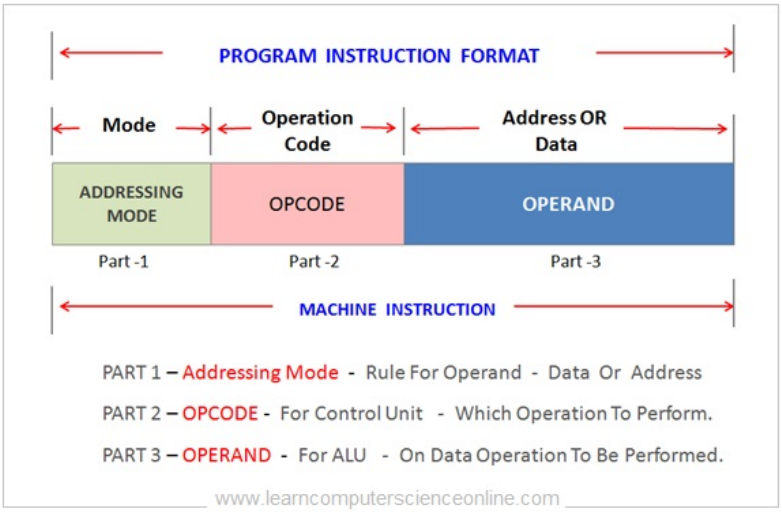
Computer architecture

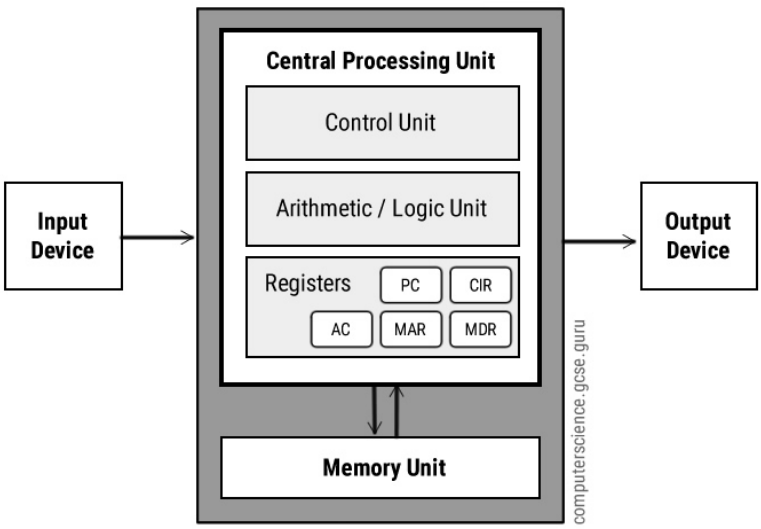
Instruction

instruction = operation code + operand

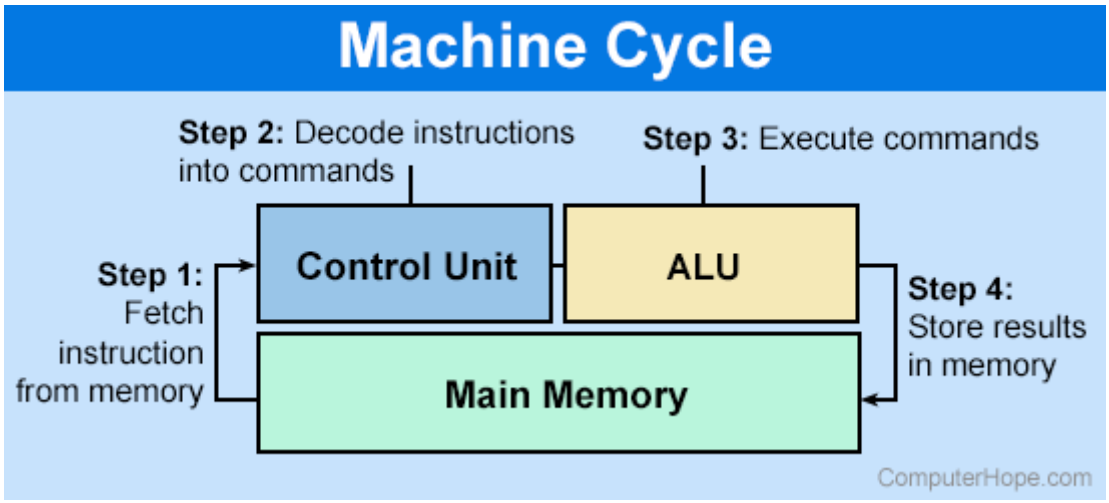


1. operation code 종류
    1. 데이터 전송
    2. Arithmetic/Boolean operation
    3. Flow control
    4. I/O control
2. operand
    1. 데이터 자체보다는 데이터가 저장된 memory or register address가 저장됨

CPU의 작동원리



ALU & control unit



Addressing mode

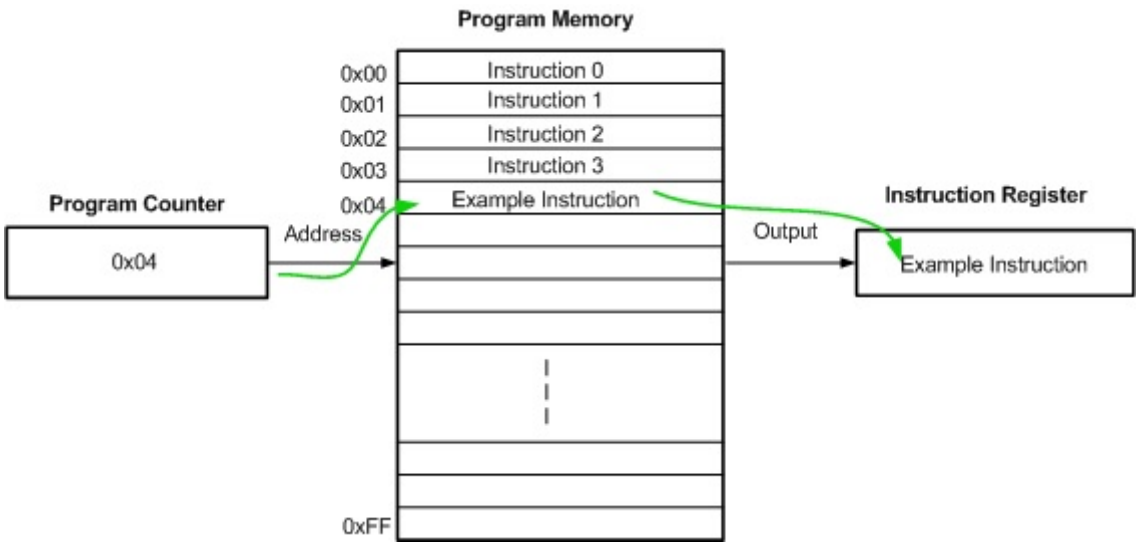
operation에 사용할 operand의 위치를 찾는 방식

1. immediate addressing mode : 연산에 사용할 operand 값을 operand field에 직접 명시
2. direct addressing mode : operand field에 effective address (데이터가 저장된 위치)
3. indirect addressing mode : effective address의 address 저장
4. register addressing mode : data를 저장한 register를 명시
5. register indirect addressing mode : 데이터는 memory에, address는 register에
6. displacement addressing mode : operand field의 값과 특정 register의 값을 더하여 effective address를 얻어냄
    1. releative addressing mode : operand + PC -> effective address
    2. base-register addressing mode -> operand + base register

Register

CPU 내에 있는 작은 임시 저장 장치

1. Program counter
    - memory에서 가져올 instruction의 address 저장
    - PC가 계속 증가하면서(0x00 -> 0x01 -> 0x02 -> 0x03 ...) instruction이 실행
2. Instruction register : memory에서 읽어들인 instruction 저장

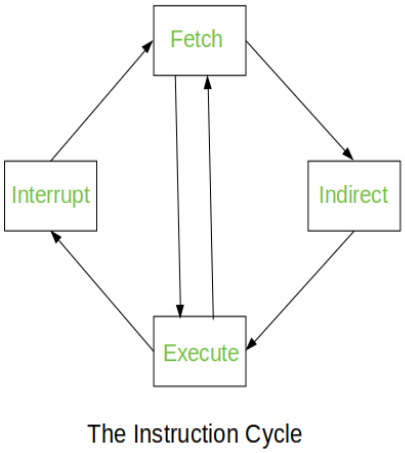


3. MAR(Memory Address Register)
- memory의 address 저장
- CPU가 읽어 들이고자 하는 주소 값을 주소 버스로 보낼 때 MAR 거침
4. MBR(Memory Buffer Register)
- 메모리와 주고받을 값(데이터와 instruction)을 저장
5. General purpose register
6. Flag register
- ALU 연산 결과에 따른 flag를 저장
7. Stack pointer
- stack의 꼭대기를 가리키는 register
- 즉 stack에 가장 마지막으로 저장된 값의 위치를 저장

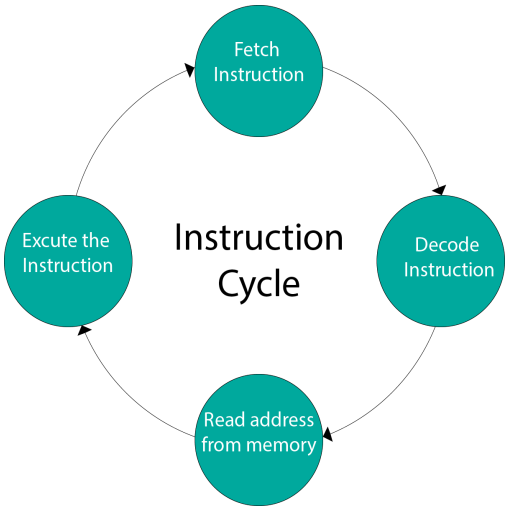
- stack은 메모리 안에 stack 영역에 있음
8. Base register
- Base-register

Instruction cycle

- 하나의 instruction을 처리하는 정형화된 흐름
- 이 흐름이 끊기는 것을 **interrupt**라고 함
- 1 cycle = fetch + execute (or fetch+decode+excute)
- fetch -> memory에 있는 instruction을 가져옴
- decode -> instruction
- execute -> 가져온 instruction을 실행함



The Instruction Cycle



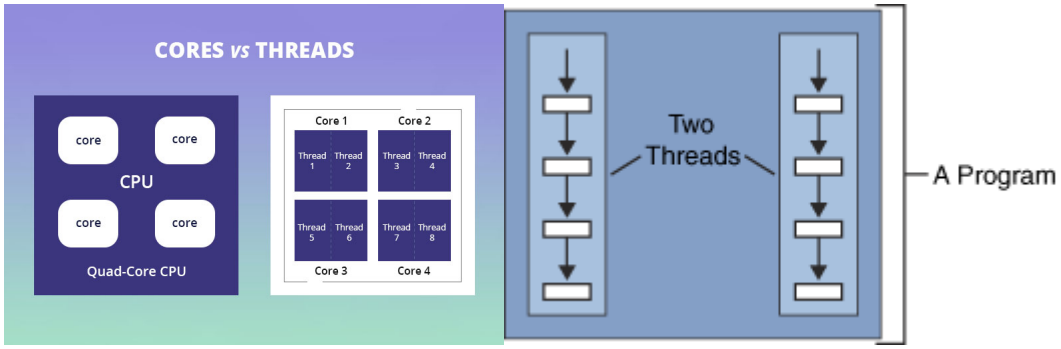
1. **Interrupt**
   - synchronous interrupt (exception)
     - CPU에 의해 발생함
     - instruction을 수행하다가 예상치 못한 오류 등을 마주쳤을 때 발생
   - asynchronous interrupt (hardware interrupt)
     - 주로 I/O 장치에 의해 발생
     - ex) 프린터가 프린트 완료되면 CPU에 interrupt 알림
     - 그러면 CPU는 프린터의 상태를 주기적으로 확인할 필요 없어서 효율적임
     - 처리순서
       (1) I/O send interrupt request to CPU
       (2) CPU checks interrupts at the end of execute cycles
       (3) CPU checks the interrupt request and checks the interrupt flag to see if the current interrupt is acceptable
       - flag 활성화 -> interrupt request 수용 가능
       - non maskable interrupt는 flag 활성화 안돼있어도 수용해야함
         (4) If acceptable, backup!
         (5) CPU refers to interrupt vector to execute interrupt service routine
       - interrupt service routine : 말 그대로 interrupt별로 어떻게 처리할 지가 규정된 프로그램
       - interrupt vector : interrupt service routine을 식별하기 위한 정보 (일종의 address)
         (6) After the interrupt service routine runs, recover the jobs that backed up
   - 전체 흐름을 요약하면 (1) interrupt 요청이 오면 잠시 작업을 중단하고 (2) interrupt service routine을 실행한 후에 다시 원래 작업으로 돌아온다
   - 원래 하던 작업에 대한 정보는 *stack* 에 저장되어 있다

CPU 성능 향상 기법

Clock, core, thread

1. Clock
   - 컴퓨터의 모든 부품들은 clock 신호에 맞춰 움직임
   - clock 신호가 빠르게 반복될수록, 전체적인 실행속도가 빨라짐
   - 보통 GHz 단위 (10억 Hz)
2. Core
   - 명령어를 실행하는 부품
   - 즉 일반적으로 생각하는 CPU
3. Thread

- process 내의 실행 흐름의 단위
- hardware thread vs software thread
- user-level thread vs kernel-level thread (thread 지원 주체별)
  (1) Hardware thread
  - 하나의 core가 동시에 처리하는 instruction 단위
  - 동시에 2개 이상의 instruction을 실행할 수 있으면 multithread라고 함
  - hyper threading -> intel의 multithreading 기술
  - 아래 예시는 4 core 8 thread CPU



(2) software thread
- 하나의 프로그램에서 독립적으로 실행되는 단위
- ex) word는 사용자로부터 입력받은 내용을 화면에 보여주는 기능, 사용자가 입력한 내용이 맞춤법에 맞는지 검사하는 기능, 사용자가 입력한 내용을 수시로 저장하는 기능 등이 있는데 이는 모두 각각 software thread라고 할 수 있음

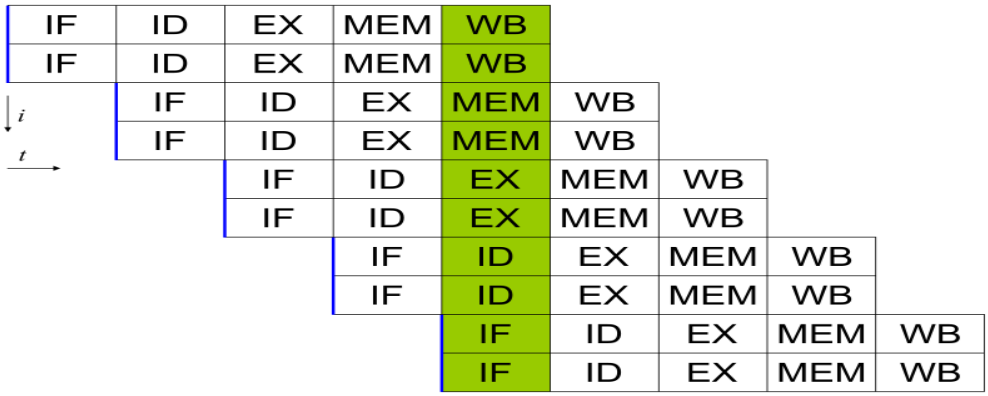
4. Multi-thread processor
   - multi thread의 핵심은 register
   - ex) PC가 2개 있으면 메모리에서 가져올 instruction의 address를 동시에 2개 지정 가능
   - logical processor : 2 core 4 thread의 경우 실제 core는 2개지만 thread가 하나의 instruction을 실행할 수 있으므로 프로그램 입장에서 봤을 땐 core가 4개처럼 보임. 즉 physical core는 2개이나 logical core는 4개

ILP (Instruction-Level Parallelism)

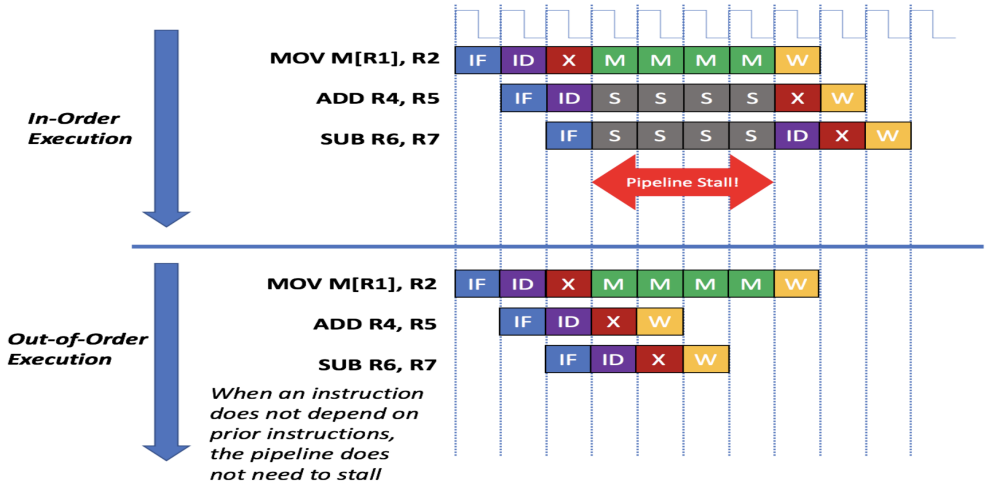
1. Instruction pipeline
   - Fetch => Decode => Execute => Write back
   - 같은 단계가 겹치지만 않는다면, 각 단계를 동시에 실행 가능
   - pipeline은 일반적으로 성능을 향상시키지만 그렇지 않은 경우가 있음 *(pipeline hazard)*
   - pipeline hazard
     - data hazard => data dependence에 의해 발생
     - control hazard => 주로 branch에 의해 발생. 10번지에서 실행 중이다가 갑자기 60번지로 가면, PC에 의해 자연스럽게 실행 예정이었던 11, 12, ... 얘네들은 쓸모가 없어짐
     - structural hazard => 서로 다른 instruction이 동시에 ALU, register를 사용

	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6
Instruction 1	Fetch	Decode	Execute			
Instruction 2		Fetch	Decode	Execute		
Instruction 3			Fetch	Decode	Execute	
Instruction 4				Fetch	Decode	Execute

2. Super scalar => pipeline을 동시에 여러개 실행



3. OoOE (Out-of-Order Execution)
- instruction을 순서대로 하지 않고(out-of-order) dependence 문제가 발생하지 않도록 순서를 조정해서 pipeline의 in-order 실행보다 더 빠르게 병렬화



CISC & RISC

- pipeline을 적용하려면, instruction이 pipeline 실행에 최적화가 되어 있어야 함
- instruction set : CPU가 이해할 수 있는 instruction의 집합 (or ISA : Instruction Set Architecture)
- ISA가 다르면 assembly 언어도 달라짐

1. CISC (Complex Instruction Set Computer)
   - 다양하고 여러 기능을 수행할 수 있는 가변 길이 instruction을 사용
   - 그만큼 적은 수의 instruction으로 다양한 프로그램을 실행할 수 있지만
   - instruction별로 실행시간이 달라 pipeline에 적합하지 않음
   - pipeline을 만드려면 instruction별로 clock cycle이 비슷해야 하니까!
2. RISC (Reduced Instruction Set Computer)
   - CISC와 다르게 짧고, 규격화되고, 되도록 1클럭 내외로 실행되는 instruction
   - 고정 길이 instruction을 활용함
   - memory에 직접 접근하는 instruction은 load, store 2개 밖에 없음
   - memory 접근을 단순화하는 대신 register를 적극적으로 활용

Memory & Cache

RAM

- volatile memory
- RAM이 클수록 프로그램을 disk에서 많이 가져다 놓을 수 있으므로 속도가 빠를 수 있음
- RAM 종류

1. DRAM (Dynamic RAM) : 시간이 지나면 저장된 데이터가 점차 사라짐. 데이터의 소멸을 막이 위해 일정 주기로 데이터를 재활성화(다시 저장)해야 함. 이런 단점이 있지만 소비 전력이 낮고, 저렴하고, 집적도가 높아 대용량으로 설계하기 유리해 대부분 RAM은 DRAM
2. SRAM (Static RAM) : 저장된 데이터가 변하지 않음. DRAM과 달리 소비 전력이 크고, 비싸고, 집적도가 낮아 소용량/빠른 속도를 요구하는 *캐시메모리* 에서 사용
3. SDRAM (Synchronous Dynamic RAM) : clock 신호와 동기화된 DRAM
4. DDR SDRAM (Double Data Rate SDRAM) : data rate(대역폭)을 늘려 속도를 빠르게 만든 SDRAM. 흔히 사용되는 DDR4 SDRAM은 SDR(Single Date Rate SDRAM)보다 16배 넓은(2^4) data rate

Address space

- physical address + logical address

1. physical address
   - 정보가 실제로 저장된 하드웨어상의 address
2. logical address
   - 실행중인 프로그램 각각에게 부여된 0번지부터 시작되는 address
   - CPU가 이해하는 address
   - 따라서 physical address를 logical로 변환해주는 작업 필요 (by **MMU**)
3. MMU (Memory Management Unit)
   - CPU가 발생시킨 logical 주소에 base register값을 더해 physical로 변환
   - base register는 프로그램의 가장 작은 physical address
   - ex) base register = 4000, logical = 100 -> physical = 4100
   - limit register는 logical이 physical 범위를 넘어서지 않도록 조절

Cache memory

- CPU와 memory 사이에 위치
- register보다 용량이 크고 memoryq보다 빠른 SRAM 기반의 저장 장치
- CPU와 가까운 순서대로 L1, L2, L3 cache / L1,L2는 core 안에, L3는 밖에 보통 위치
- multicore 환경에서, 각 코어마다 L1, L2 cache를 가지고, L3는 core간에 공유하는 형태
- ***principle of locality***
  - cache hit : cache 내 데이터가 CPU에서 활용되는 경우
  - cache miss : memory에서 데이터를 가져와야 하는 경우
  - cache hit ratio = (cache hit) / (cache hit + miss)
  - cache hit를 높이기 위한 전략이 **principle of locality**
  - temporal locality : CPU는 최근에 접근했던 memory space에 다시 접근
  - spatial locality : 접근한 memory space 근처를 접근 하는 경향