 명령어의 위치를 저장하고 있는 PC(Program Counter)는 자동으로 '+1' 된다.

- **명령어 실행순서를 변경하는 방법**
  - 명시적 (explicit) 지정
    - 다음에 실행할 명령어가 저장된 위치 정보를 명령어 내부에 포함.
  - 분기(Branch) 명령어나 함수 호출(Function Call) 명령어를 사용하여 명령어를 실행흐름을 변경한다.

1.3 명령어 코드

- **명령어를 표현하는 코드**
  - 기계어 코드(Machine Code)
    - 명령어의 바이너리 표현.
  - 어셈블리 코드(Assembly Code)
    - 문자 심볼을 사용하여 Machine code의 가독성을 높인 표현.
      - ADD, SUB, MUL, LD, ST, ….
    - 기계 중심 언어(Machine-oriented).
  - 고수준 프로그래밍 언어(HPL: High-level Programming Language)
    - 명령어의 추상성을 향상시킨 언어.
    - 사용자 중심 언어(User-oriented).
    - C, C++, Java, Python, ….

1.4 명령어 길이

- **명령어 길이(Instruction Length)는 1개의 명령어를 구성하는 비트들의 크기.**

- **명령어 길이 결정 요소**
  - 메모리 용량 및 구조
  - 명령어 종류
  - 오퍼랜드 유형과 개수

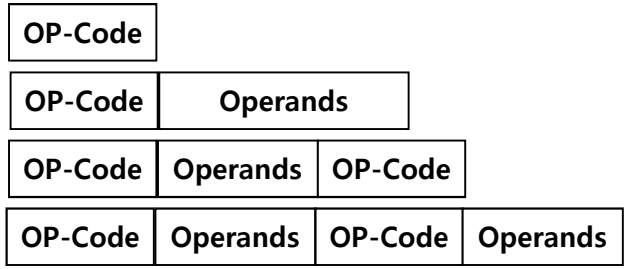
2. 명령어 포맷

- **명령어는 일련의 비트들로 표현**
  - 비트 그룹(Field) 단위로 구분
  - 필드는 명령어의 각 구성요소들에 대응

- **명령어 포맷(Instruction Format)**
  - 명령어 구성요소 필드의 비트 배치 및 해석 방법
    - 명령어 길이에 영향
  - 명령어 필드에 대한 비트 할당 방법

- **명령어 세트에서 1개 이상의 명령어 형식을 사용**

- **명령어 형식의 설계**
  - 매우 복잡한 작업
  - 비트 패턴 할당
    - 소비전력 고려

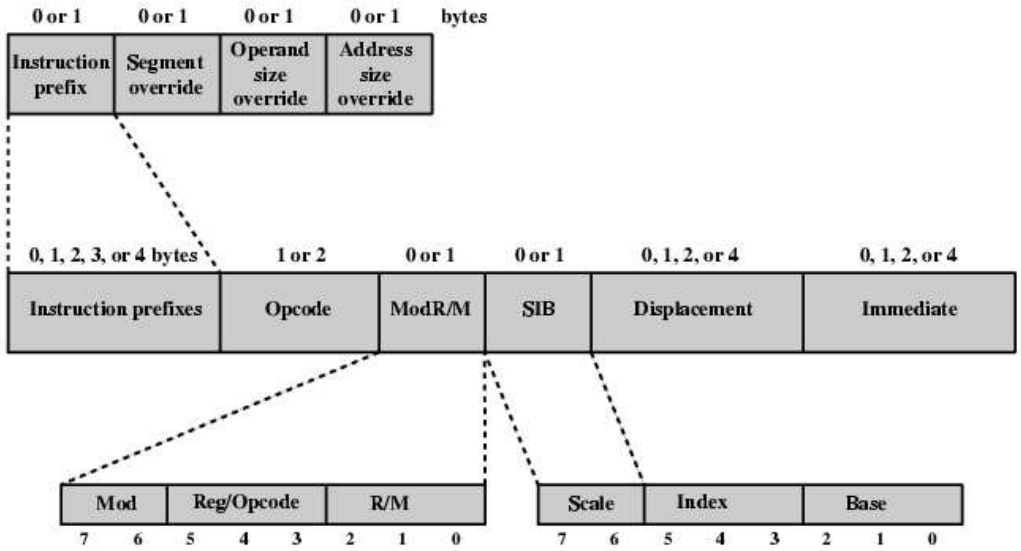


2.1 비트 할당

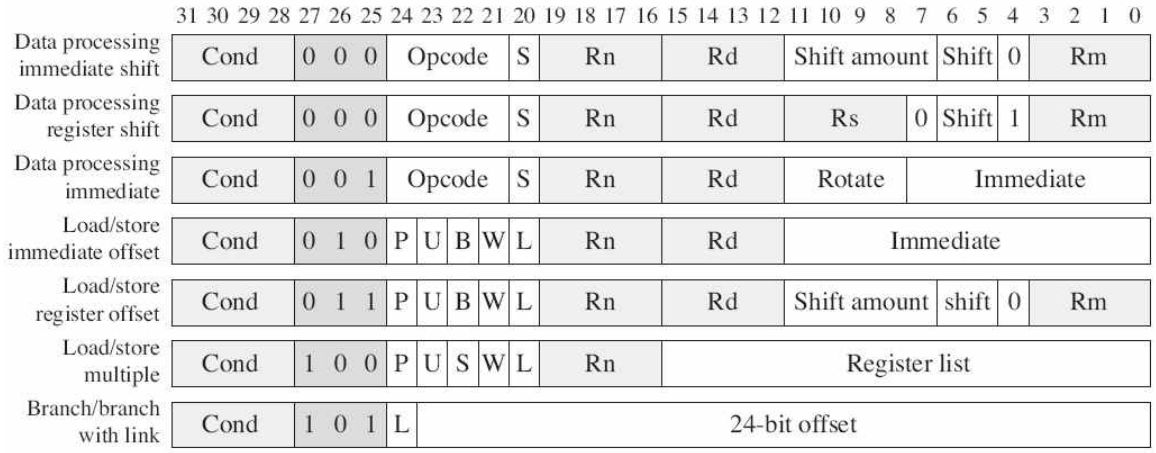
- **연산 코드 (op-code) 의 종류와 주소지정 능력 사이의 상호조정이 필요**
  - 연산 종류 증가 → 연산 코드 증가 → 주소지정 능력 감소

- **주소지정 비트 수 결정에 영향을 주는 요소**
  - 주소지정 방식의 수 (Number of addressing modes)
  - 오퍼랜드의 수 (Number of operands)
  - 레지스터 세트의 수 (Number of register sets)
  - 주소 범위 (Address range)
  - 메모리의 접근단위 (Address granularity) : byte, half-word, word

2.2 Pentium 프로세서 명령어 포맷



2.3 ARM 프로세서 명령어 포맷



3. 명령어 설계시 고려사항

- **연산 종류 (Operation Repertoire)**
  - 연산 종류 수, 연산 작업 유형, 연산의 복잡성
- **데이터 유형 (Data Types)**
- **명령어 형식 (Instruction Formats)**
  - 명령어 길이 (비트 수), 오퍼랜드 개수, 명령어 필드의 크기.
- **레지스터 (Registers)**
  - 명령어들에 의해 참조될 레지스터들의 수
  - 레지스터들의 용도
- **주소지정 (Addressing) 방식**

명령어 오퍼랜드

1. 명령어의 오퍼랜드

- **명령어의 오퍼랜드는 명령어 실행에 필요한 데이터를 말한다.**
  - 명령어 유형에 따라 구성방법과 해석방법이 달라진다.

- **오퍼랜드 유형**
  - 주소(Address)
  - 수(Numeric Data)
    - 정수(integer), 부동 소수점(floating point), 10진수(decimal)
  - 문자(Character)
    - ASCII
  - 논리적 데이터 (Logical data)
    - 비트 단위 처리
      - 기억장치의 효율적 활용
      - 비트별 조작처리 가능

1.1 오퍼랜드 저장 위치

- **명령어 내부**
  - 실행할 명령어 내부에 포함
  - 즉치 (Immediate) 값, 상수(Constant) 값,

- **CPU 내부 레지스터**
  - 레지스터 이름 (R0, R1, …., Rn) 으로 표기

- **메모리**
  - 프로그램 메모리 또는 데이터 메모리

- **I/O 장치의 내부 레지스터 또는 메모리**

1.2 오퍼랜드 수 결정시 고려사항

- **많은 오퍼랜드를 사용할 경우**
  - 명령어의 복잡도 증가하지만 프로그램의 융통성이 향상된다.
  - 명령어 길이가 증가하지만, 프로그램 전체 명령어 수가 감소된다.
  - 여러 개의 레지스터를 사용할 수 있어서, 프로그램의 실행 속도를 향상시킬 수 있다.

- **오퍼랜드의 수가 적으면 ?**
  - 명령어들이 단순해지며, CPU가 덜 복잡해진다.
  - 명령어 길이가 짧아지기 때문에,
    - 프로그램의 전체 명령어들의 수가 증가할 수 있다.
    - 프로그램 크기가 증가, 복잡해지고, 실행시간이 증가할 수 있다.

2. 오퍼랜드 어드레싱

- **오퍼랜드가 명령어 내부에 있는 상수를 제외한 나머지 저장장소는 모두 어드레스를 사용하여 위치를 지정한다.**
  - 레지스터 어드레스(레지스터 명), 메모리 어드레스
  - I/O 장치의 경우, 레지스터/메모리 어드레스

- **3-어드레스 명령어**
  - 2개의 소스 오퍼랜드, 1개의 목적 오퍼랜드로 구성.
    - A = B + C;
  - 명령어 길이 증가

- **2-어드레스 명령어**
  - 오퍼랜드 하나가 소스 오퍼랜드와 결과 오퍼랜드에 같이 사용.
    - A = A + B
  - 명령어 길이 절약
    - 중간 값을 저장하기 위한 임시 저장장소 필요

2. 오퍼랜드 어드레싱

- **1-어드레스 명령어**
  - 2 번째 주소 = 묵시적 주소
    - 주로 레지스터 (ACC : Accumulator) 를 사용.
    - ACC = ACC + B
  - 초기 컴퓨터에서 많이 사용.

- **0-어드레스 명령어**
  - 모든 주소가 묵시적 (Implicit)
    - 스택 (Stack) 사용

0-Operand 명령어 사용례
(c = a + b 경우)

```
PUSH a
PUSH b
ADD
POP c
```

3. 어드레싱 구조

- **오퍼랜드 어드레싱 유형에 따라 어드레싱 구조(Addressing Architecture)가 결정된다.**
- **Memory-To-Memory Architecture**
  - 오퍼랜드가 모두 메모리 어드레스를 사용할 경우.

ADD T1, A, B	$M[T1] \leftarrow M[A] + M[B]$
ADD T2, C, D	$M[T2] \leftarrow M[C] + M[D]$
MUL X, T1, T2	$M[X] \leftarrow M[T1] \times M[T2]$

- **Register-Memory Architecture**
  - 오퍼랜드가 레지스터 어드레스와 메모리 어드레스를 같이 사용하는 경우,

ADD	R1,A	$R1 \leftarrow R1 + M[A]$

3. 어드레싱 구조

- **Register-to-Register Architecture**
  - Load/Store Architecture
  - 데이터 로딩과 저장 명령어만 메모리 액세스.
  - 모든 연산은 레지스터에 저장된 오퍼랜드만 사용.

LD	R1, C	$R1 \leftarrow M[C]$
LD	R2, D	$R2 \leftarrow M[D]$
ADD	R1, R1, R2	$R1 \leftarrow R1 + R2$
MUL	R1, R1, R3	$R1 \leftarrow R1 \times R3$
ST	X, R1	$M[X] \leftarrow R1$

- **Stack Architecture**
  - 스택을 이용한 데이터 로딩 및 연산
    - 데이터 로딩 : PUSH 명령어
    - 연산 : 스택에 저장된 오퍼랜드에 대한 연산

```
PUSH A
PUSH B
ADD
```

명령어 유형

1. 명령어 분류

- **데이터 전송 (data transfer) 명령어**
- **데이터 처리 명령어**
  - 산술 연산 (arithmetic)
  - 논리 연산 (logical)
  - 변환 (conversion)
- **입출력 연산 (I/O)**
- **시스템 제어 (system control)**
- **프로그램 실행흐름 제어 (transfer of control)**

2. 데이터 전송 명령어

- **데이터 전송 (Data Transfer) 명령어에 포함되어야 할 사항**
  - 데이터 전송 소스 오퍼랜드와 목적 오퍼랜드의 위치 정보
  - 전송할 데이터 크기 정보
  - 어드레싱 모드에 대한 정보

- **데이터 전송 명령어가 메모리를 액세스할 경우,**
  - 오퍼랜드의 메모리 어드레스 계산 → 가상 어드레스를 사용하는 경우, 물리 어드레스로 변환 → 캐시(Cache) 에 존재여부 검사 → 캐시에 없는 경우, 메모리 읽기/쓰기 접근.

2.1 데이터 전송 명령어 유형

- 메모리 관련 명령어 : Load/Store/Move
- 레지스터 이동 : Move/Exchange
- 레지스터 값 설정 : Set/Clear
- 스택 전송 : Push/Pop

Operation Name	Description
Move (transfer)	Transfer word or block from source to destination
Store	Transfer word from processor to memory
Load (fetch)	Transfer word from memory to processor
Exchange	Swap contents of source and destination
Clear (reset)	Transfer word of 0s to destination
Set	Transfer word of 1s to destination
Push	Transfer word from source to top of stack
Pop	Transfer word from top of stack to destination

2. 데이터 처리 명령어

- **데이터 처리 명령어**
  - 산술 연산 (Arithmetic)
  - 논리 연산 (Logical)
  - 변환 (Conversion)

2.1 산술 연산

- **산술 연산 (arithmetic)**
  - ALU 를 사용한 연산
  - 조건 코드 (condition code) 및 플래그 (flag) 설정.

Add	Compute sum of two operands
Subtract	Compute difference of two operands
Multiply	Compute product of two operands
Divide	Compute quotient of two operands
Absolute	Replace operand by its absolute value
Negate	Change sign of operand
Increment	Add 1 to operand
Decrement	Subtract 1 from operand

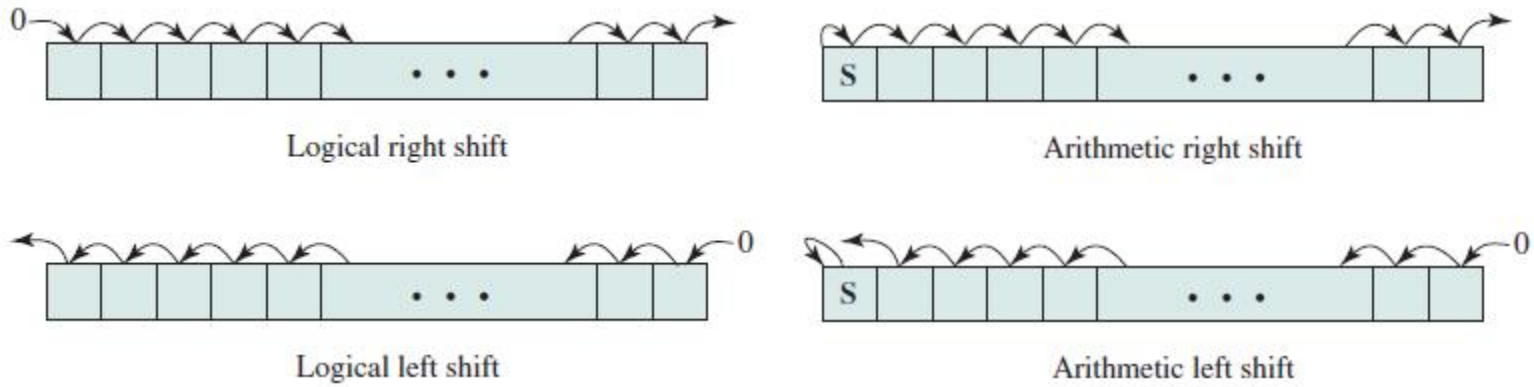
2.2 논리 연산

- **논리 연산 (logical)**
  - AND, OR, NOT
  - 비트단위 (Bitwise) 연산 : bit-mask
  - Shifting & rotating

AND	Perform logical AND
OR	Perform logical OR
NOT (complement)	Perform logical NOT
Exclusive-OR	Perform logical XOR
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
Shift	Left (right) shift operand, introducing constants at end
Rotate	Left (right) shift operand, with wraparound end

2.3 쉬프트 연산

- **쉬프트 연산 (Shift Operation)**
  - 이동방향 : Left, Right
  - 쉬프트 유형 : Arithmetic shift, Logical shift, Rotate
  - 쉬프트 크기(Amount) : 이동량. One-bit, multi-bit

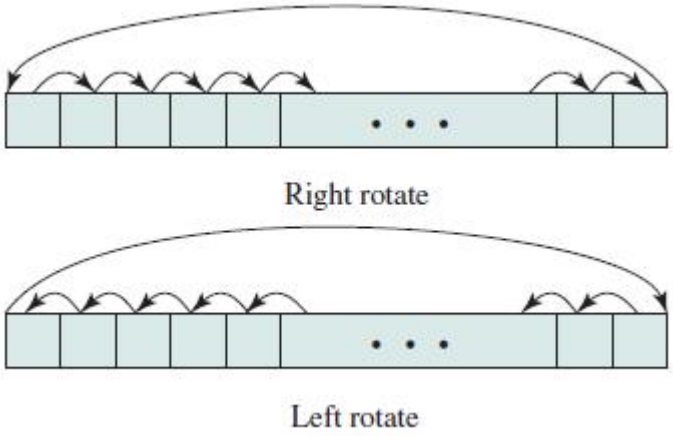


Logical right shift　　Arithmetic right shift

Logical left shift　　Arithmetic left shift

2.4 Rotate 연산

- **Rotate 연산**
  - MSB와 LSB를 연결한 쉬프트 연산

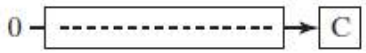
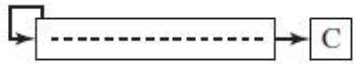
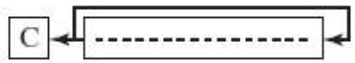
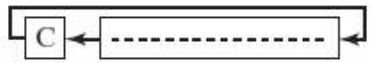


Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Shift 및 Rotate 예

2.5 Intel IA-32 의 쉬프트 연산

- 자리올림(C : Carry)를 포함한 쉬프트 연산

Name	Mnemonic	Diagram
Logical shift right	SHR	
Logical shift left	SHL	
Arithmetic shift right	SHRA	
Arithmetic shift left	SHLA	
Rotate right	ROR	
Rotate left	ROL	
Rotate right with carry	RORC	
Rotate left with carry	ROLC	

2.6 포맷 변환

- **포맷 변환 (Format Conversion)**
  - 데이터 포맷 변환 : bin-2-decimal, bin-2-BCD,

Translate	Translate values in a section of memory based on a table of correspondences
Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

3. 입출력 연산

- **입출력 연산 (I/O)**
  - 입출력 장치 전용
    - 입출력 장치 접근 방법에 따라 명령어 용도가 다름.

Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (write)	Transfer data from specified source to I/O port or device
Start I/O	Transfer instructions to I/O processor to initiate I/O operation
Test I/O	Transfer status information from I/O system to specified destination

4. 시스템 제어

- **시스템 제어 (System Control)**
  - 운영체제 용도의 특권 명령어 (privileged instructions)
    - 시스템 제어 레지스터 (control register) 액세스
    - storage protection key 수정
    - process control block 액세스

5. 프로그램 실행흐름 제어

- **제어권 이동 (transfer of control)**
  - 프로그램의 실행흐름 제어
    - Branch / Skip / Procedure Call

- **분기 (Branch)**
  - Conditional / unconditional
  - Forward / backward
  - Near / far

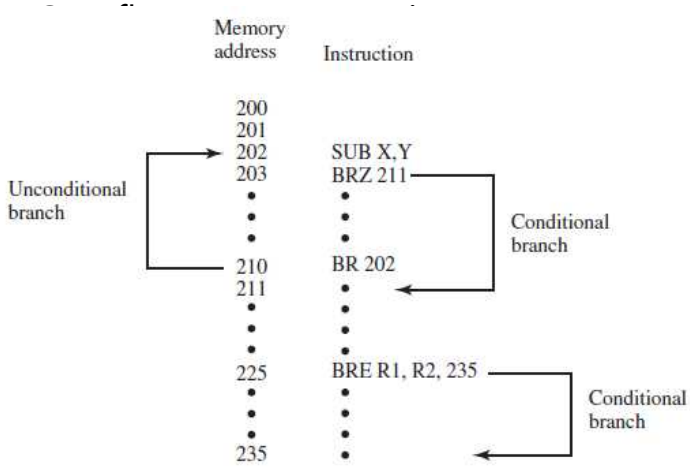
- **Skip**
  - ISZ (increment and skip if zero)
  - 명령어 1개를 Skip.

- **프로시저 호출 (procedure call)**
  - 함수 호출(Function call)
  - 인터럽트 처리(Interrupt handling)

Operation Name	Description
Jump (branch)	Unconditional transfer; load PC with specified address
Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
Jump to Subroutine	Place current program control information in known location; jump to specified address
Return	Replace contents of PC and other register from known location
Execute	Fetch operand from specified location and execute as instruction; do not modify PC
Skip	Increment PC to skip next instruction
Skip Conditional	Test specified condition; either skip or do nothing based on condition
Halt	Stop program execution
Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
No operation	No operation is performed, but program execution is continued

5.1 분기 명령어

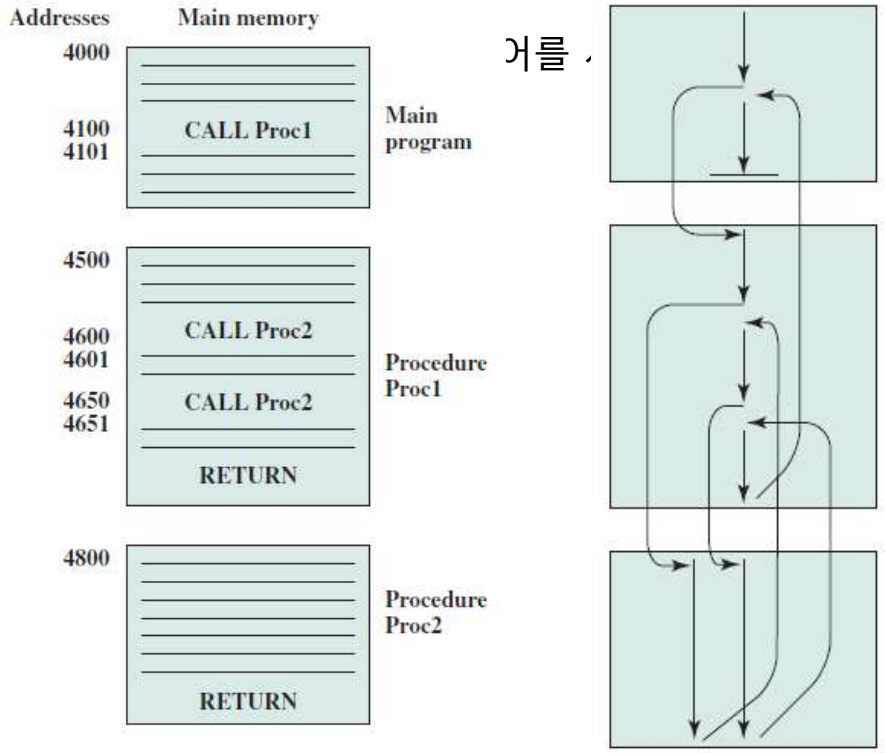
- **무조건/조건 분기**
  - 조건(Condition)을 테스트하여, 그 결과에 따라 분기
  - 조건 : 주로 연산결과의 상태 플래그(Flag) 상태. (Car
- **분기방향**
  - 어드레스 증가방향 분기 : 전방향(Forward) 분기.
  - 어드레스 감소방향 분기 : 후방향(Backwar) 분기.
- **분기 거리**
  - 현재 실행위치에서 분기하는 거리에 따라 Near/Far
  - 구분 기준은 시스템에 따라 다름.



5.2 프로시저 호출

- **프로시저 호출(Procedure Call)**
  - Caller 함수에서 Call 명령어로 호출하고 어를 ...

- **프로시저 사용**
  - 이점
    - 코딩 비용 절약
    - 프로그램 메모리 공간 절약
    - 모듈화 정도(Modularity) 향상
  - 불리한 점
    - 호출/복귀로 인한 실행속도 저하
    - 스택공간 사용부담



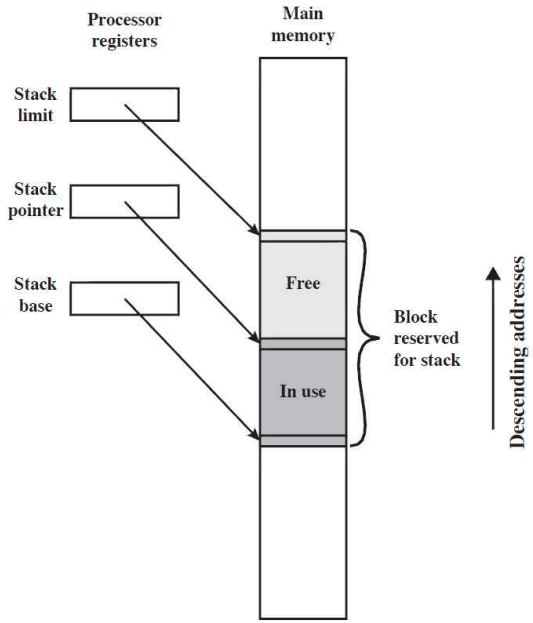
프로그램 실행 흐름

5.3 프로시저 호출 규약

- **호출규약 (Call Convention)**
  - Caller 함수에서 사용한 레지스터의 임시 저장값 및 함수 전달 파라미터에 관한 관리 규약
  - 컴파일러 및 프로그래머가 참고
    - 예) C언어와 어셈블리 언어를 연동할 경우.

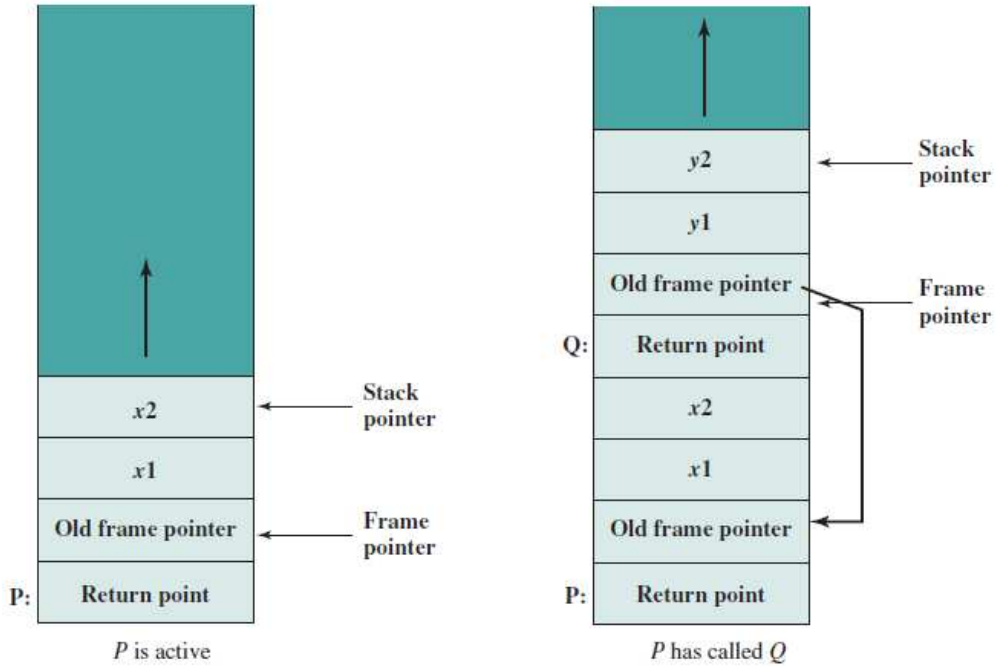
5.4 스택

- **스택(Stack)**
  - 프로시저 사이의 공용 데이터 저장공간
    - 데이터 메모리 공간의 일부를 LIFO(Last-In-First-Out) 형태로 사용
  - 전용 명령어 사용 : PUSH(데이터 저장) / POP(데이터 인출)
  - 스택의 현 위치
    - Top of Stack : 스택의 저장위치
    - 스택 포인터(Stack Pointer)를 사용하여 위치정보 저장
- **스택 유형**
  - Ascending / Descending : 스택 어드레스 증가방향에 따라 구분
  - Full / Empty : Top of Stack 위치에 따라 구분
- **스택에 저장되는 데이터**
  - 복귀주소(Return Address)
  - 함수 전달인자 (Function Parameters)
  - CPU 상태정보 (Status)
  - 사용중인 레지스터 내용
  - 함수에서 사용할 지역변수(Local Variables)



5.5 스택 프레임

- **스택 프레임 (Stack Frame)**
  - 하나의 프로시저를 호출하기 위해 저장되는 복귀 주소를 포함한 데이터들의 전체 집합



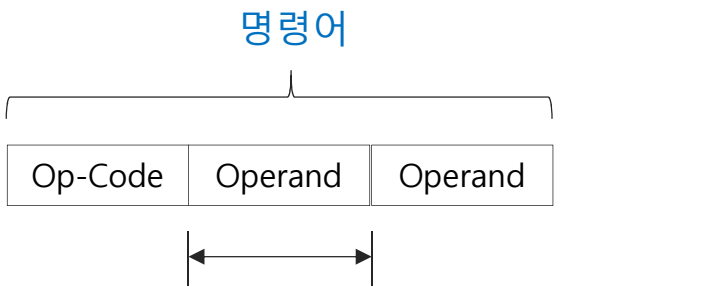
Stack Frame Growth Using Sample Procedures P and Q

어드레싱 모드 개요

1. 어드레싱 모드

- **명령어 오퍼랜드 필드**
  - 명령어에 정의된 오퍼랜드 필드에 저장되는 데이터
    - 순수 데이터, 어드레스

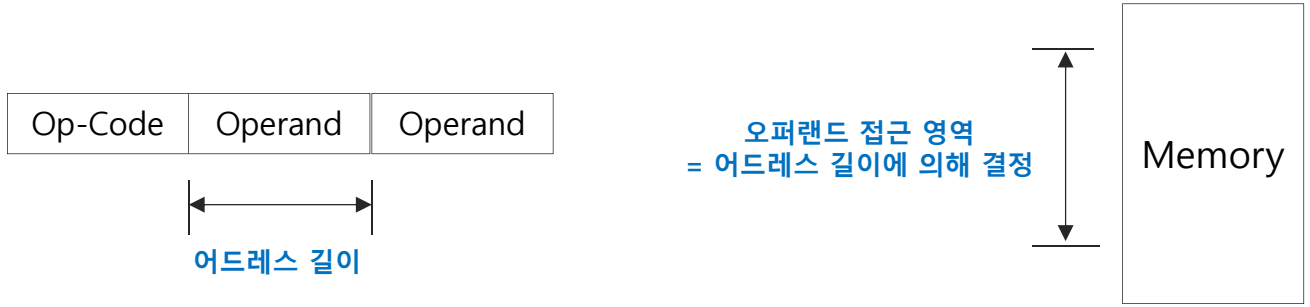
- **어드레싱 모드(Addressing Mode)**
  - 실제 데이터를 저장하고 있는 장소 정보를 명령어 오퍼랜드 필드내에 다양한 형태로 지정.
    - 데이터 저장 장소 : 명령어 내부, 레지스터, 메모리
    - 데이터 저장 장소 정보 : 레지스터 번호, 메모리 어드레스



2. 오퍼랜드 필드의 제약성

- **오퍼랜드 필드의 제약성**
  - 명령어내에서 오퍼랜드 필드 공간을 충분하게 확보하기 어렵다.
    - 오퍼랜드 필드 길이가 증가하면 명령어 길이가 증가.
    - 오퍼랜드 필드 길이가 증가하면 오퍼랜드 접근 영역 확대 가능.
  - 오퍼랜드 필드 길이와 명령어 길이와의 균형(Trade-off) 필요

- **오퍼랜드 필드의 제약성 해결방법**
  - 명령어에서 다양한 어드레싱 모드를 사용



3. 어드레싱 모드 설계 고려요소

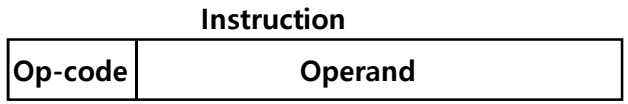
- **다양한 어드레싱 모드를 설계시 고려할 점**
  - 오퍼랜드 필드 길이와 어드레싱 모드 사이의 균형
    - 제한된 오퍼랜드 필드에서 복잡한 어드레싱 모드를 사용하여 오퍼랜드 접근 범위를 확장 가능
  - 메모리 참조 횟수와 주소 계산 복잡성사이의 균형
    - 어드레스 계산이 복잡하면 메모리 참조 횟수가 증가할 수 있음.
  - 어드레싱 모드 해석 부담
    - 실제 주소 계산 과정이 필요.

4. 어드레싱 모드 유형

- **어드레싱 모드 유형**
  - 즉치 주소 지정 방식(Immediate Addressing Mode)
  - 직접 주소 지정 방식(Direct Addressing Mode)
  - 간접 주소 지정 방식(Indirect Addressing Mode)
  - 레지스터 주소 지정 방식(Register Addressing Mode)
  - 레지스터 간접 주소 지정 방식(Register Indirect Addressing Mode)
  - 변위 주소 지정 방식(Displacement/Indexed Addressing Mode)
  - 스택 주소 지정 방식(Stack Addressing Mode)

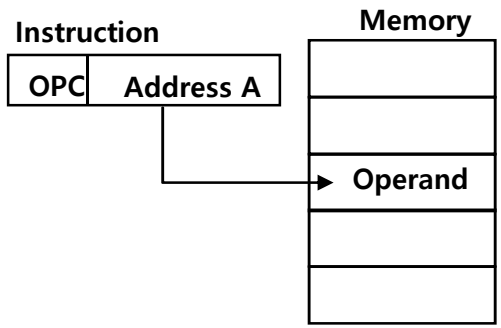
4.1 즉치 주소 지정 방식

- **즉치 주소 지정 방식(Immediate Addressing Mode)**
  - 명령어 실행에 필요한 데이터가 명령어 자체에 포함.
    - 상수(Constant)를 정의할 때 사용.
    - 변수의 초기 값 설정에 사용.
- **장점**
  - 메모리 접근이 필요하지 않기 때문에 오퍼랜드 접근이 빠름.
- **단점**
  - 사용할 수 있는 오퍼랜드 표현 범위가 오퍼랜드 필드의 크기로 제한됨.



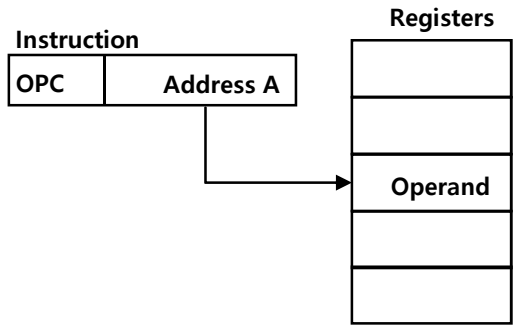
4.2 직접 주소 지정 방식

- **직접 주소 지정 방식(Direct Addressing Mode)**
  - 명령어의 오퍼랜드 필드에 오퍼랜드의 실제 주소 (Effective Address) 가 저장됨.
- **장점**
  - 오퍼랜드에 접근할 때 한 번의 메모리만 참조.
  - 실제 주소를 알아내기 위해 추가적 계산이 불필요.
- **단점**
  - 주소 필드에 표현할 수 있는 주소 공간이 제한적이다.



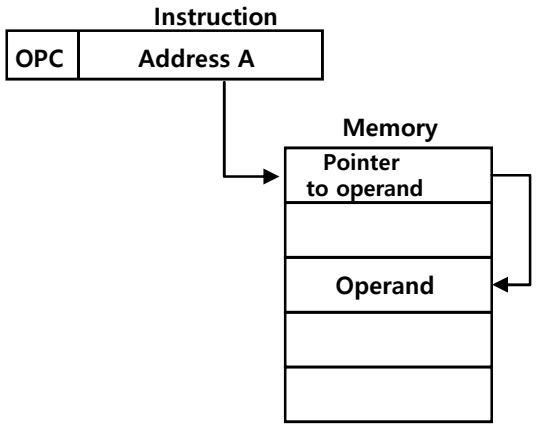
4.3 레지스터 주소 지정 방식

- **레지스터 주소 지정 방식(Register Addressing Mode)**
  - 오퍼랜드가 레지스터에 저장된 경우 사용.
  - 오퍼랜드 필드에 오퍼랜드를 저장하고 있는 레지스터 번호가 지정.
    - 실제 주소 = 레지스터 번호
- **장점**
  - 아주 작은 오퍼랜드 필드 사용 가능
    - 명령어 길이 단축
  - 메모리 접근이 불필요
    - 빠른 실행속도
- **단점**
  - 사용할 수 있는 주소 공간이 레지스터 수로 제한됨.



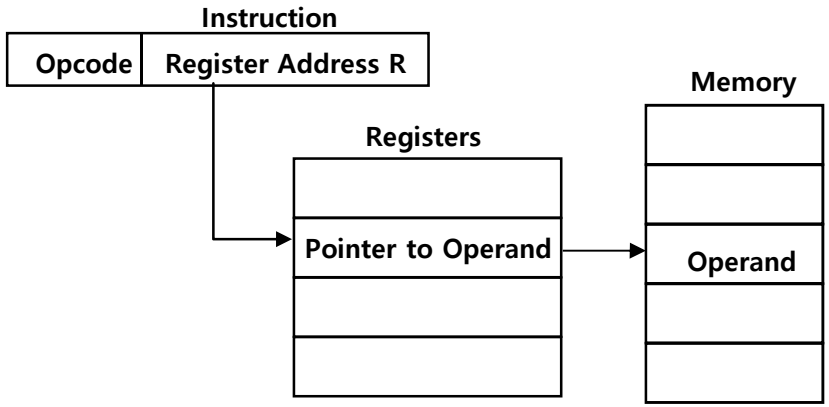
4.4 간접 주소 지정 방식

- **간접 주소 지정 방식(Indirect Addressing Mode)**
  - 오퍼랜드 필드가 가리키는 기억장소에 오퍼랜드가 저장된 주소 값이 저장되어 있다.
    - 실제 주소 = 오퍼랜드 필드가 지정하는 기억장소에 저장된 값
- **장점**
  - 넓은 주소 공간을 사용할 수 있다.
- **단점**
  - 오퍼랜드에 접근하기 위해 두 번의 메모리 접근이 필요.



4.5 레지스터 간접 주소 지정 방식

- **레지스터 간접 주소 지정 방식(Register Indirect Addressing Mode)**
  - 레지스터에 오퍼랜드가 저장된 메모리 주소 값이 저장.
    - 실제 주소 = 레지스터에 저장된 값
- **장점**
  - 넓은 주소 공간 사용가능
- **단점**
  - 오퍼랜드를 접근하기 위해 2번의 동작 (레지스터 접근 + 메모리 접근) 필요.
    - 간접 주소 방식보다 메모리 참조 횟수가 적다.

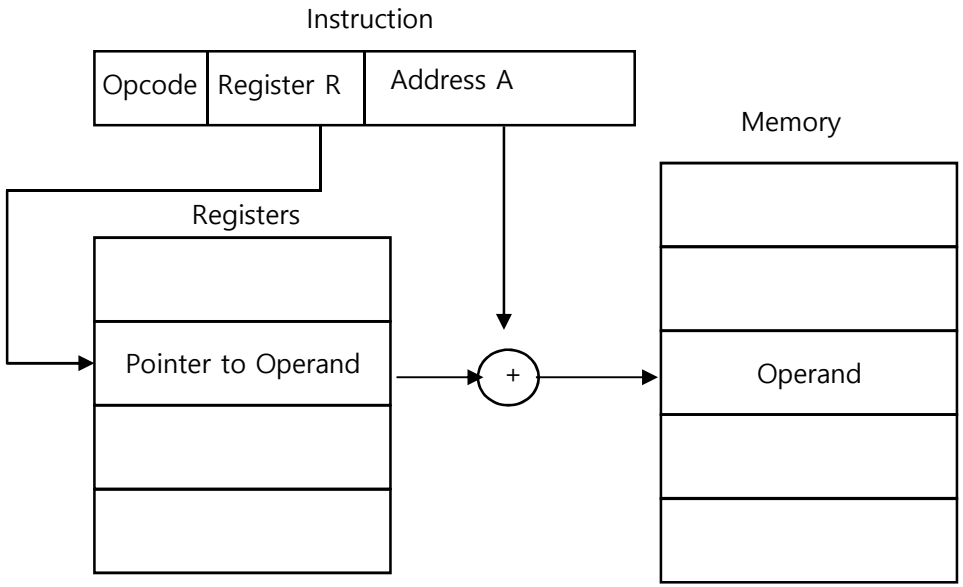


4.6 변위 주소 지정 방식 개요

- **변위 주소 지정 방식(Displacement Addressing Mode)**
  - 2 개의 오퍼랜드 필드를 사용 : 베이스 어드레스, 변위 어드레스
  - 실제 주소(Effective address) = 베이스 어드레스 + 변위 어드레스
- **베이스(Base) 어드레스**
  - 메모리 접근시 사용하는 기준 어드레스
  - 절대 값.
- **변위(Displacement) 어드레스**
  - 베이스 어드레스로 부터 떨어진 거리
    - 오프셋(Offset), 인덱스(Index)
  - 상대 값

4.6.1 변위 주소 지정 방식 장단점

- **변위 주소 지정 방식의 장단점**
  - 장점 : 넓은 주소 공간을 사용가능.
  - 단점 : 실제 주소 값 계산이 복잡.

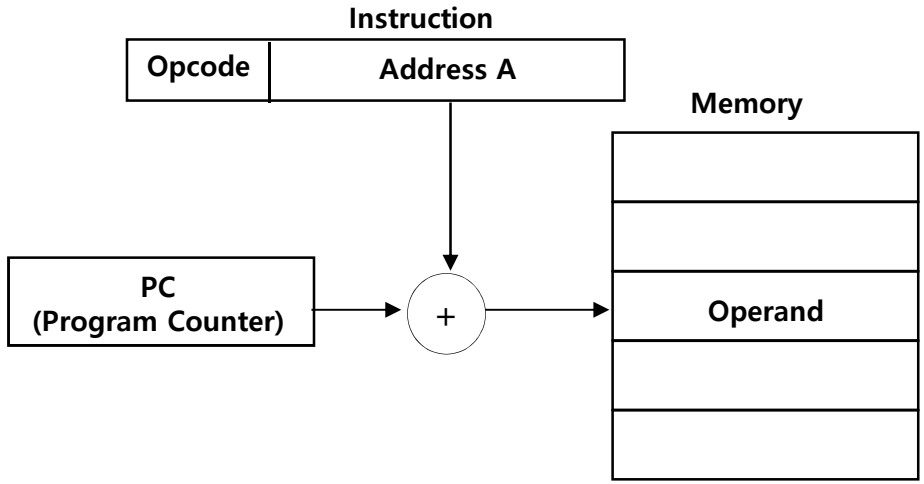


4.6.2 변위 주소 지정 방식 유형

- **대표적 변위 주소 지정 방식**
  - 베이스 어드레스를 저장하고 있는 레지스터에 따라 분류
    - 상대 주소 지정 방식(Relative Addressing Mode)
    - 베이스-레지스터 주소 지정 방식(Base-Register Addressing Mode)
    - 인덱싱 주소 지정 방식(Indexing Addressing Mode)

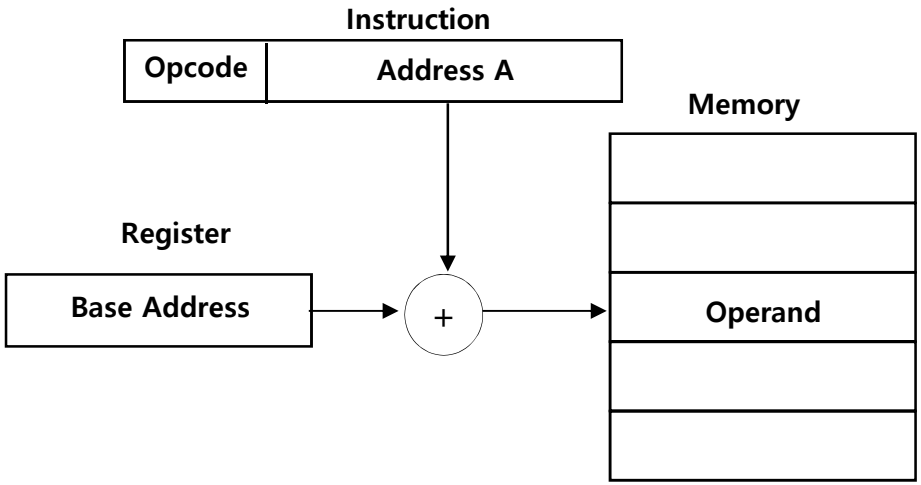
4.7. 상대 주소 지정 방식

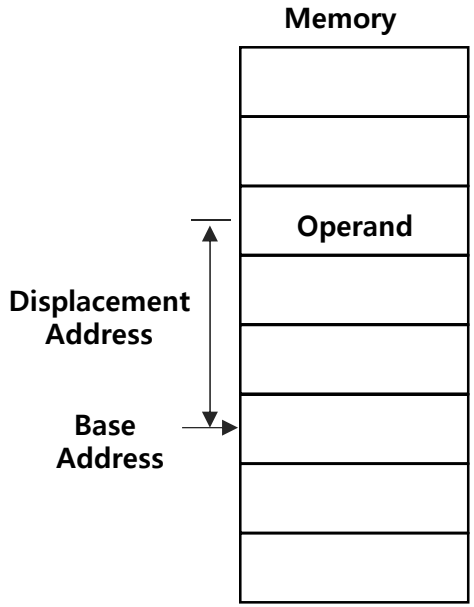
- **상대 주소 지정 방식(Relative Addressing Mode)**
  - 베이스 어드레스 : 프로그램 카운터
  - 변위 어드레스 : 명령어의 오퍼랜드 필드
  - PC-relative addressing
    - 분기(Branch) 명령어에서 사용.
    - 프로그램 메모리에 저장된 상수에 접근 가능.



4.8 Base-Register Addressing Mode

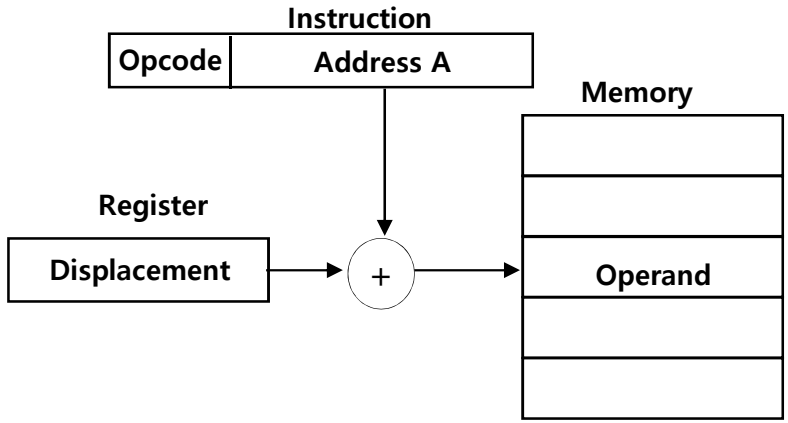
- **Base-Register Addressing Mode**
  - Base Address : Register
  - Displacement Address : 명령어의 오퍼랜드 필드
  - 배열(Array)형 데이터 접근에 유용.





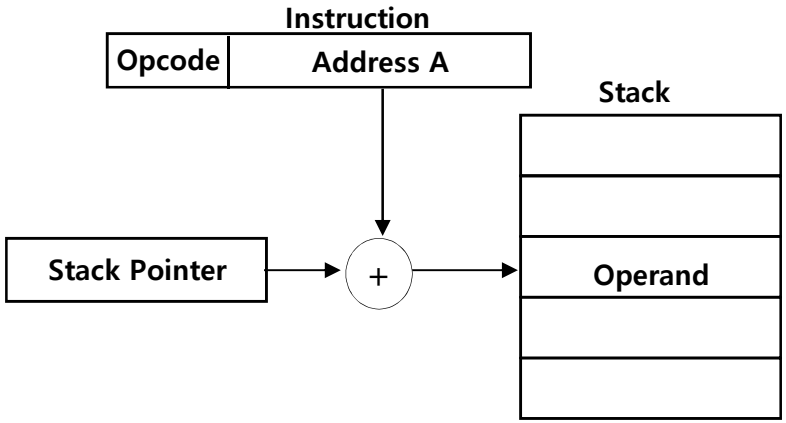
4.9 Indexing Addressing Mode

- **Base Address**
  - Operand Field of Instruction
- **Displacement Address**
  - Register
- **반복 연산에 효과적**
- **자동 인덱싱(Auto-Indexing)**
  - 인덱스 레지스터 값의 자동 증가/감소
    - 사전 인덱싱(Pre-Indexing)
      - 명령어 실행 전에 인덱스 레지스터 값 변경
    - 사후 인덱싱(Post-Indexing)
      - 명령어 실행 후, 인덱스 레지스터 값 변경



4.10 Stack Addressing Mode

- **메모리의 특정공간(스택)에 오퍼랜드를 저장하거나 참조할 때 사용**
  - 스택 전용 명령어 사용
  - 오퍼랜드가 스택의 최상위 위치에 저장.
- **묵시적 주소 지정 방식**
  - 베이스 어드레스 : 스택 포인터
  - 변위 어드레스 : 자동증가/감소, 명령어의 오퍼랜드 필드



어드레싱 모드 사례

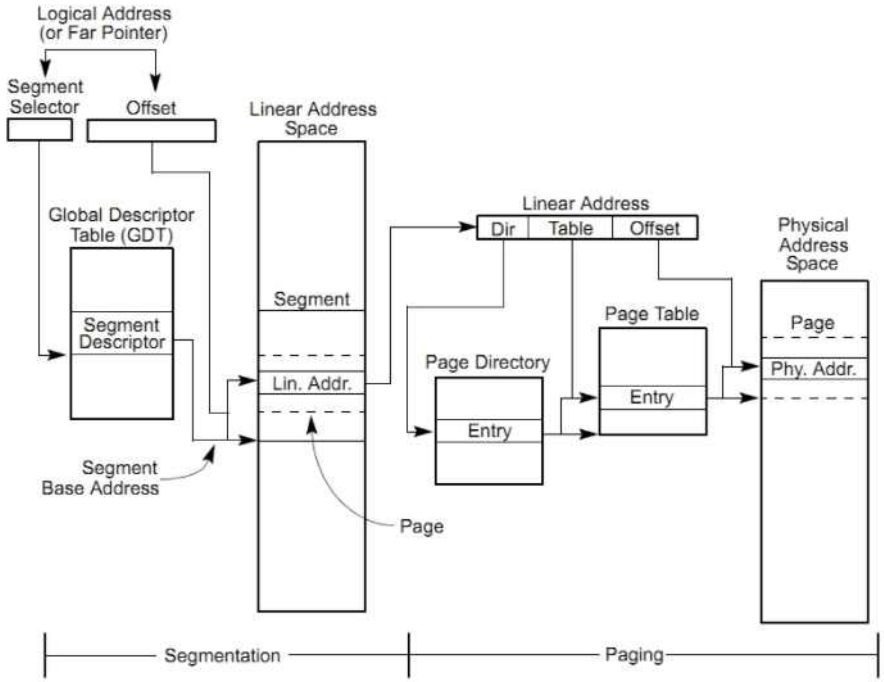
1. x86 프로세서의 주소 지정 방식

- **x86 프로세서에서의 어드레스 변환**
  - 메모리 관리 방법 : 세그먼트(Segmentation)와 페이지(Paging) 단위로 관리

- **x86 프로세서의 어드레스**
  - 논리 어드레스(Logical Address)
    - 명령어에서 사용하는 어드레스. 가상 어드레스
    - 실제 주소 = 세그먼트 시작주소 + 오프셋
  - 선형 어드레스(Linear Address)
    - 부호없는 32-비트 정수로 표현되는 주소.
    - 4GB 공간 표현(0x00000000 ~ 0xffffffff)
  - 물리 어드레스(Physical Address)
    - 물리 메모리에 접근할 때 사용되는 어드레스
    - 부호없는 32-비트 정수

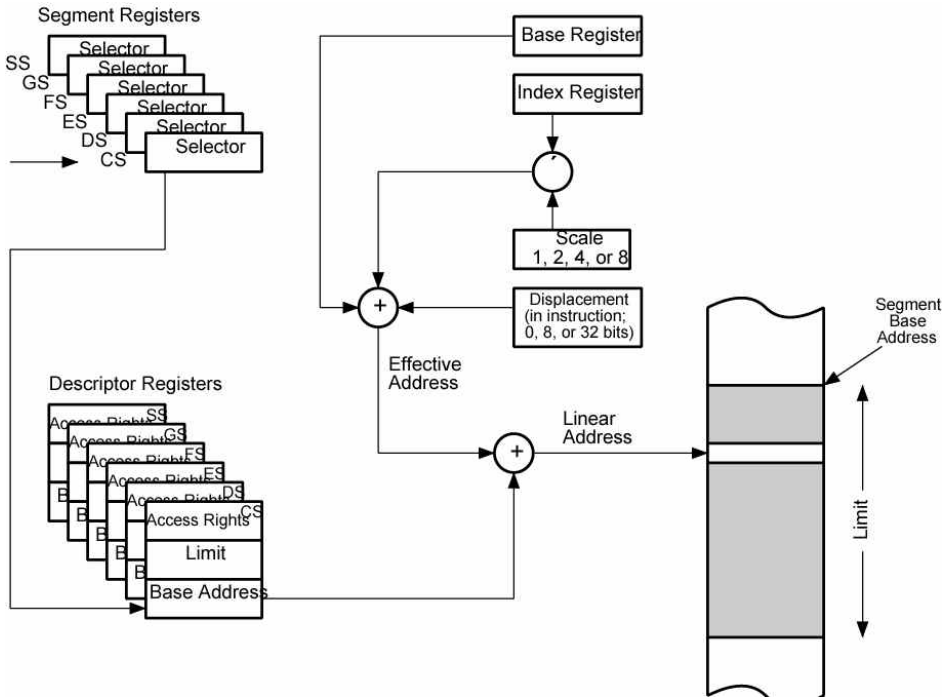
1.1 x86 프로세서에서의 어드레스 변환

- **x86 프로세서에서의 어드레스 변환**
  - 논리 어드레스의 물리 어드레스 변환
    - Segmentation Unit 과 Paging Unit 하드웨어를 사용하여 자동 변환



1.2 선형 어드레스 변환

Linear address = Segment address + Effective address
Effective address = Base + (Index * Scale) + Displacement

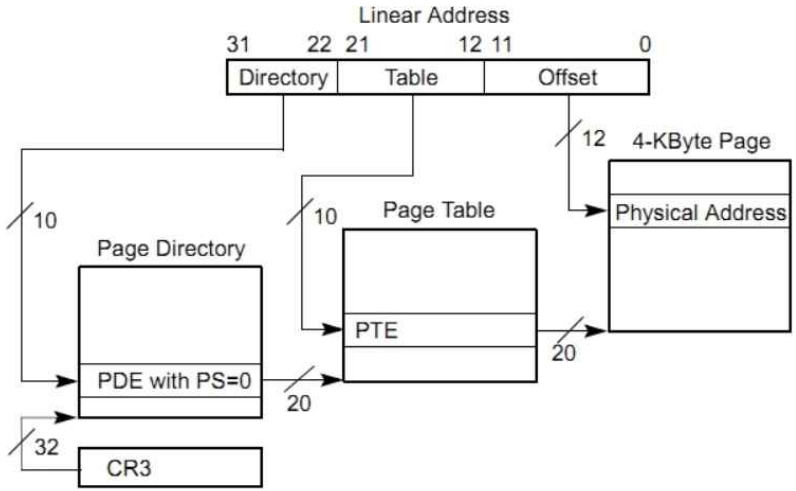


1.3 물리 어드레스 변환

- **페이지 테이블을 사용한 물리 어드레스 변환**
  - 페이지 디렉토리와 페이지 테이블 사용
  - 페이지 크기 : 4KB

- **선형 어드레스 구성**
  - 최상위 10-비트 : 페이지 디렉토리 인덱스
  - 하위 10-비트 : 페이지 테이블 인덱스
  - 최하위 12-비트 : 페이지내의 오프셋.

- **전체 페이지 수**
  - 1024 PDE * 1024 PTE = 220 pages



1.4 x86 프로세서의 주소 지정 방식

- **x86 어드레싱 모드**
  - Immediate
  - Register operand
  - Displacement
  - Base
  - Base with displacement
  - Scaled index with displacement
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

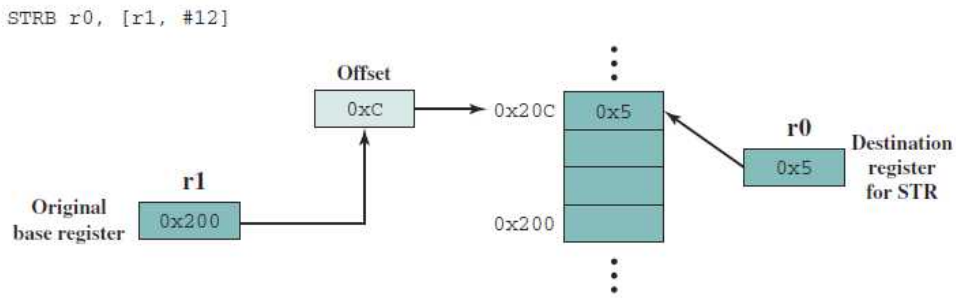
2. ARM 프로세서의 주소 지정 방식

- **ARM 프로세서 주소 지정 방식의 특징**
  - 비교적 간단한 주소 지정 방식을 사용하는 RISC 프로세서와 달리, 복잡한 방식을 사용.
  - Load/Store 구조를 사용하기 때문에 메모리 접근은 Load/Store 명령어만 가능.
  - 데이터 처리 명령어는 레지스터 오퍼랜드만 사용.

- **ARM 프로세서 어드레싱 모드 유형**
  - 명령어 유형에 따라 어드레싱 모드가 다름.
    - Load/Store 명령어 어드레싱
    - 데이터 처리 명령어 어드레싱
    - 분기 명령어 어드레싱
    - 다중 Load/Store 명령어 어드레싱

2.1 Load/Store 명령어 어드레싱

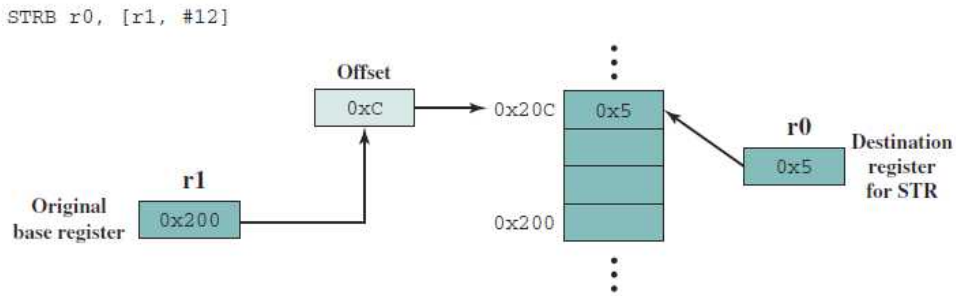
- **Load/Store 명령어의 주소 지정 방식**
  - Load/Store 명령어는 메모리에 접근할 수 있는 유일한 명령어
  - 변위 주소 지정 방식 사용
    - 베이스 레지스터 + Offset
- **베이스 레지스터 값**
  - Pre-Indexing/Post-Indexing 방식으로 변경 가능.
- **Offset 값**
  - 명령어에 포함되거나 다른 레지스터에 저장된 값을 사용.
  - 다른 레지스터에 저장된 값을 사용하는 경우 : Scaled register addressing 모드
    - Scaling factor는 명령어에 포함.



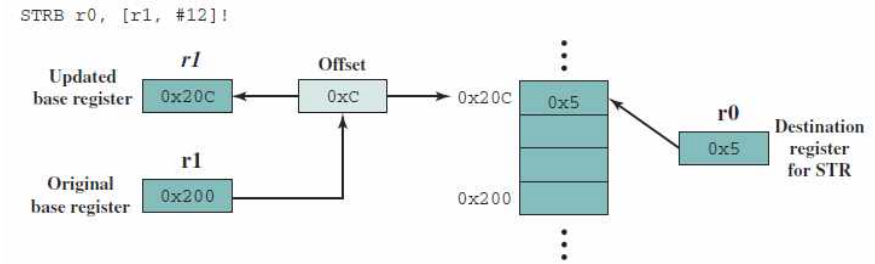
2.1 Load/Store 명령어 어드레싱

- **Load/Store 명령어의 어드레싱 예시**

Offset



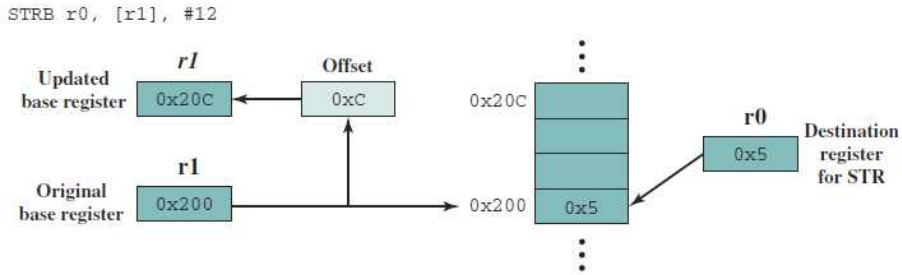
Pre-indexing



2.1 Load/Store 명령어 어드레싱

- **Load/Store 명령어의 어드레싱 예시**

Post-indexing



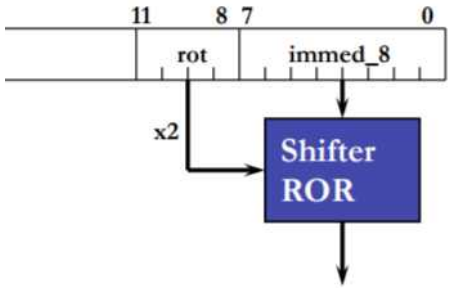
2.2 데이터 처리 명령어 어드레싱

- **데이터 처리 명령어의 주소 지정 방식**
  - 레지스터 주소 지정 방식
    - 레지스터에 저장된 값에 대한 Shift 연산을 사용한 Scaling 가능
  - 레지스터 주소 지정 방식과 즉치 주소 지정 방식을 혼합 사용

- **Scaling에 적용 가능한 shift 연산**
  - Logical shift left/right
  - Arithmetic shift right
  - Rotate right
  - Rotate right extended : Rotation with carry

2.2 데이터 처리 명령어 어드레싱

- **데이터 처리 명령어의 즉치 값 계산**
  - 명령어에서 12-비트를 사용하여 즉치 값 표현
    - 상위 4-비트 : Rotate right 크기(amount)
    - 하위 8-비트 : 실제 즉치 값
  - 실제 즉치 값
    - 하위 8-비트의 즉치 값을 상위 4-비트를 2배 한 양 만큼 rotate right 한 결과 값.
    - 모든 값을 다 표현 할 수 없다. 표현할 수 없는 값은 Literal Pool(lookup table) 사용.
- **명령어에 즉치 값 표현은 큰 부담**
  - ARM 프로세서는 일부만을 효과적으로 표현함으로써 명령어 길이 증가 부담과 표현 가능한 즉치 값 범위 축소 부담을 절충(Trade-off)




```
0xFF000000
MOV r0, #0xFF,8
```

Immed_8=0xFF, rot =4

2.3 분기 명령어 어드레싱

- **분기 명령어의 주소 지정 방식**
  - PC-Relative 주소 지정 방식
    - PC + 변위
    - 변위는 명령어에 24-비트로 포함.
  - 어드레스 계산
    - 변위를 2-비트 shift left : word 정렬
    - 24-bit signed 변위를 사용하기 때문에 PC로 부터 +/- 32 MB 변위까지 접근 가능

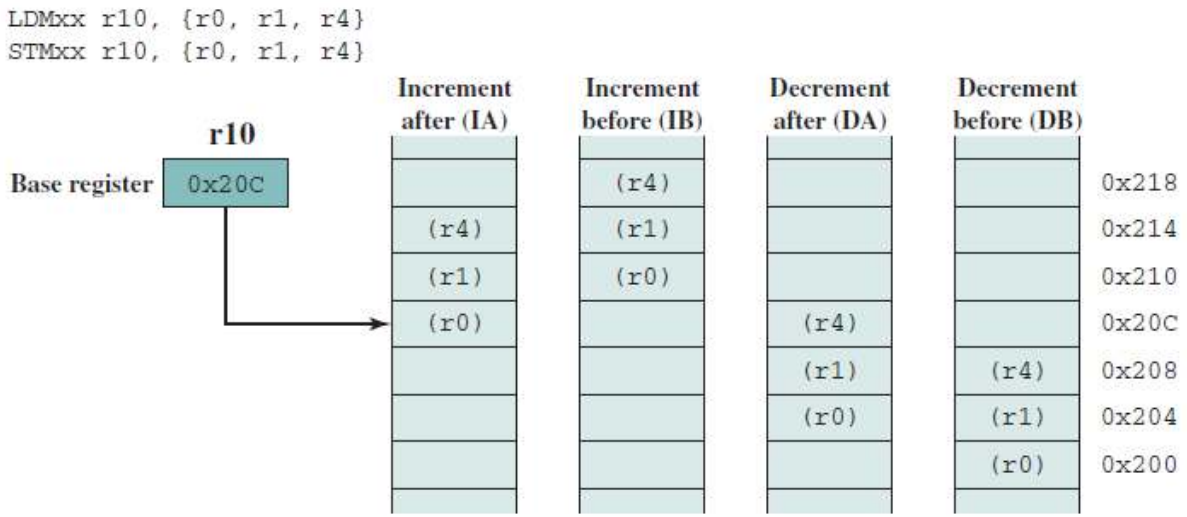
2.4 다중 Load/Store 명령어 어드레싱

- **Multiple Load/Store 명령어의 주소 지정 방식**
  - 여러 개 레지스터 내용을 메모리에 읽거나 쓰는 명령어에서 사용하는 주소 지정 방식
    - 최대 16 개의 레지스터를 한 개의 명령어로 Load/Store 가능
    - 16 개 레지스터 지정은 명령어의 주소 필드(16-비트)에 비트 단위로 매핑
  - 메모리의 시작 주소는 베이스 레지스터에 저장.
    - 4개의 주소 지정 모드 사용
      - Increment After(IA), Increment Before(IB), Decrement After(DA), Decrement Before(DB)
    - 메모리 주소의 자동 증가/감소는 첫번째 메모리 접근 발생시 자동 실행.
  - 레지스터 번호와 메모리 주소 매핑
    - 레지스터 번호가 낮으면 낮은 메모리 주소에 매핑

- **블록 단위 데이터 Load/Store, 스택 PUSH/POP, 함수 호출/귀환 작업시 유용.**

2.4 다중 Load/Store 명령어 어드레싱

- 베이스 레지스터가 R10 인 경우.



end