Computer Architecture

컴퓨터개론

(Introduction to Computer Systems)

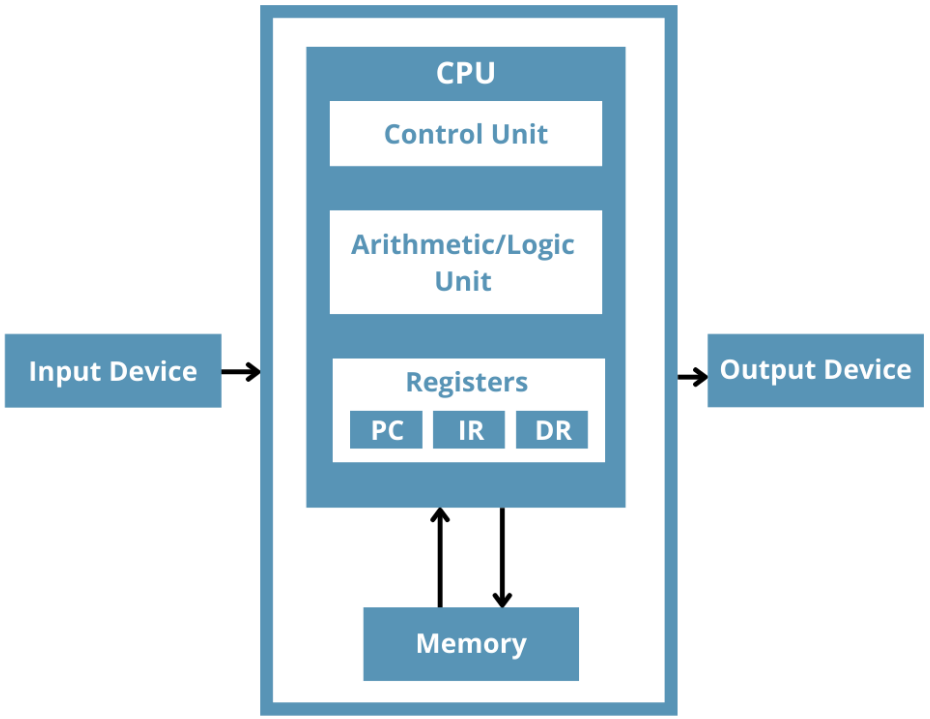
GEN1030

CPU

Central Processing Unit (CPU)

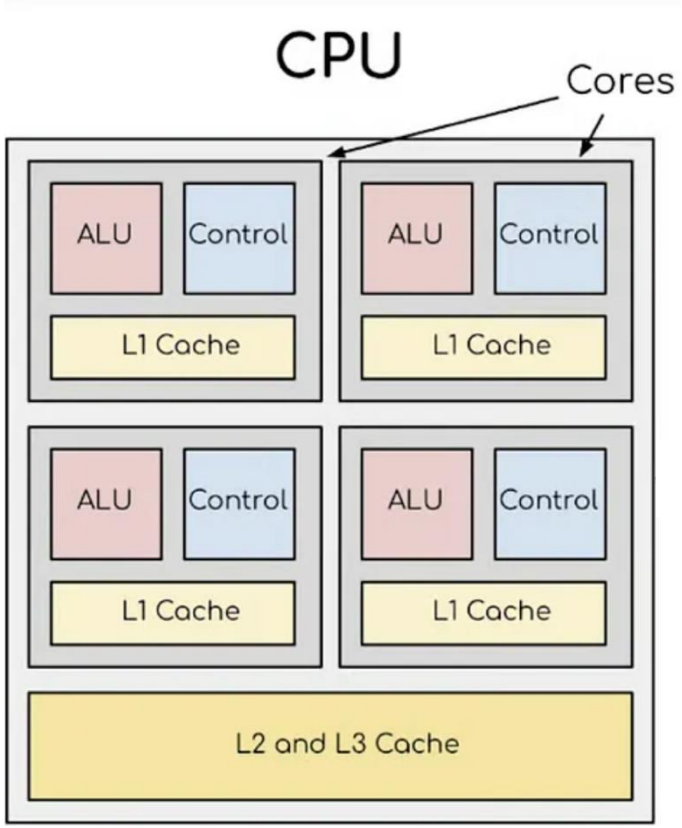
- 메모리에 저장된 프로그램, 데이터를 이용하여 실제 작업을 수행하는 장치

- 연산장치(ALU)
  - 자료의 연산 수행
- 제어장치(Control Unit)
  - 컴퓨터의 작동 제어
- 레지스터(Register)
  - 연산에 사용되는 자료 임시 저장



Central Processing Unit (CPU)

- 현대 CPU는 여러 개의 코어(Core)로 구성 됨
  - 각 코어는 독립적으로 명령어 실행할 수 있는 작은 CPU라고 보면 됨

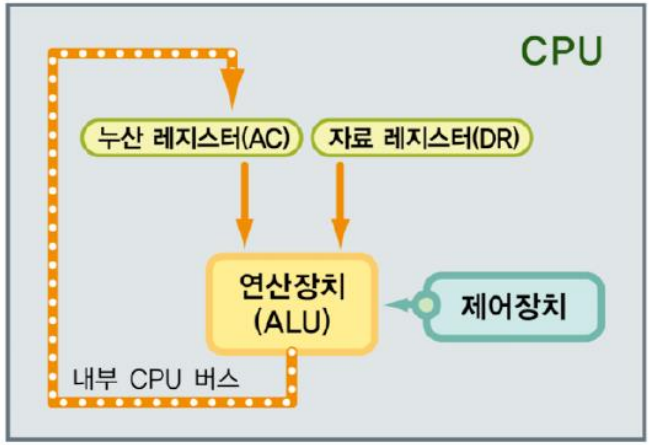


연산장치

- Arithmetic and Logic Unit (ALU)
  - 산술 연산 모듈(Arithmetic module): 산술 연산(ex. 덧셈, 뺄셈) 수행
  - 논리 연산 모듈(Logic module): 논리 연산(ex. NOT, AND, OR) 수행

- 레지스터
  - CPU 내부에 있는 고속 메모리
  - 연산장치에서 사용하는 피연산자는 보통 1개 또는 2개
  - CPU 내부의 누산 레지스터(Accumulator) 및 자료 레지스터(Data register)에 저장된 자료를 피연산자로 사용
  - 연산 결과는 다시 누산 레지스터에 저장

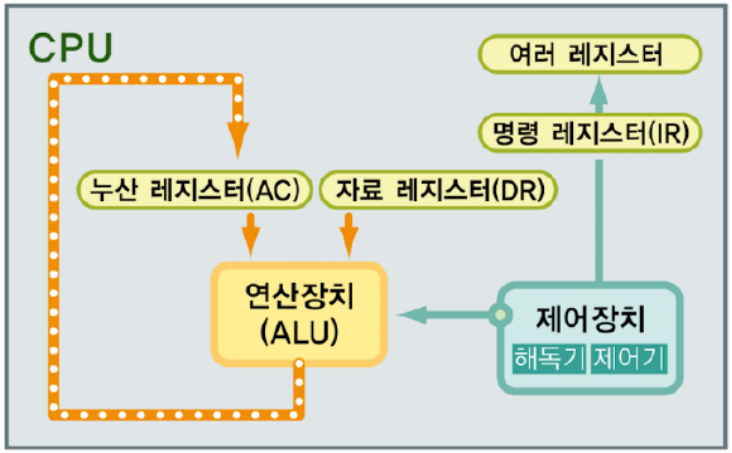
Accumulator ← Accumulator
+ Data register



제어장치

- Control Unit
  - 산술 및 논리 연산에 요구되는 작업을 수행하는 신호를 보내 연산장치와 레지스터가 명령을 수행하게 하는 장치
  - 해독기(Decoder)와 제어기 등으로 구성 됨
  - 명령어의 실행에 필요한 연산 순서, 종류 등을 종합적으로 제어

  - 실행할 명령어를 해석
  - 해석 결과에 따라 제어 신호 생성하고 각 구성 요소 동작 제어



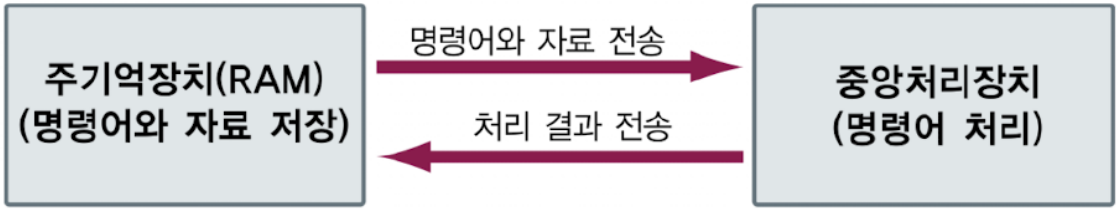
레지스터(Register)

- CPU 내부처리 연산에 사용되는 임시 기억장소
- 여러 레지스터 존재
  - 참조 속도가 매우 빠르고 가격이 비쌈 → 수가 제한적

레지스터 심볼	레지스터 이름	기능
DR	자료 레지스터 (Data Register)	연산에 필요한 피연산자를 저장하는 레지스터
AR	주소 레지스터 (Address Register)	현재 접근할 기억장소의 주소를 기억하는 레지스터
AC	누산 레지스터 (Accumulator Register)	연산장치의 입출력 데이터를 임시적으로 기억하는 레지스터
IR	명령어 레지스터 (Instruction Register)	현재 수행 중인 명령어를 저장하고 있는 레지스터
PC	프로그램 카운터 (Program Counter)	다음에 실행할 명령어의 메모리 주소가 저장된 레지스터
TR	임시 레지스터 (Temporary Register)	임시로 자료를 저장하는 레지스터로 범용 레지스터라고도 부름

명령어 처리 과정

- CPU와 주기억장치 사이 데이터 전송



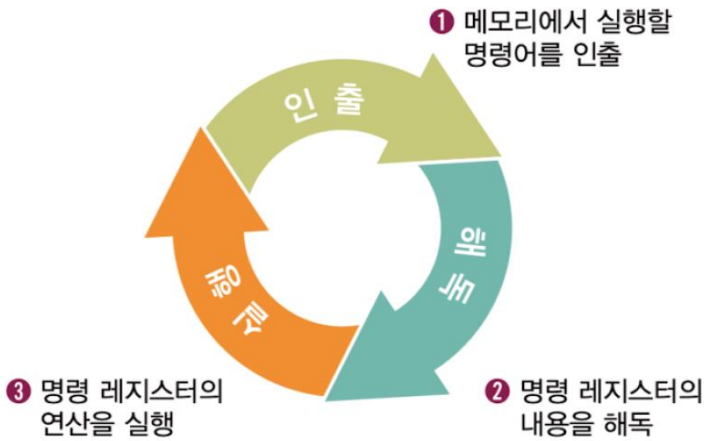
- 처리 과정
  1. 주기억장치에 저장 된 명령어 및 데이터가 CPU의 레지스터로 전송
  2. 명령어 처리
  3. 처리 결과 다시 주기억장치로 전송

CPU Machine Cycle

- CPU는 하나의 명령어 실행을 위해 아래의 과정을 거침
  - 인출(Fetch)
  - 해독(Decode)
  - 실행(Execution)

  → 기계 주기(Machine cycle or Instruction cycle)

- 프로그램 실행?
  - 프로그램 = 명령어의 집합
  - 위 cycle을 계속 반복

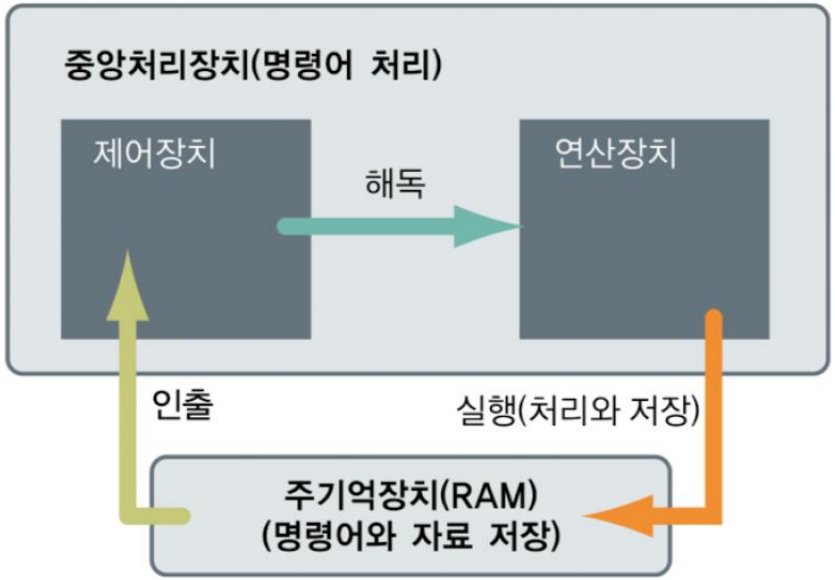


CPU Machine Cycle

- 인출(Fetch) 단계
  - 제어장치가 프로그램 카운터에 있는 주소로 다음에 수행할 명령어를 명령 레지스터에 저장
  - 다음 명령어를 수행하기 위해 프로그램 카운터 +1
- 해독(Decode) 단계
  - 제어장치는 명령어 레지스터에 있는 명령어를 연산 부분, 피연산 부분으로 해독
  - 피연산 부분이 있는 경우 피연산 부분의 메모리 주소를 레지스터에 저장
- 실행(Execution) 단계
  - 실제 명령을 수행 (CPU 지시에 따라)
  - ex) 산술 연산이 필요한 작업이면 ALU 회로를 작동

CPU Machine Cycle

- 하나의 명령어 실행 종료되면 프로그램 카운터가 가리키는 다음 명령어에 대해 기계 주기 반복
  - Fetch-Decode-Execute cycle



프로그램 실행 과정

두 정수의 합

- 32 + (-18) = ?
  - A = 32, B = -18

C ← A + B

- 여러 명령어 집합으로 구성하여 실행

```
LDA  A
ADD  B
STA  C
HLT
```

- **LDA A**: 메모리 A의 내용을 누산 레지스터(AC)에 저장
- **ADD B**: 메모리 B의 내용과 누산 레지스터의 값을 더하여 누산 레지스터에 다시 저장
- **STA C**: 누산 레지스터의 값을 메모리 C에 저장
- **HLT**: 프로그램 종료

두 정수의 합

- 명령어가 처리 되기 위해 실제 발생하는 데이터 이동은 여러 단계일 수 있음

- **명령어 LDA 0012FF40**
  - 메모리의 내용을 누산 레지스터(AC)에 저장

  0. AR ← 0012FF40
     - 주소 레지스터(AR)에 0012FF40 주소 저장
  1. DR ← M[AR]
     - 주소 레지스터(AR)의 주소 값을 갖는 메모리 자료(M[AR])를 자료 레지스터(DR)에 저장
  2. AC ← DR
     - 처리를 위해 자료 레지스터(DR)를 다시 누산 레지스터(AC)에 저장

두 정수의 합

- 명령어가 처리 되기 위해 실제 발생하는 데이터 이동은 여러 단계일 수 있음

- **명령어 ADD 0012FF44**
  - 메모리의 내용과 누산 레지스터의 값을 더하여 누산 레지스터에 다시 저장

1. DR ← M[AR]
   - 주소 레지스터(AR)의 주소 값을 갖는 위치의 메모리 데이터(M[AR])를 자료 레지스터(DR)에 저장
2. AC ← AC + DR
   - 누산 레지스터(AC)와 자료 레지스터를 더하고 그 결과를 다시 누산 레지스터에 저장

두 정수의 합

- 명령어가 처리 되기 위해 실제 발생하는 데이터 이동은 여러 단계일 수 있음

- **명령어 STA 0012FF48**
  - 누산 레지스터의 값을 메모리에 저장

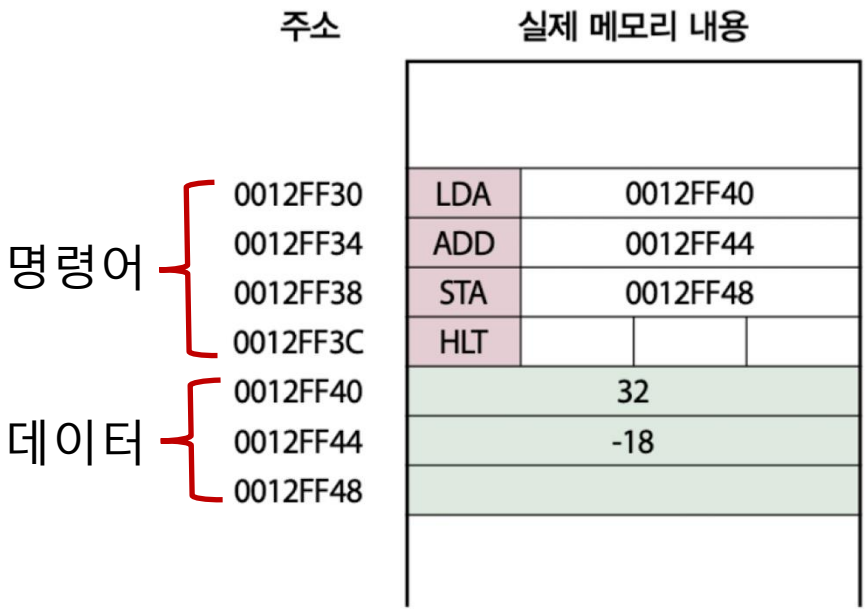
  1. M[AR] ← AC
     - 누산 레지스터(AC)의 값을 주소 레지스터(AR)에 들어있는 메모리 위치(M[AR])에 저장

메모리 공간 예시

- 메모리 저장 예시

32 + (-18)　　**C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```



A (32): 메모리 주소 0012FF40

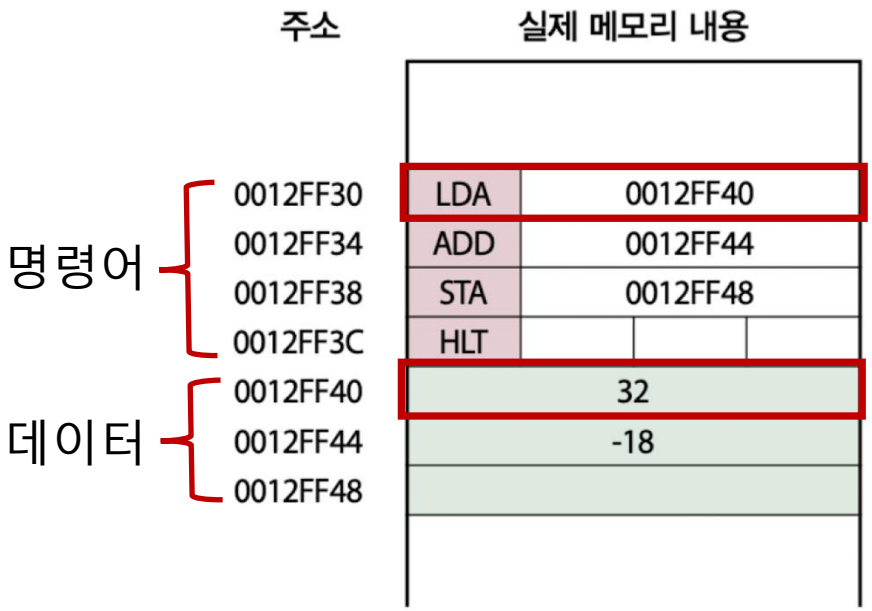
B (-18): 메모리 주소 0012FF44

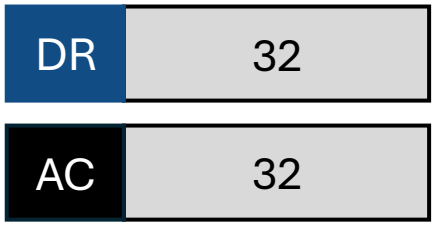
메모리 공간 예시

- 메모리 저장 예시

32 + (-18) **C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```





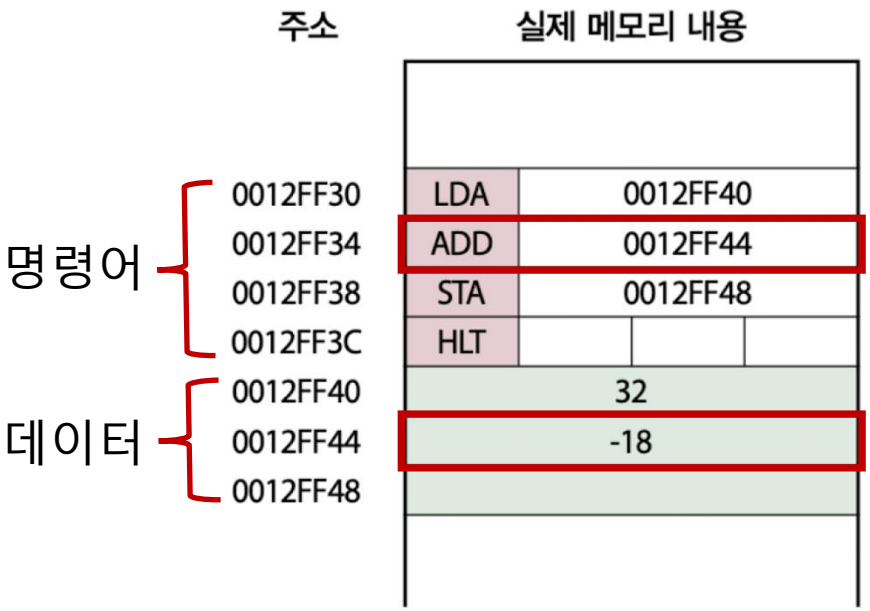
메모리의 내용을
누산 레지스터(AC)에 저장
DR ← M[AR]
AC ← DR

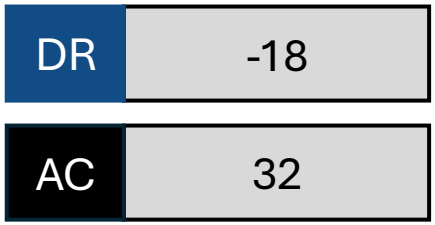
메모리 공간 예시

- 메모리 저장 예시

32 + (-18) **C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```





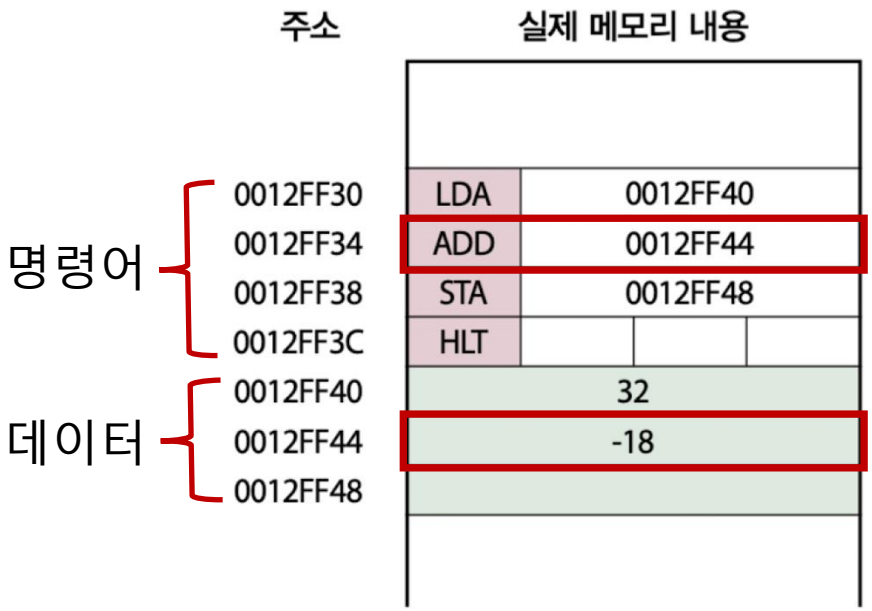
메모리의 내용과
누산 레지스터의 값을 더하고
누산 레지스터에 다시 저장
DR ← M[AR]
AC ← AC + DR

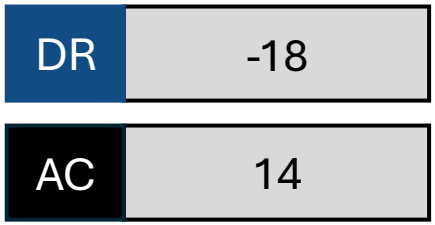
메모리 공간 예시

- 메모리 저장 예시

32 + (-18) **C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```





메모리의 내용과
누산 레지스터의 값을 더하고
누산 레지스터에 다시 저장

DR ← M[AR]

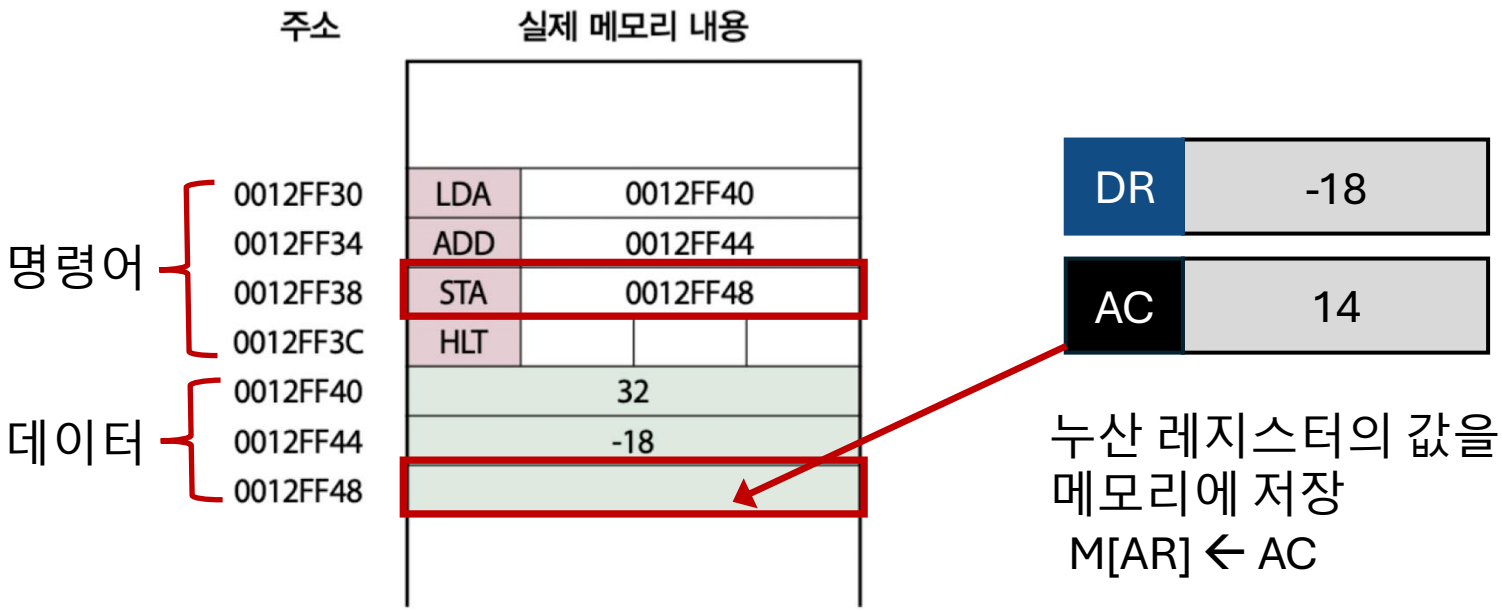
AC ← AC + DR

메모리 공간 예시

- 메모리 저장 예시

32 + (-18) **C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```

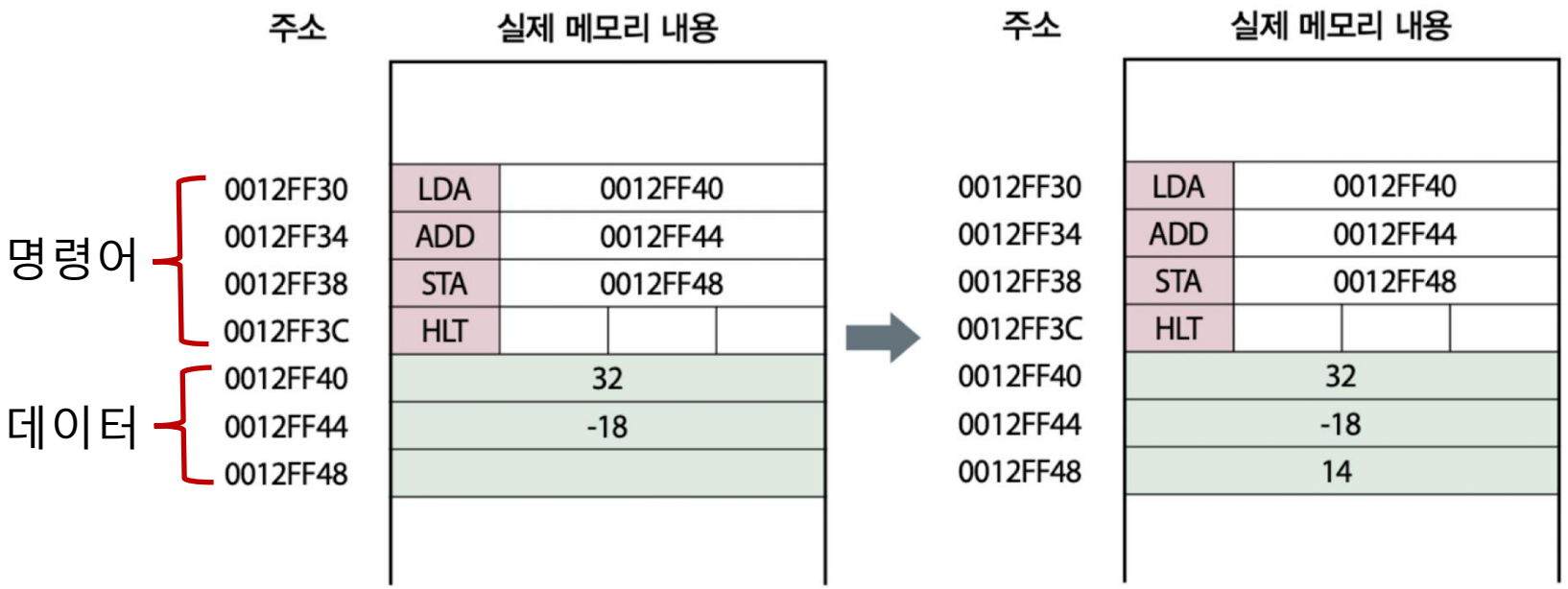


메모리 공간 예시

- 메모리 저장 예시

32 + (-18)　　**C ← A + B**

```
LDA  A
ADD  B
STA  C
HLT
```



CPU 성능

- CPU 성능
  - 자료 버스의 폭(크기)
  - 사이클당 연산 수
  - 레지스터의 수와 크기
  - 캐시의 크기
  - ...

자료 버스

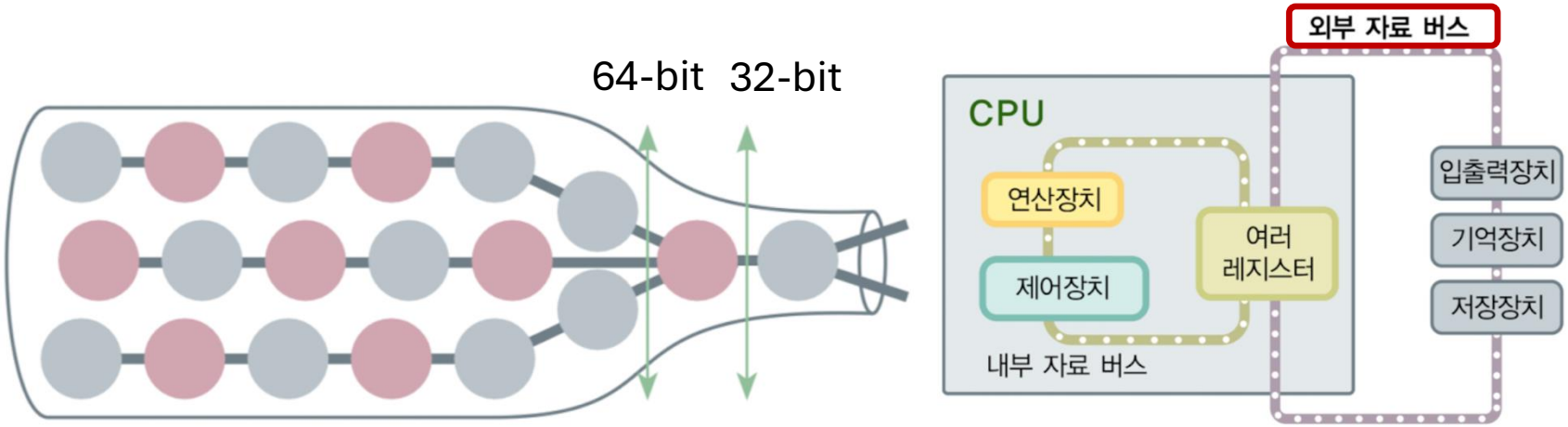
- 워드(Word)
  - 컴퓨터가 한 번에 작업할 수 있는 데이터의 크기
  - 32 bit CPU - 워드 크기 32 bit, 64 bit CPU - 워드 크기 64 bit
  - 레지스터의 크기와 일반적으로 같음

- 자료 버스(Data bus)
  - 데이터 전달 통로
  - 워드 크기와 같거나 클 수도 있음

자료 버스

- 자료 버스에 의한 병목 현상
  - 레지스터 64 bit
  - 자료 버스 32 bit

→ 한 번에 자료/데이터 전달 할 수 없음



클럭 속도

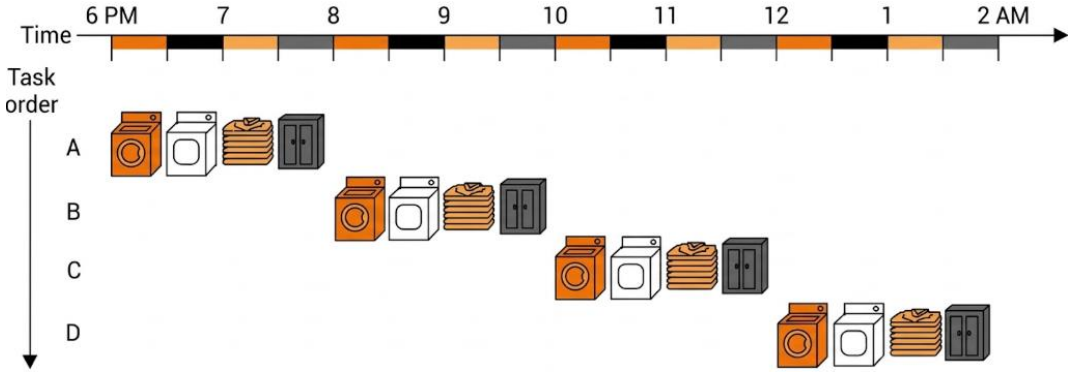
- 시스템 클럭(Clock)
  - CPU 내부 동작을 동기화하기 사용하는 주기적인 디지털 신호
  - 내부 회로(ALU, 제어장치 등)은 모두 처리 속도 다름

- 수정 진동자를 사용
  - 초당 수백만 회의 진동수를 일정하게 유지
  - 클럭 속도의 단위: Hz (1초당 진동의 반복 횟수)
    - 1GHz? 초당 2^{30}(10억) 회 진동
  - 컴퓨터는 이 진동수에 맞춰 연산 처리 시간을 조정

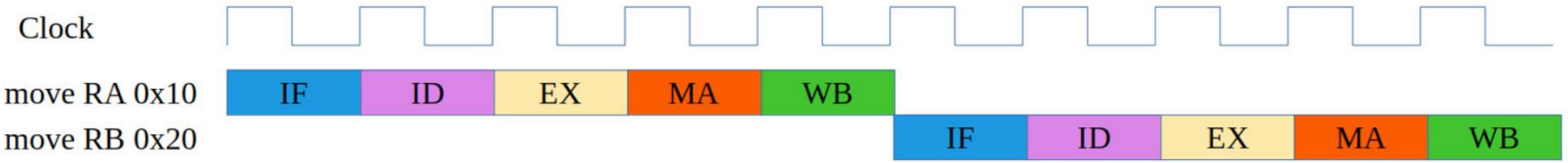
- CPU는 하나의 명령어를 여러 클럭 사이클에 걸쳐 실행
  → 클럭 속도는 연산 속도와 비례

클럭 속도

- CPU를 세탁소라고 생각
  - 전체 세탁 과정을 한 번 끝내기 위해 여러 단계가 필요
  - 각 과정은 일정한 리듬으로 진행

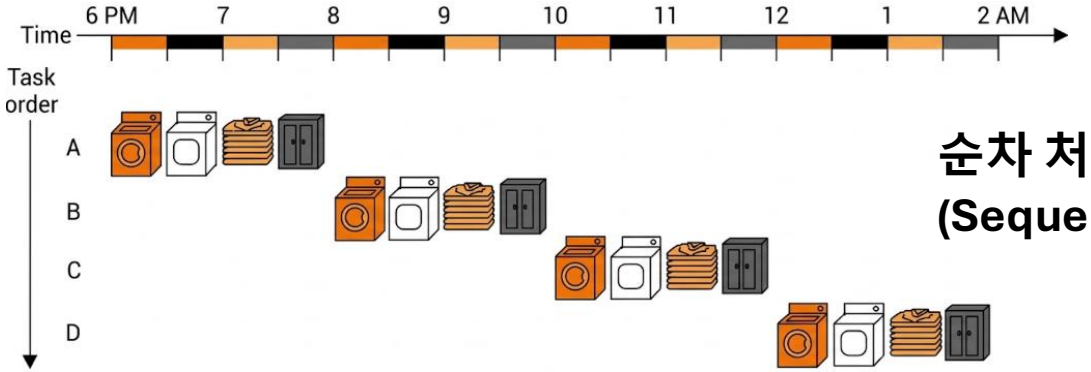


  - 단계 = IF (Instruction Fetch), ID (Instruction Decode), EX (Execute)
  - 리듬 = 클럭

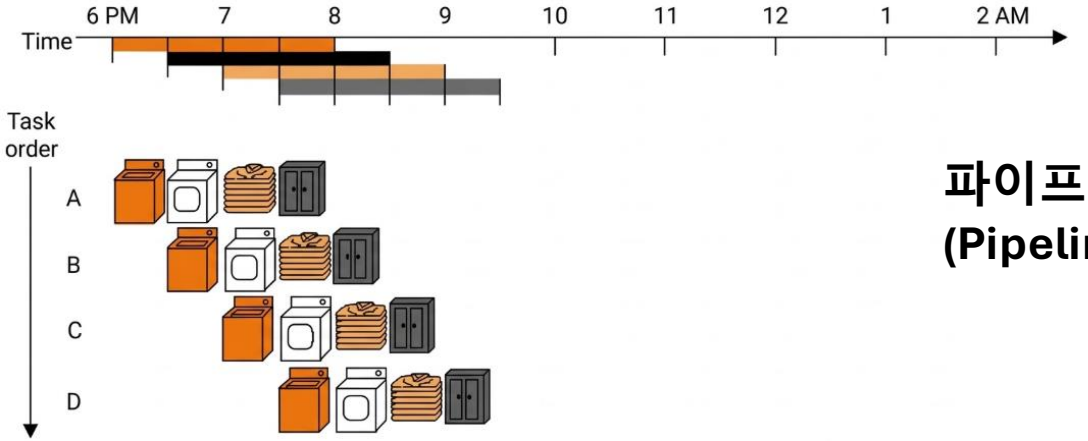


클럭 속도

- CPU를 세탁소라고 생각
  - 전체 세탁 과정을 한 번 끝내기 위해 여러 단계가 필요
  - 각 과정은 일정한 리듬으로 진행



순차 처리
(Sequential processing)

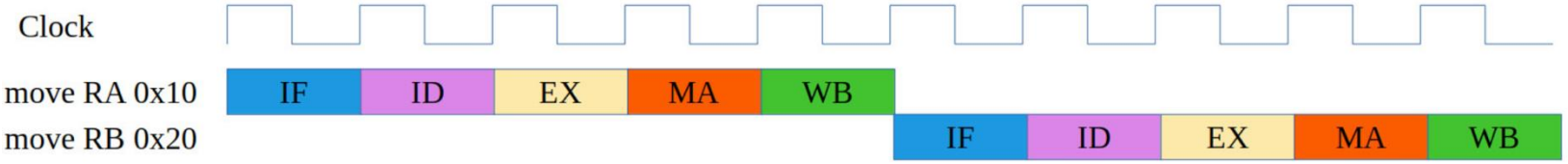


파이프라인 처리
(Pipelined processing)

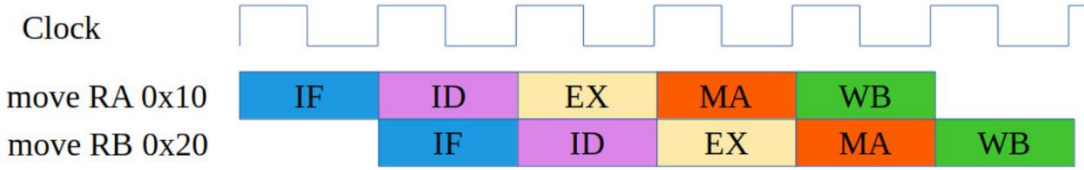
클럭 속도

- CPU를 세탁소라고 생각
  - 전체 세탁 과정을 한 번 끝내기 위해 여러 단계가 필요
  - 각 과정은 일정한 리듬으로 진행

순차 처리
(Sequential processing)



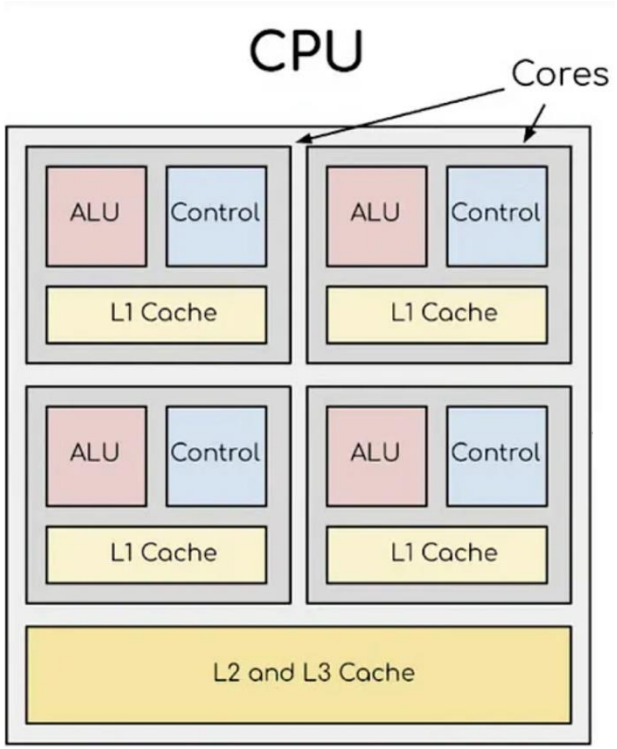
  - 단계 = IF (Instruction Fetch), ID (Instruction Decode), EX (Execute)
  - 리듬 = 클럭



파이프라인 처리
(Pipelined processing)

병렬 처리

- 하나의 컴퓨터에서 2개 이상의 CPU를 이용
- 한 번에 여러 명령어를 동시에 실행하는 방법



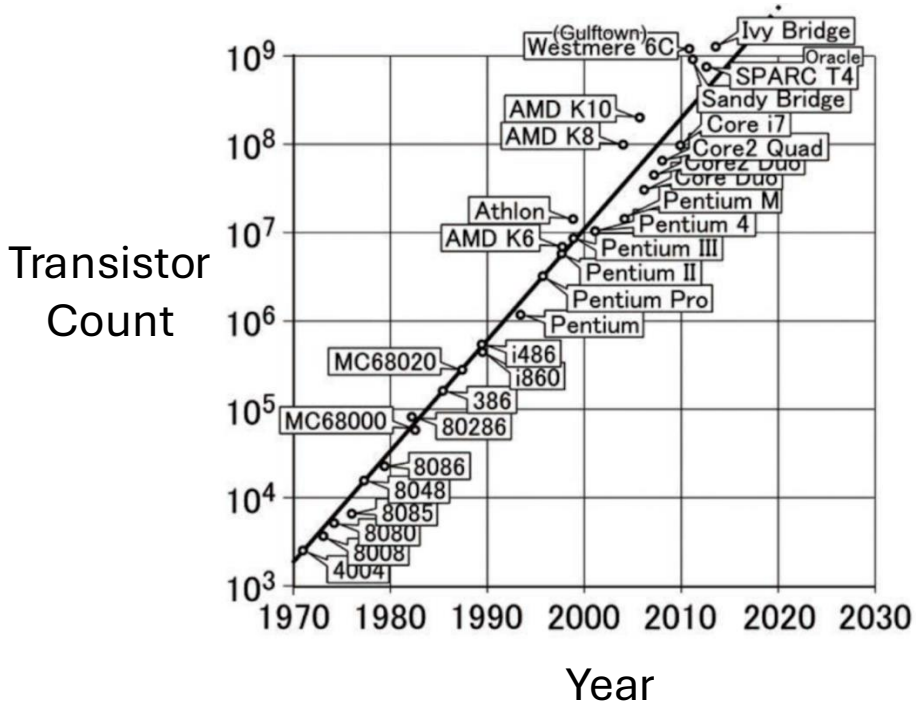
Intel CPU

- 1971년 하나의 칩으로 된 컴퓨터(Computer On a Chip)라 부르는 마이크로프로세서 4004 최초로 출시

발표 연도	명칭	버스 폭	클럭 속도
1971	4004	4비트	740 KHz
1974	8080	8비트	2 MHz
1979	8088	16비트	~8 MHz
1982	80286	16비트	~ 12 MHz
1985	80386	32비트	~ 33 MHz
1989	80486	32비트	~ 100 MHz
1993	Pentium	64비트	~ 200 MHz
1995	Pentium Pro	64비트	200 MHz ~
1997	Pentium MMX	64비트	233 MHz ~
1998	Pentium II	64비트	233 MHz ~
1998	Celeron	64비트	400 MHz ~
1999	Pentium III	64비트	450 MHz ~
2000	Pentium IV	64비트	1.3 GHz ~
2001	Itanium	64비트	800 MHz ~
2002	Itanium 2	64비트	900 MHz ~
2005	Pentium D	64비트	2.0 GHz ~
2006	Core	64비트	1.8 GHz ~
2006	Core 2 Duo	64비트	2.6 GHz ~
2007	Core 2 Quad	64비트	2.4 GHz ~
2008	Core i7	64비트	2.5 GHz ~
2009	Core i5	64비트	2.4 GHz ~
2010	Core i3	64비트	2.9 GHz ~
2011 이후	2, 3, 4세대 Core i3, Core i5, Core i7	64비트	2.5 GHz ~
2013	Core i7 Extreme Edition	64비트	4.0 GHz ~
2018	Core i9	64비트	4.8 GHz

무어의 법칙(Moore's Law)

- 인텔의 공동 창립자인 고든 무어(Gorden Moore)가 1965년도에 한 연설에서 유래



- *"반도체 칩의 트랜지스터 수가 일정 기간마다 2배로 증가한다"*
  - 약 2년

무어의 법칙(Moore's Law)

- 한계론
  - 2016년 국제반도체기술로드맵(ITRS) 위원회 - 반도체 기술의 미세화 기술이 2021년 이후에는 더 이상 지속될 수 없다는 사실을 공식적으로 선언
- 긍정론
  - 반도체 공정 및 디자인 부분에 혁신이 필요한 시기이며 이러한 어려움은 극복 가능
  - 미국 국가과학재단(National Science Foundation)은 '무어의 법칙 이면의 과학 및 엔지니어링'(Science and Engineering behind Moore's Law)이라는 이름의 프로젝트
    - 제조, 나노기술, 다중 코어칩, 양자 컴퓨팅 등의 새로운 기술 연구를 재정적으로 지원
  - 앞으로 몇 년 후에 무어의 법칙이 들어맞지 않더라도 계속된 노력으로 다른 측면의 기술은 발전 가능하다는 시각

명령어에 따른 분류

- 프로세서마다 고유한 명령어 집합 제공
- Complex Instruction Set Computing (CISC)
  - 복합 명령어 집합 컴퓨팅계열
- Reduced Instruction Set Computing (RISC)
  - 축소 명령어 집합 컴퓨팅 계열

CISC

- Complex Instruction Set Computing (CISC)
  - 복합 명령어 집합 컴퓨팅 계열

"명령어 하나로 최대한 많은 일을 하자"

- 복잡한 명령어
  - 한 명령어가 여러 작업 수행
- 적은 명령어 수
- 메모리 직접 참조하는 연산 명령어 다수

ADD [mem], EAX

- 대표 예시
  - Intel, AMD (x86)

RISC

- Reduced Instruction Set Computing (RISC)
  - 축소 명령어 집합 컴퓨팅 계열

"단순한 명령어를 이용해서 빠르게 처리하자"

- 간단한 명령어
- 많은 명령어 수
- 레지스터가 많음
- 메모리 직접 참조는 제한적
  - LOAD/STORE 명령어만 가능
  - 모든 명령어는 레지스터 사용

```
LOAD R1, [mem]
ADD R2, R1, R3
STORE R2, [mem]
```

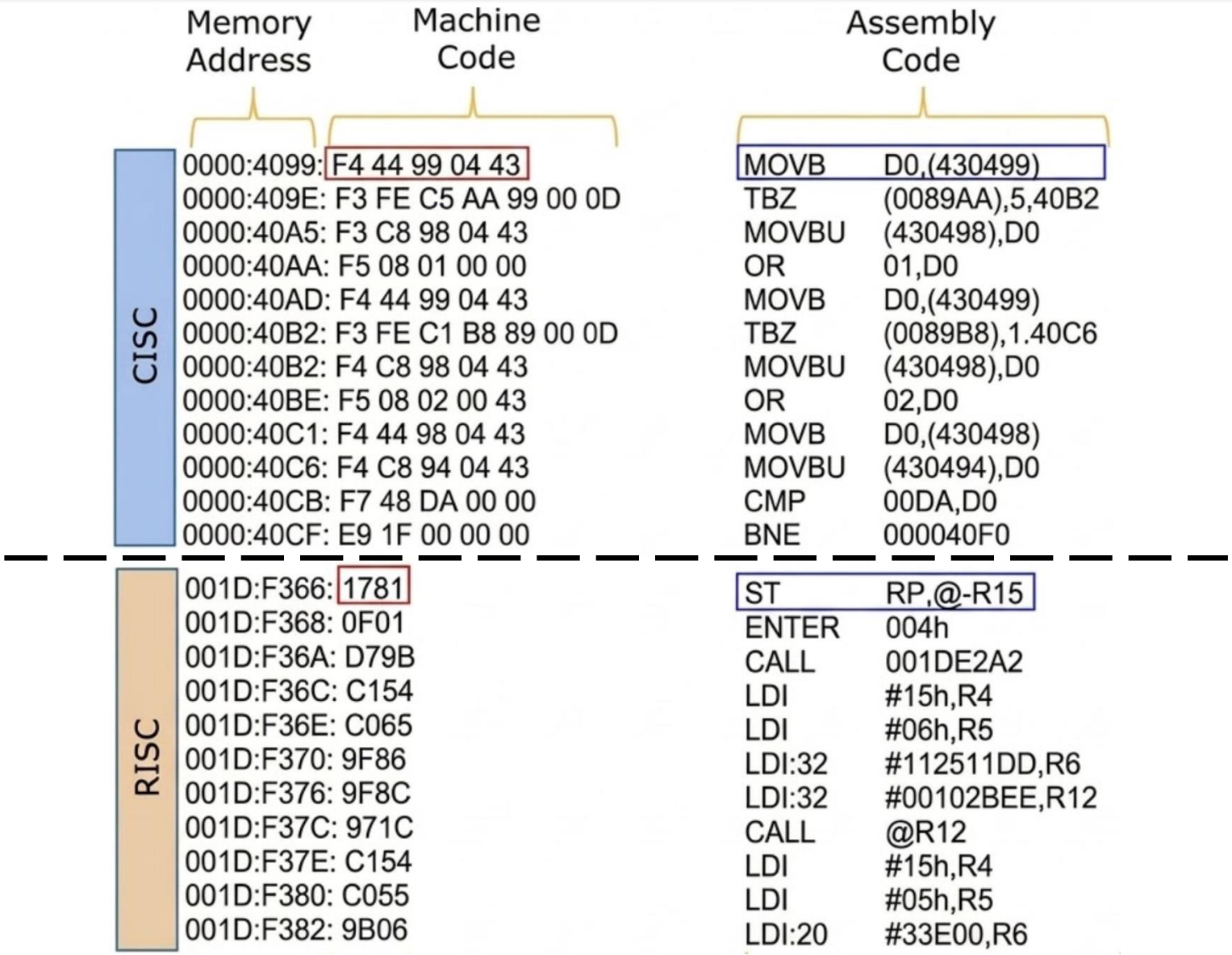
- 대표 예시
  - ARM

CISC vs. RISC

- **CISC**
  - Pros
    - 복잡한 프로그램을 적은 수의 명령어로 구성 가능
  - Cons
    - 하드웨어 복잡, 전력소모 증가
    - 명령어의 길이가 상이 (처리 시간이 다름)

- **RISC**
  - Pros
    - 명령어 길이가 일정
    - 하드웨어 단순
  - Cons
    - 명령어 수가 많아짐, 프로그램 코드 크기 증가

CISC vs. RISC


Memory Address
Machine Code
Assembly Code
CISC
0000:4099: F4 44 99 04 43
0000:409E: F3 FE C5 AA 99 00 0D
0000:40A5: F3 C8 98 04 43
0000:40AA: F5 08 01 00 00
0000:40AD: F4 44 99 04 43
0000:40B2: F3 FE C1 B8 89 00 0D
0000:40B2: F4 C8 98 04 43
0000:40BE: F5 08 02 00 43
0000:40C1: F4 44 98 04 43
0000:40C6: F4 C8 94 04 43
0000:40CB: F7 48 DA 00 00
0000:40CF: E9 1F 00 00 00
MOVB D0,(430499)
TBZ (0089AA),5,40B2
MOVBU (430498),D0
OR 01,D0
MOVB D0,(430499)
TBZ (0089B8),1.40C6
MOVBU (430498),D0
OR 02,D0
MOVB D0,(430498)
MOVBU (430494),D0
CMP 00DA,D0
BNE 000040F0
RISC
001D:F366: 1781
001D:F368: 0F01
001D:F36A: D79B
001D:F36C: C154
001D:F36E: C065
001D:F370: 9F86
001D:F376: 9F8C
001D:F37C: 971C
001D:F37E: C154
001D:F380: C055
001D:F382: 9B06
ST RP,@-R15
ENTER 004h
CALL 001DE2A2
LDI #15h,R4
LDI #06h,R5
LDI:32 #112511DD,R6
LDI:32 #00102BEE,R12
CALL @R12
LDI #15h,R4
LDI #05h,R5
LDI:20 #33E00,R6

Summary

- 프로그램 내장 방식
- 메모리의 계층 구조
  - 주기억장치
  - 보조기억장치
- CPU
  - Machine cycle
  - 프로그램 실행 과정
  - CPU 성능
  - CISC, RISC