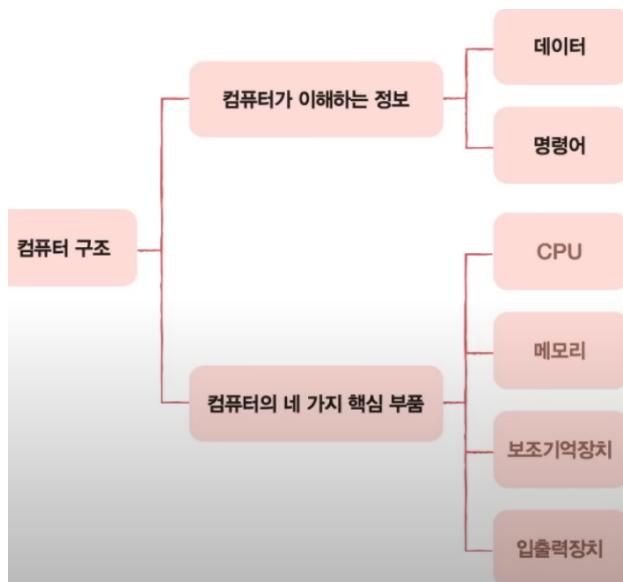
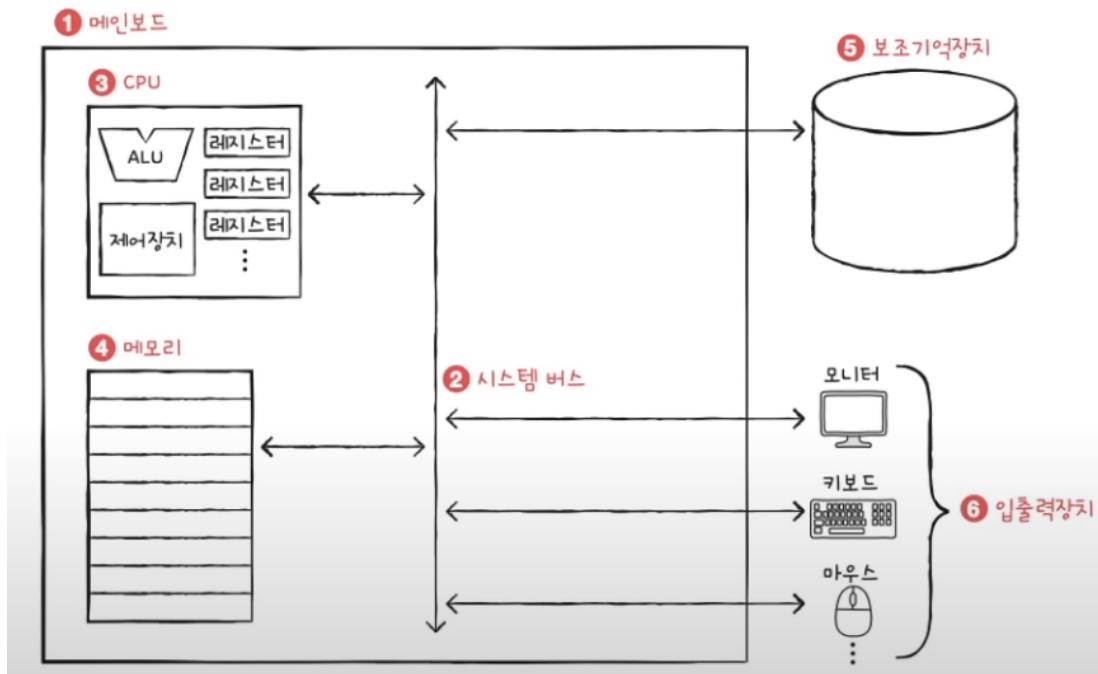


Chapter 1. 컴퓨터 구조 (문제 해결, 성능/용량/비용)

- 문제 해결 (개발자의 기본 소양)
 - 같은 코드를 작성했으나 특정 컴퓨터에서는 안되거나, 개발과정에서 잘 되던 코드가 실제 현장에서 잘 안되거나 등 여러 상황에 봉착하게 됨
 - 컴퓨터 구조를 이해한다면 문법만으로는 알기 어려운 성능, 용량, 비용 등을 고려할 수 있다.
- Comments
 - 컴퓨터의 구조를 이해한다면 문제 해결, 업무 수행 역량이 증대된다.
 - 일반인들은 개발자가 컴퓨터에 대해서 만능일 것이라는 기대치를 충족해주지 못하면 불편한 상황에 봉착하기 쉬운 편이다.
 - *Amagramer, Coder*라는 단어가 아닌 *Programer, Developer* 명칭으로 불리려면 컴퓨터 구조 지식은 필수이며 새로운 지식의 꾸준한 학습이 필요하다.
- 컴퓨터 : 명령어를 처리하는 기계
- 컴퓨터 구조



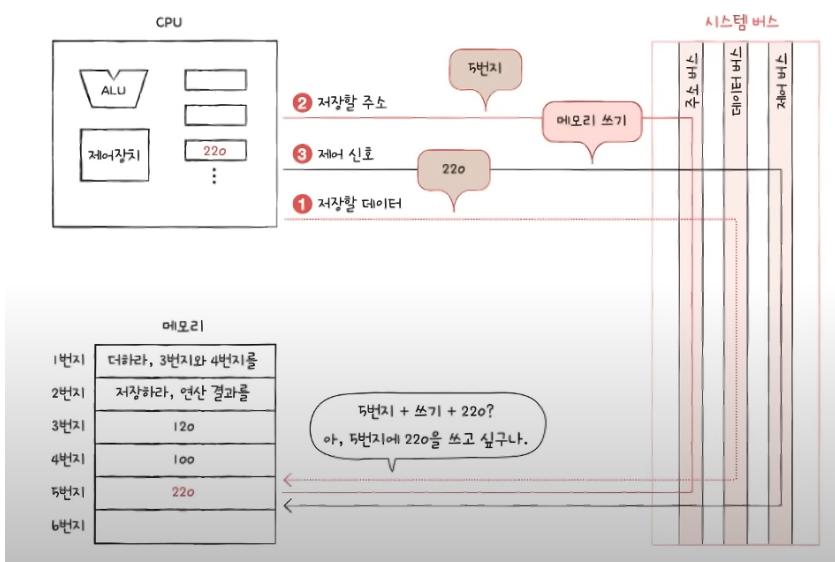
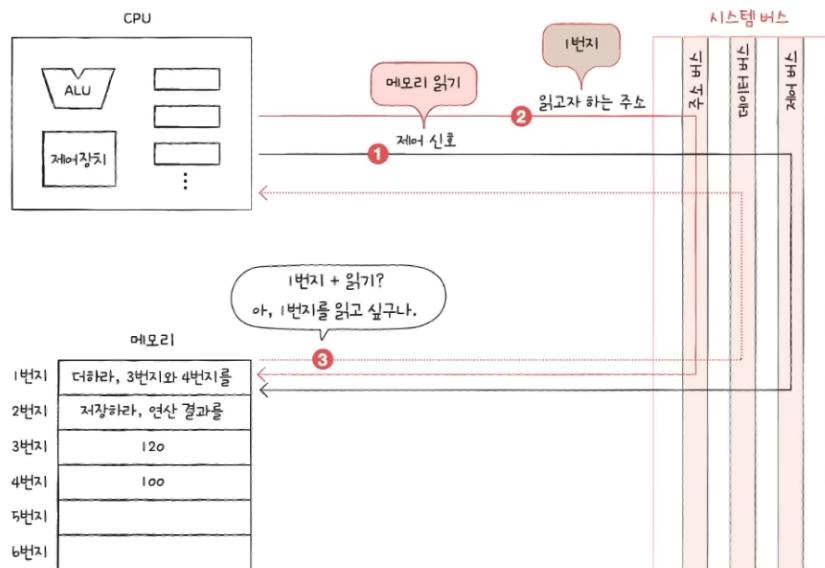
- 컴퓨터가 이해하는 정보
 - 데이터(data)
 - 명령어(instruction)
- 컴퓨터의 네 가지 핵심 부품
 - 중앙처리장치(CPU, Central Processing Memory)
 - 주 기억장치(Primary Memory, Computer Memory)
 - RAM(Random Access Memory)
 - ROM(Read Only Memory)
 - 캐시 메모리(Cache Memory)
 - 보조 기억장치(Secondary Memory, Auxiliary Memory)
 - HDD(Hard Disk Driver)
 - SSD(Solid State Drive)
 - 입출력장치(input/output(I/O) Device)
 - 모니터, 키보드, 마우스



- CPU : 컴퓨터의 두뇌, 메모리¹에 저장된 명령어를 읽고, 해석, 실행, 계산
 - 산술논리연산장치(ALU, Arithmetic Logic Unit) : 컴퓨터 내부에서 수행되는 대부분의 계산을 도맡는다.
 - 레지스터(Register) : CPU 내부의 임시 저장 장치로 프로그램이 실행할 때 필요한 값들을 임시로 저장한다. 서로 다른 이름과 다양한 역할을 한다.
 - 제어장치(CU, Control Unit) : 제어신호(Control Signal)라는 전기신호를 내보내고 명령어를 해석하는 장치
 - CPU가 메모리에 저장된 값을 읽고 싶을 때 메모리 읽기라는 제어신호 전송
 - CPU가 메모리에 어떤 값을 저장할 때 메모리 쓰기라는 제어신호 전송
- 메모리 : 현재 실행되는 프로그램의 명령어와 데이터를 저장하는 부품
 - 프로그램이 실행되기 위해서는 반드시 메모리에 저장(적재, Load)되어 있어야 한다.
 - 저장된 명령어와 데이터의 위치는 정돈되어 있어야 한다.
 - 명령어와 데이터는 모두 0과 1로 표현된다.
 - 주소(Address) : 메모리에 저장된 값에 빠르고 효율적으로 접근하기 위해 사용하는 개념
 - 메모리는 주로 주 기억장치(RAM, ROM)을 의미한다.
 - 가격이 비싸 저장 용량이 적다.
 - 전원이 꺼지면 저장된 내용을 읽는다.(RAM의 경우)
- 보조기억장치 : 전원이 꺼져 있어도 보관할 프로그램을 저장하는 부품
 - 종류 : HDD, SSD, USB, DVD, CD-ROM
 - 메모리는 현재 '실행되는' 프로그램을 저장, 보조기억장치는 '보관할' 프로그램을 저장
- 입출력장치 : 컴퓨터 외부에 연결되어 컴퓨터 내부와 정보를 교환할 수 있는 부품
 - 주변장치(Peripheral Device) : 모니터, 키보드, 마우스 등 컴퓨터 주변에 붙어있는 장치
 - 보조기억장치도 입출력 장치의 예시 일수 있으나 입출력장치에 비해 메모리를 보조한다는 특별한 기능을 수행하는 차이

¹ 메모리는 주로 RAM을 지칭한다. 넓게는 레지스터, RAM, 보조기억장치를 통틀어 일반적인 저장 장치들을 의미한다.

- 메인보드(Mainboard, Motherboard) : 컴퓨터 내부의 여러 부품들을 연결
 - 버스(Bus) : 메인보드와 연결된 부품이 서로 정보를 주고 받을 수 있는 통로
 - 시스템버스(System Bus) : 컴퓨터의 네 가지 핵심 부품들의 연결 통로
 - 주소버스(Address Bus) : 주소를 주고받는 통로
 - 데이터버스(Data Bus) : 명령어와 데이터를 주고받는 통로
 - 제어버스(Control Bus) : 제어신호를 주고받는 통로
 - CPU의 제어장치는 제어버스를 통해 제어 신호를 송수신
 - i. CPU가 메모리속 명령어를 읽기 위해
제어버스를 통해 ‘메모리읽기’ 제어 신호 보냄
 - ii. CPU가 주소버스를 통해 읽고자하는 주소도 보냄
 - iii. 메모리는 데이터버스를 통해 CPU가 요청한 주소로 내용을 보냄
 - i. CPU가 메모리에 어떤 값을 저장하기 위해
데이터버스를 통해 메모리에 저장할 값을 보냄
 - ii. CPU가 주소버스를 통해 저장할 주소를 보냄
 - iii. 메모리는 제어버스를 통해 ‘메모리쓰기’ 제어 신호를 보냄
 - iv.



Chapter 2. 데이터 (비트, 바이트, 이진법, 2의 보수, 십육진법)

- 비트(Bit) : 0 or 1, 컴퓨터가 이해하는 가장 작은 정보 단위, 2^N 가지로 정보를 표현
 - 1 bit(2개) = (0,0), (0,1)
 - 2 bit(4개) = (0,0), (0,1), (1,0), (1,1)
 - 3 bit(8개) = (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)
- 바이트(Byte) : = 8 bit(256개)
 - 1 Kb, 1MB, 1GB, 1TB
- 워드(Word) : CPU가 한번에 처리할 수 있는 데이터의 크기, CPU마다 다르다.
 - CPU가 한번에 16bit를 처리할 수 있으면 1 워드는 16bit, 한번에 32bit를 처리할 수 있으면 1워드는 32bit가 된다.
 - 정의된 워드의 절반을 Half word, 1배 크기는 full word, 2배는 double word
 - 현재 컴퓨터의 워드 크기는 대부분 32bit or 64bit
 - 인텔의 x86 CPU는 32bit 워드, x64 CPU는 64 bit 워드 CPU
- 이진법(Binary) : 컴퓨터가 이해하는 0과 1로 모든 숫자를 표현
 - 2의 보수(two's complement) : 이진수의 음수표현, 어떤 수를 그보다 큰 2^N 에서 뺀값
 - 111에서 모든 0과 1을 치환하면 000인데 여기에 1을 더한 001이 됨
 - 플래그(Flag) : ALU 연산에 대한 추가 정보(양수or음수)

```
int num = 10; // 10진수
int bNum = 0B1010; // 2진수 0b
int oNum = 012; // 8진수 0
int xNum = 0xA; // 16진수 0x
```

- 십육진법(Hexadecimal) : 0~9, A~F

십진수	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...
십육진수	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	...

자리 올림

- 숫자앞에 0x를 붙여서 구분
- 이진수와 십육진수의 변환이 쉽기 때문에 사용
- 16 to 2 : 십육진수를 이루는 숫자 하나를 이진수로 표현할 때 4비트 2^4 (=16)가 필요
- 2 to 10 : 네자리 씩 끊어서 하나를 십육진수로 변환
- 문자 집합(Character Set) : 컴퓨터가 인식하고 표현할 수 있는 문자 모음
 - {a,b,c,d} 는 인식 f,g <-{괄호가 없음}은 미인식
 - 문자 인코딩(Character Encoding) : 문자를 0과 1로 변환해 컴퓨터가 이해하도록 함
 - 문자 디코딩(Character Decoding) : 0과 1을 문자 변환해 사람이 이해하도록 함

- 아스키코드(ASCII, American Standard Code for information Interchange) : 아스키 문자에 대응된 고유한 수

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	#	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELLOWS]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	,	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	:	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	-	127	7F	[DEL]

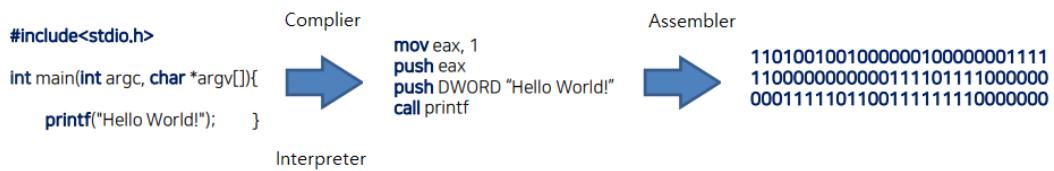
<https://www.ascii-code.com/>

- 초창기의 문자 집합중 하나로 영어, 알파벳, 아라비아 숫자, 일부 특수 문자를 포함
- 아스키 문자 : 각각 7 bit로 표현², 0 부터 127까지 총 128개의 문자중 하나의 고유한 수에 1:1 대응된다.
 - 코드포인트(Code Point) : 글자에 부여된 고유한 값
 - ex) 아스키 문자 A의 코드 포인트는 65
- 아스키 코드표에는 Backspace, Escape, Cancel, Space등 제어 문자도 포함되어 있다.
- 확장된 아스키(Extended ASCII, Extended BCD) : 더 다양한 문자 집합을 위해 1비트 추가한 8비트의 확장형
- EUC-KR : 한글을 2byte 크기로 인코딩할 수 있는 완성형 인코딩 방식
 - 초성,중성,종성이 모두 결합된 한글 단어에 2byte 크기의 코드를 부여
 - 한글은 16bit인 십육진수로 총 2,350개 정도의 한글 단어를 표현 가능
 - 웹, 뻥 은 불가, 은행 및 공공기관등 호환성 문제 존재
 - CP(code page)949 : MS사의 EUC-KR 확장 버전
- Unicode : 한글 포함 대부분 나라의 문자, 특문등을 코드로 표현하는 통일된 문자 집합
 - ex) '한' = D55C(16), '글' = AE00(16)
 - 십육진수의 유니코드 표현법 : 유니코드 글자에 부여된 값 앞에 U+ 를 붙임
 - 아스키 코드와 문자의 1:1 대응과 달리 유니코드는 다양한 인코딩이 존재
 - UTF-8 : 통상 1~4byte까지

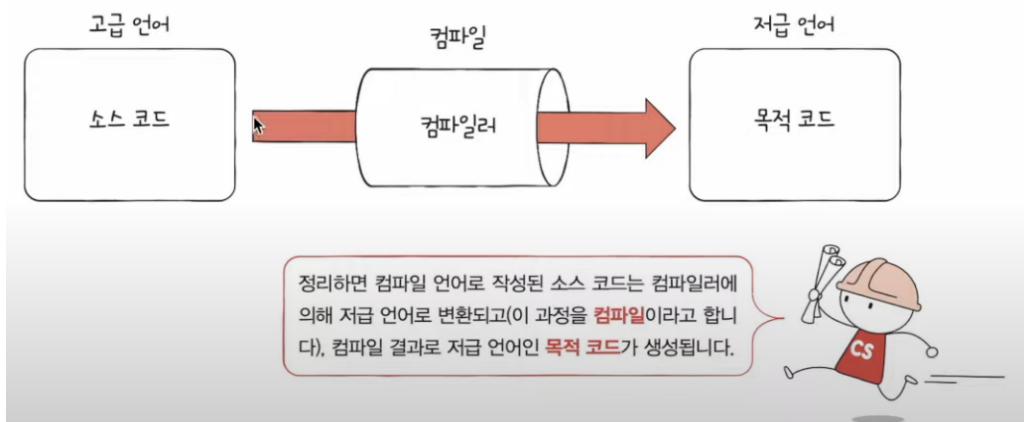
² 실제로는 하나의 아스키 문자를 나타내기 위해 8비트를 사용, 하지만 8비트 중 1비트는 패리티 비트(Parity Bit)라고 불리는, 오류 검출을 위해 사용되는 비트이기 때문에 실질적으로 문자 표현을 위해 사용되는 비트는 7 bit

Chapter 3. 명령어 (저급 언어, 고급언어, 기계어, 어셈블리어, 컴파일언어, 인터프리터 언어)

- UI -> 프로그래밍 언어(컴퓨터인식 불가) -> 명령어 -> 컴퓨터 인식
 - 고급언어 -변환-> 저급언어
- 고급 언어(High-Level programming language) : 컴퓨터가 이해하지 못하며 사람이 이해하고 작성하기 쉽게 만들어진 언어, 대부분의 프로그래밍언어가 여기에 속함
- 저급 언어(Low-Level programming language) : 컴퓨터가 이해하는 언어, 명령어
 - 기계어, 어셈블리어
- 기계어(Machine Code) : 0과 1로 이루어진 명령어 비트 모음
 - 가독성을 위해 16진수로 표현하기도 함
- 어셈블리어(Assembly language) : 기계어를 읽기 편한 형태로 번역한 언어
 - 하드웨어와 밀접하게 맞닿아 있는 프로그램을 개발하는 임베디드 개발자, 게임, 정보 보안 분야 등의 개발자가 주로 사용
 - 작성과 관찰의 언어로 프로그램이 어떤 과정으로 실행, 작동하는지 추적이 용이

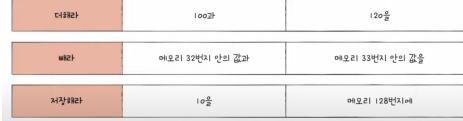


- 컴파일언어(compile language) : 컴파일러에 의해 코드가 저급 언어로 변환되는 고급 언어
 - 컴파일(compile) : 컴파일 언어로 작성된 소스 코드를 저급 언어로 변환되는 과정
 - 컴파일러(compiler) : 컴파일을 수행해 주는 도구
 - 개발자가 작성한 소스 코드에 문법적 오류, 실행 가능, 불필요 코드 확인 수행하여 저급 언어로 컴파일함, 오류 발생 시 컴파일에 실패함
 - 목적 코드(object code)³ : 컴파일러를 통해 저급언어로 변환 성공된 코드
- 인터프리터언어(interpreter language) : 인터프리터에 의해 코드가 한 줄씩 실행되는 고급 언어
 - ex) python
 - 인터프리터(interpreter) : 소스 코드를 한 줄씩 저급 언어로 변환, 실행해주는 도구
 - 컴파일러와는 달리 문법 오류되기전까지의 수행은 가능
 - 컴퓨터가 언어를 한 줄씩 해석& 실행하므로 컴파일러에 비해 느린 편
 - 컴파일언어와 인터프리터언어의 구분이 모호하기도 하며, python도 컴파일을 수행함, 둘다 가능하며 각각의 방식이 존재함



³ 심화 : 목적 코드는 실행파일이 되기 위해 링킹(linking) 작업을 통해 더하기, 출력 명령어로 실행된다.

- 스택(Stack) : 후입선출 방식(LIFO, Last in First out), 한쪽 끝만 열린 저장 공간(ex; 텁블러, 통)
 - Push : 저장, Pop : 호출
- 큐(Queue) : 선입선출 방식(First in First Out), 양쪽이 뚫린 저장 공간
- 명령어 : 연산 코드(operation code) 와 오퍼랜드(operand) 로 구성

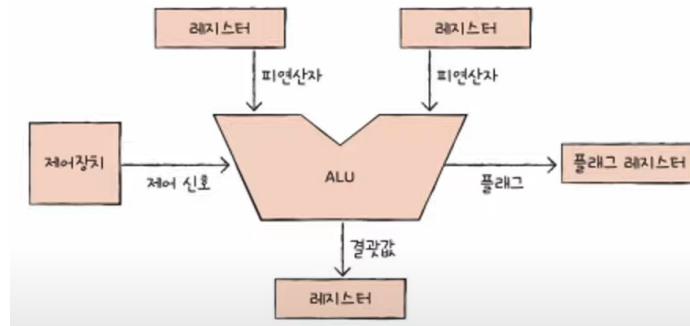


명령어 : 연산코드(레드), 오퍼랜드(화이트, 화이트, 화이트)

- 연산 코드 : 명령어가 수행할 연산
 - 데이터 전송
 - MOVE : 데이터를 이동
 - STORE : 메모리에 데이터를 저장
 - LOAD(FETCH) : 메모리에서 CPU로 데이터를 호출
 - PUSH : 스택에 데이터를 저장
 - POP : 스택의 최상단 데이터를 호출
 - 산술/논리 연산
 - ADD / SUBTRACT / MULTIPLY / DIVIDE : 사칙연산
 - INCREMENT / DECREMENT : 오퍼랜드에 1을 더하라 / 빼라
 - AND / OR / NOT : 수행
 - COMPARE : 두 개의 숫자 또는 TRUE / FALSE 값 비교
 - 제어 흐름 변경
 - JUMP : 특정 주소로 실행 순서 이동
 - CONDITIONAL JUMP : 조건 부합 시 특정 주소로 실행 순서 이동
 - HALT : 실행 종단
 - CALL : 되돌아올 주소를 저장한 채 특정 주소로 실행 순서 이동
 - RETURN : CALL 을 호출할 때 저장했던 주소로 복귀
 - 입출력 제어
 - READ(INPUT) : 특정 입출력 장치로부터 데이터를 읽음
 - WRITE(OUTPUT) : 특정 입출력 장치로부터 데이터를 쓰
 - START IO : 입출력 장치 시작
 - TEST IO : 입출력 장치 상태 확인
- 오퍼랜드 : 주소필드, 연산에 사용될 데이터 및 저장된 위치 or 메모리 주소 or 레지스터 이름
 - 0,1,2,3-주소 명령어 : 오퍼랜드에 위치할 명령어는 0~3개도 가능
 - 명령어의 크기나 길이가 크다면 오퍼랜드에 사용할 최대 용량도 배분해야 함
 - 유효 주소(Effective address) : 연산의 대상이 되는 데이터가 저장된 위치
- 주소 지정 방식(Addressing Mode) : 연산에 사용할 데이터 위치를 찾는 방법
 - 즉시(immediate AM) : 연산코드 + 연산 데이터
 - 데이터가 작지만 찾는 과정이 없기에 빠른 편
 - 직접(direct AM) : 연산코드 + 유효 주소 + 메모리(연산 데이터)
 - 즉시 주소 방식보다 커진 편, 메모리 호출로 느려짐
 - 간접(indirect AM) : 연산코드+유효주소의주소+메모리(유효주소->연산 데이터)
 - 직접 주소보다 유효 주소만큼의 범위가 증가, 메모리 이중접근으로 더 느림
 - 레지스터(register AM) : 연산코드+유효주소+레지스터(연산 데이터)
 - 직접 주소와 유사, CPU 내부의 레지스터가 더 빠름
 - 레지스터 간접(register indirect AM) :
 - 연산코드+유효주의주소+레지스터(유효주소)+메모리(연산 데이터)
 - 간접 주소 지정 방식보다 빠른 편

Chapter 4. CPU 작동 원리 (ALU, 플래그, 레지스터, 주소 지정 방식, 명령어 사이클, 예외)

- 산술논리연산장치(ALU, Arithmetic Logic Unit)

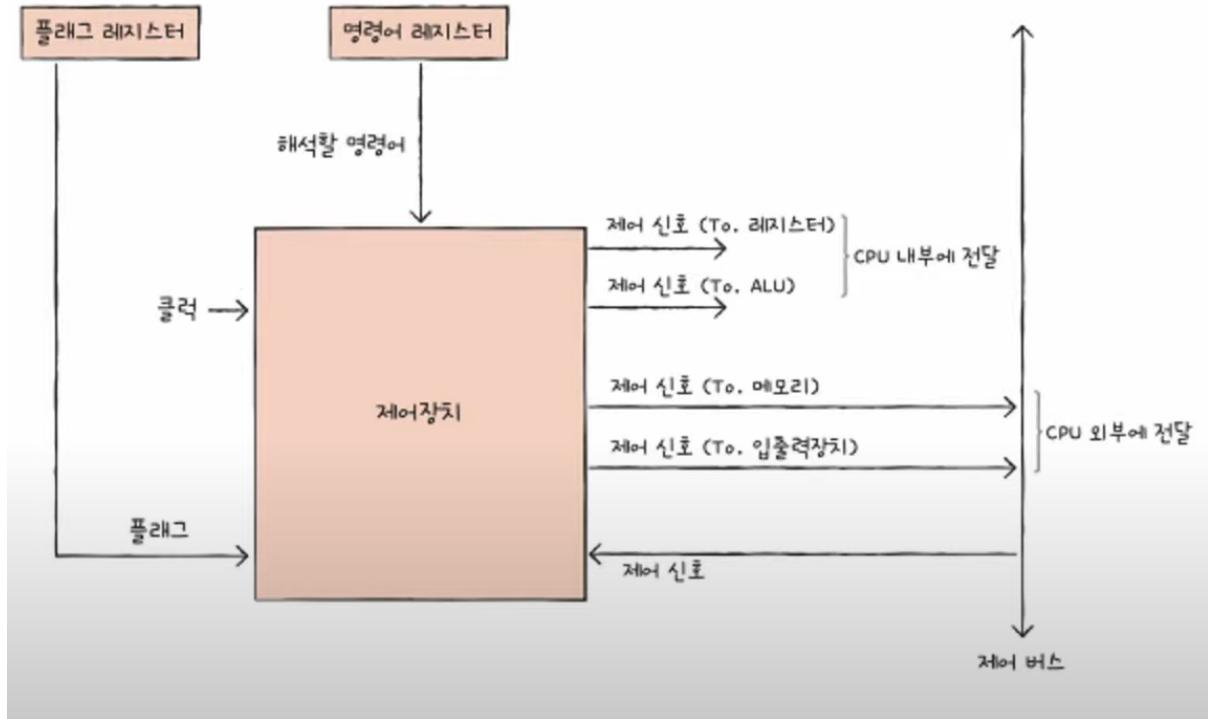


- 레지스터(피연산자)를 통해 연산할 데이터 (1, 1)을 받음
- 제어장치(제어신호)를 통해 수행할 연산 (사칙연산)을 받음
- 결과값(문자, 숫자, 주소등)을 레지스터에 임시 보관
 - 임시 보관처가 있어 메모리와 낮은 접촉, 빠른 실행
- 플래그를 플래그 레지스터에 보냄

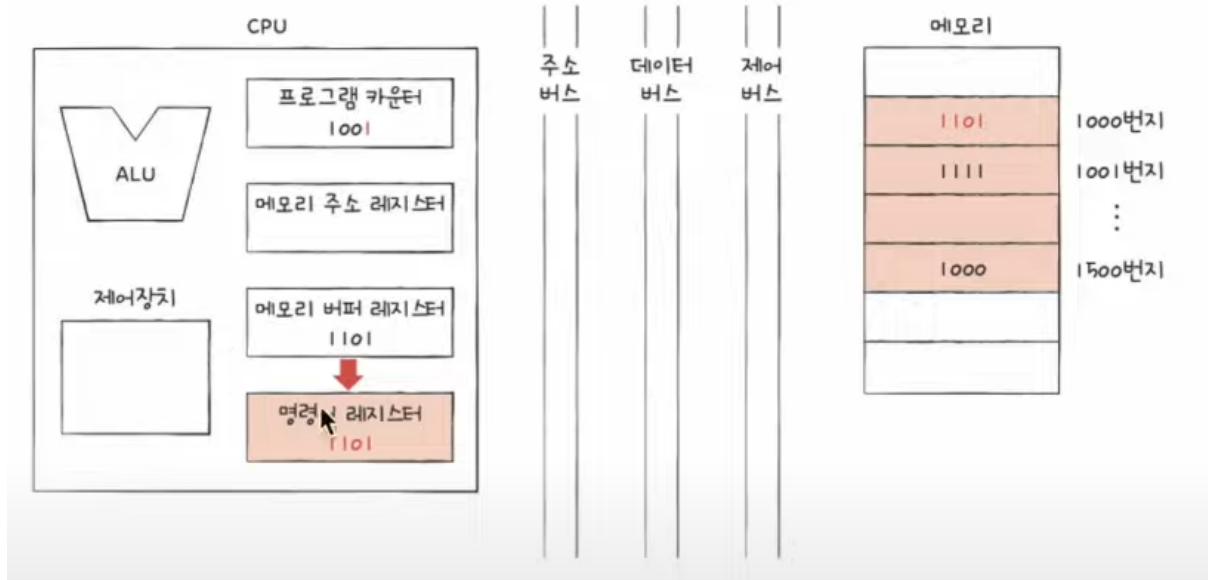
플래그 종류	의미	사용 예시
부호 플래그	연산한 결과의 부호를 나타낸다.	부호 플래그가 1일 경우 계산 결과는 음수. 0일 경우 계산 결과는 양수를 의미한다.
제로 플래그	연산 결과가 0인지 여부를 나타낸다.	제로 플래그가 1일 경우 연산 결과는 0. 0일 경우 연산 결과는 0이 아님을 의미한다.
캐리 플래그	연산 결과 올림수나 빌림수가 발생했는지를 나타낸다.	캐리 플래그가 1일 경우 올림수나 빌림수가 발생했음을 의미하고, 0일 경우 발생하지 않았음을 의미한다.
오버플로우 플래그	오버플로우가 발생했는지를 나타낸다.	오버플로우 플래그가 1일 경우 오버플로우가 발생했음을 의미하고, 0일 경우 발생하지 않았음을 의미한다.
인터럽트 플래그	인터럽트가 가능한지를 나타낸다. 인터럽트는 04-3절에서 설명한다.	인터럽트 플래그가 1일 경우 인터럽트가 가능함을 의미하고, 0일 경우 인터럽트가 불가능함을 의미한다.
슈퍼바이저 플래그	커널 모드로 실행 중인지, 사용자 모드로 실행 중인지를 나타낸다. 커널 모드와 사용자 모드는 09장에서 설명한다.	슈퍼바이저 플래그가 1일 경우 커널 모드로 실행 중임을 의미하고, 0일 경우 사용자 모드로 실행 중임을 의미한다.

- 이외에도 가산기(+), 보수기(-), 시프터(시프트연산), 오버플로우 검출기(결과값이 레지스터보다 큰 상황)등이 있음

- 제어장치(CU, Control Unit) : CPU 내외부에 제어 신호를 송수신함

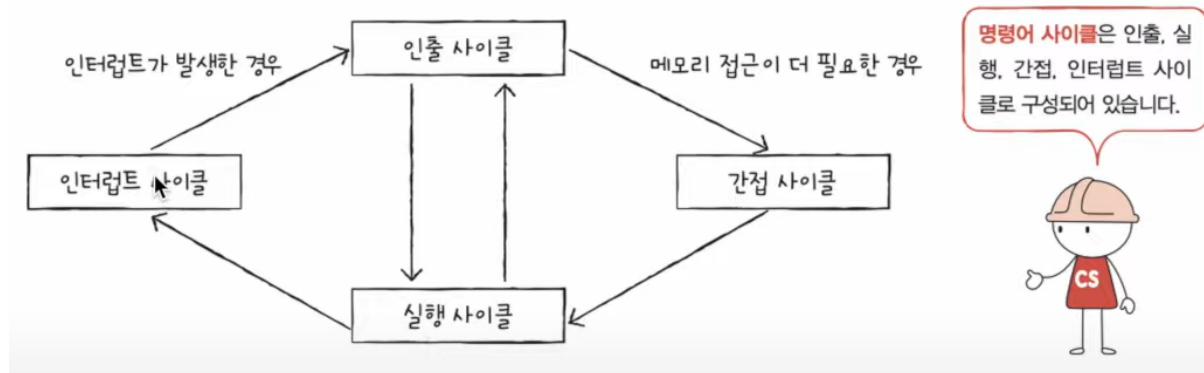


- 제어 장치가 받는 제어 신호
 - 클럭(clock, 일정한 주기) 신호를 받음
 - 명령어 레지스터에 저장된 해석할 명령어를 받음
 - 해석 후 제어신호로 부품들에게 수행할 내용을 전파
 - 플래그 레지스터에 저장된 플래그(부가정보)를 받음
 - 플래그 값을 확인하여 제어 신호 발생
 - 컴퓨터 내부에서 발생, 시스템 버스를 통해 제어 신호를 받음(3 핵심부품)
 - 컴퓨터 외부에서 발생, 제어 버스를 통해 제어 신호를 받음(I/O 장치)
 - 제어장치가 내보내는 제어 신호
 - 메모리로부터 읽거나 저장하는 제어 신호를 전달
 - 입출력장치의 값을 읽거나 새로운 값을 제어 신호로 전달
 - 레지스터(Register) : CPU 내부의 임시저장소
 - 프로그램 속 명령어와 데이터는 실행 전후로 반드시 레지스터에 저장됨
 - 프로그램카운터(PC, Program Counter) or 명령어포인터(IP, Instruction Pointer) : 메모리에게 전달할(읽을) 명령어의 주소를 저장
 - 메모리 주소 레지스터(MAR) : 주소 버스를 통해 메모리 주소를 메모리에게 전달
 - 메모리 버퍼 레지스터(MBR) or 메모리 데이터 레지스터(MDR) : 메모리와 주고받을 값을 데이터 버스를 통해 송수신
 - 명령어레지스터(IR, Instruction Register) : 메모리에서 읽은 명령어를 저장, 해석한 뒤 제어 신호를 전달
 - 범용레지스터(General Purpose Register) : MAR은 주소 버스로 주소값을 송수신 + MBR은 데이터 버스로 값을 송수신, GPR은 이 둘을 모두 저장 가능



- i. PC에 1000 이 저장
 - ii. PC가 MAR에 1000을 전달
 - MAR은 주소버스를 통해 메모리 주소를 메모리에 전달
 - CU는 제어버스를 통해 메모리에 메모리 읽기를 명령
 - iii. 메모리가 데이터버스를 통해 1000의 대응값을 MBR에 전달
 - PC는 다음 명령어 수행, 카운터 증가
 - iv. MBR에 IR에 저장된 값을 전달
 - v. IR은 전달된 값을 해석, 제어 신호를 발생
- Comments : PC -> MAR -> RAM (address, Control signal) -> MBR -> IR
- 스택 주소 지정 방식 : 스택과 스택 포인터를 이용
 - 스택 포인터 : 해당 스택의 꼭대기(가장 먼저 호출될 값)를 가리키는 레지스터
 - 스택 영역 : 메모리 안에 다른 공간과는 달리 스택처럼 사용하도록 지정된 영역
- 변위 주소 지정 방식(displacement addressing mode) : 오퍼랜드의 필드 값(변위)과 특정 레지스터의 값을 더해 메모리의 유효 주소를 알아냄
 - 변위 값과 어떤 레지스터를 더하는지에 따라 상대, 베이스로 나뉨
- 상대 주소 지정 방식(relative addressing mode) : 변위 값과 프로그램 카운터 값을 더해 메모리의 유효 주소를 알아냄
 - 베이스 레지스터 주소 지정 방식(base-register addressing mode) : 변위 값과 베이스 레지스터 값을 더해 메모리의 유효 주소를 알아냄
 - 베이스 레지스터는 기준 값을, 변위 값은 기준 값으로 부터 이동될 값이 저장됨

- 명령어 사이클 : 하나의 명령어가 처리되는 주기로 인출, 실행, 간접, 인터럽트 사이클로 구성



- 인출 사이클(fetch cycle) : 메모리에 저장된 명령어를 CPU로 가져옴
- 실행 사이클(execution cycle) : 제어장치가 명령어 레지스터에 담긴 값을 해석, 제어 신호 발생
 - 대체로 인출 <-> 실행의 반복되는 편
- 간접 사이클(indirect cycle) : 간접 주소 방식의 경우 오퍼랜드 필드에 유효 주소의 주소를 명시하므로 메모리에 한번 더 접근하는 과정
 - 인출->메모리 접근+1->실행->인출
- 인터럽트⁴(interrupt) : CPU 작업 방해, 끌어들기
 - 예외(exception) or 동기 인터럽트(synchronous interrupt) : 프로그래밍상 오류, CPU는 작업 중단하고 해당 예외를 처리함
 - 폴트(fault) : 예외를 처리한 직후 예외가 발생한 명령어부터 실행을 재개, 예외부터 처리
 - 트랩(trap) : 예외를 처리한 직후 예외가 발생한 명령어의 다음 명령어부터 실행을 재개, 건너뛰기, 디버깅용
 - 중단(abort) : CPU가 실행중인 프로그램을 강제 중단, 심각한 오류 발생
 - 소프트웨어 인터럽트 : 시스템 호출 발생
 - 비동기 인터럽트(asynchronous interrupt) or 하드웨어 인터럽트 : 알림, 입출력 장치가 작업을 끝내고 완료 알림(인터럽트)을 보내는 것
 - 처리순서
 - 인terrupt 요청 신호 : 정상 작업중인 CPU에게 작업 필요 전달(ex: I/O 장치들)
 - 인terrupt 플래그 : 가능이 되면 요청 받고, 불가능하면 요청 무시
 - 수락시 기존 작업을 스택에 백업함
 - 인terrupt 서비스 루틴 or 인terrupt 핸들 : 요청 수락시 어떻게 진행할지에 대한 정보
 - 기존 작업 복귀
 - 인terrupt 벡터 : 주소를 통해 우선 처리할 인terrupt 서비스 루틴을 수행

⁴ 폴링(polling) : CPU가 장치 컨트롤러의 상태 레지스터를 확인하여, 입출력 장치의 상태와 처리할 데이터가 있는지 주기적으로 확인하는 방식

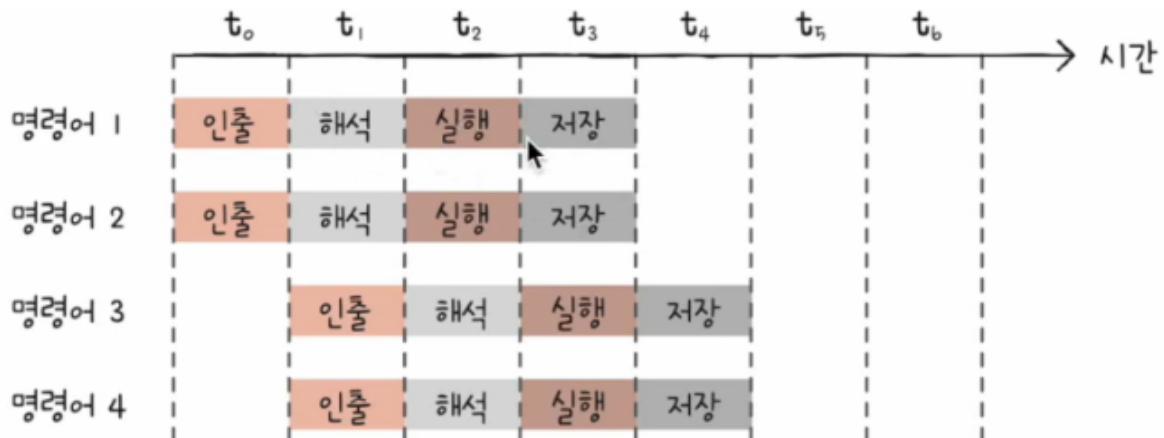
Chapter 5. CPU 성능 향상 기법 (ALU, 플래그, 레지스터, 주소 지정 방식, 명령어 사이클, 예외)

- 클럭속도 : 1초에 클럭이 반복되는 속도, 헤르츠(Hz) 단위
 - ex) 2.5GHz ~ 4.9GHz($4.9 * 10^9$) : 초당 25억 ~49억
- 코어(Core) : CPU 내부에서 명령을 실행하는 부품들(ALU, 제어장치, 레지스터들)
 - 멀티코어 프로세서 : 명령어를 실행할 수 있는 하드웨어 부품이 CPU안에 두개 이상
- 스레드(thread) : 실행 흐름의 단위
 - 하드웨어적 스레드 : 하나의 코어가 동시에 처리하는 명령어 단위(논리 프로세서)
 - 멀티스레드 프로세서 : 하나의 코어로 여러 명령어 동시 실행
 - 소프트웨어적 스레드 : 하나의 프로그램에서 독립적으로 실행되는 단위
 - 각각의 스레드가 화면표시, 맞춤법 검사, 수시저장등을 수행
- 명령어 병렬 처리(ILP, Instruction-Level Parallelism) :
- 명령어 파이프라인 : 같은 단계(인출->인출)가 겹치지 않으면 명령어를 동시에 실행



- 명령어 파이프라인 : 인출, 해석, 실행, 저장 정도를 하나의 세트로 실행
- 데이터 위험 : 명령어간 데이터 의존성에 의해 발생
 - 명령어 1 : $R1=R2+R3$, 명령어 $R4=R1+R5$ 가 동시실행시 꼬임
- 제어 위험 : 분기등에 의한 프로그램 카운터의 갑작스러운 변화
 - skip으로 다다음 명령어 실행해야 하는데 다음 명령어의 작업물은 헛수고
 - 분기 예측(branch prediction) : 분기 할 주소를 미리 예측, 인출하는 기술
- 구조적위험 : 자원 위험, 동시 진행중 같은 CPU부품(ALU) 사용시 발생

- 슈퍼스칼라(superscalar) : CPU 내부에 여러개의 파이프라인을 포함한 구조



- 비순차적 명령어 처리(OoOE, Out-of-order execution) : 의존성 없는 명령어의 순서를 조절해 효율성 증가, 명령어의 합법적 새치기, 순서를 앞당겨도 실행에 무리가 없음, 파이프라인의 중단 방지 효과

```

① M(100) ← 1
② M(101) ← 2
④ M(150) ← 1
⑤ M(151) ← 2
⑥ M(152) ← 3
⑦ M(102) ← M(100) + M(101)

```

- 명령어 집합 구조(ISA, Instruction set Architecture) : CPU의 언어
 - CPU 사에 따라 명령어 차이가 있으며 이로 인해 컴퓨터 구조와 설계 방식의 차이 CISC(x86) vs IOS에 따라 컴파일, 어셈블리어도 다르다.
- CISC(Complex Instruction set Computer) : 복잡한 명령어 집합을 사용하는 컴퓨터(CPU)
 - 가변 길이 명령어 : 명령어의 형태와 크기가 다양, 독특한 주소 지정 방식
 - 명령어의 수가 적으면 컴파일된 프로그램의 크기도 작을 수 있음
 - 복잡해서 많은 클럭 주기가 필요하며 사용 빈도가 낮음
 - 개선점 : 명령어 길이와 수행 시간이 짧고 규격화 필요, 자주 쓰이는 기본 명령어를 작고 빠르게 하는게 효율적
- RISC(Reduced Instruction set Computer) : 1클럭 내외의 명령어 파이프라인에 최적화
 - 고정길이 명령어
 - CISC보다 레지스터를 이용한 연산이 많고, 범용 레지스터 개수도 많음
 - 명령어 개수가 적기에 CISC보다 많은 명령으로 프로그램 동작
- Comments : CISC - 복잡하고 다양한 종류의 가변 길이 명령어 집합 활용
RISC - 단순하고 적은 종류의 고정 길이 명령어 집합 활용

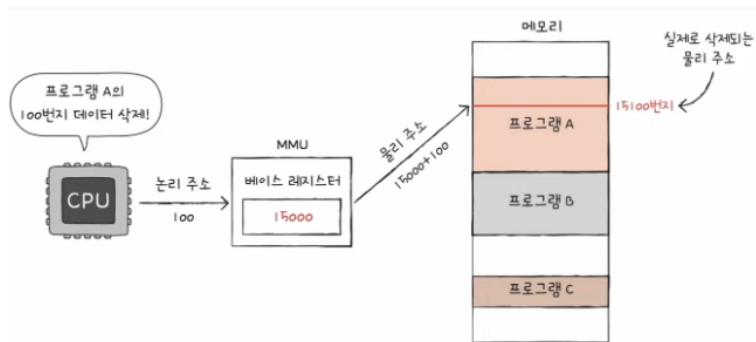
CISC	RISC
복잡하고 다양한 명령어	단순하고 적은 명령어
가변 길이 명령어	고정 길이 명령어
다양한 주소 지정 방식	적은 주소 지정 방식
프로그램을 이루는 명령어의 수가 적음	프로그램을 이루는 명령어의 수가 많음
여러 클럭에 걸쳐 명령어 수행	1클럭 내외로 명령어 수행
파이프라인하기 어려움	파이프라인하기 쉬움

Chapter 6. 메모리

- RAM(Random Access Memory) : 프로그램 실행 위치, 휘발성 저장
 - 용량이 커지더라도 실행 속도가 비례해서 증가하지 않는다.
- ROM(Read Only Memory) : 프로그램 저장 위치, 비휘발성 저장
 - CPU : 책/ RAM : 책상 / ROM : 책장
- DRAM(Dynamic RAM) : 주기적 데이터 활성화(저장) 필요, 대용량 설계가 용이하므로 일반적 메모리의 기준 중 하나
- SRAM(Static RAM) : 비휘발성, 주로 캐시메모리 용도

	DRAM	SRAM
재충전	필요함	필요 없음
속도	느림	빠름
가격	저렴함	비쌈
집적도	높음	낮음
소비 전력	적음	높음
사용 용도	주기억장치(RAM)	캐시 메모리

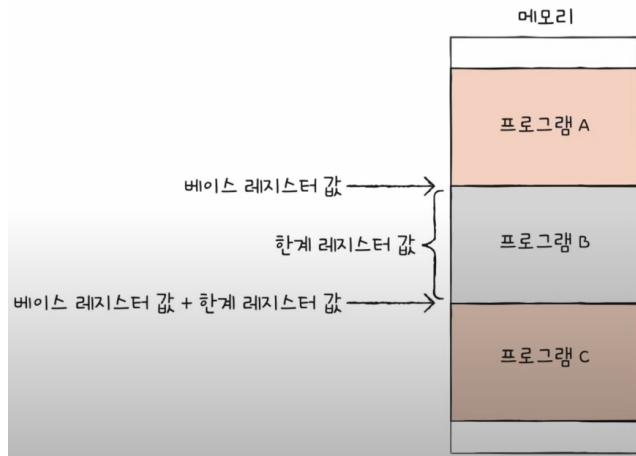
- SDRAM(synchronous DRAM) : 클럭마다 맞춰 동작하며 CPU와 정보를 송수신하는 DRAM
 - = SDR SDRAM(Single Data Rate SDRAM)
- DDR SDRAM(Double Data Rate SDRAM) : 최근형 RAM, 넓은 대역폭⁵
 - DDR2 SDRAM(2배 넓음), DDR4(4배)
- 물리주소 : 메모리 하드웨어가 사용하는 주소 # 실제 저장중인 주소
- 논리주소 : CPU와 실행중인 프로그램이 사용중하는 주소 # 메모리에 적재될 임시 주소
- 메모리 관리 장치(MMU, Memory Management Unit) : 하드웨어에서 논리주소와 물리주소 간의 변화를 담당하며, CPU와 주소 버스사이에 위치
 - CPU가 발생시킨 논리 주소에 베이스 레지스터 값과 더하여 논리를 물리 주소로 변환



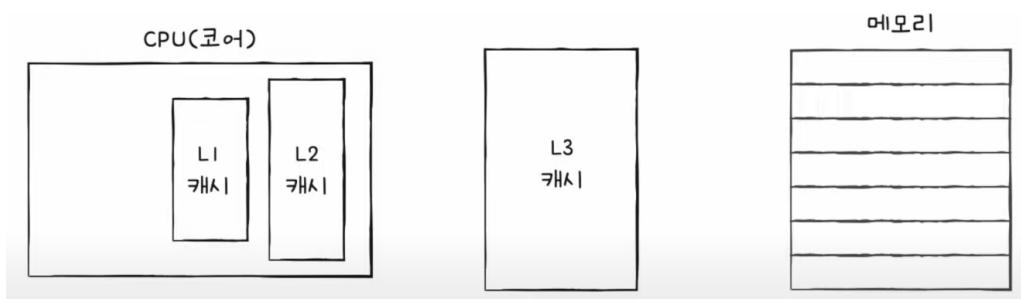
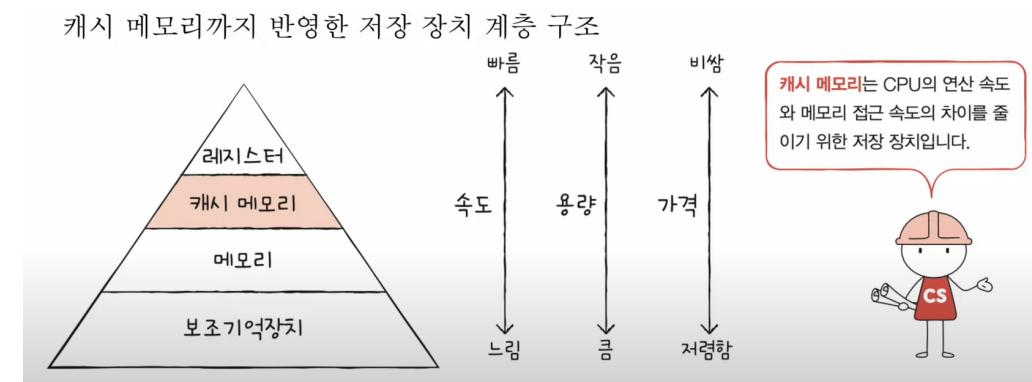
베이스 레지스터는 CPU가 명령하면 본인이 가진 체계 기준(15000)대로 이해하고 메모리와 송수신

⁵ 대역폭(data rate) : 데이터를 주고받는 길의 넓이

- 베이스 레지스터 : 프로그램의 첫 물리 주소를 저장
- 한계 레지스터 : 논리 주소의 최대 크기, 정해진 구역(A,B,C)를 벗어나는 명령은 미실행
 - 물리 주소의 범위 : 베이스 레지스터 값 이상, 베이스 레지스터 값 + 한계 레지스터 값 미만
 - CPU는 메모리에 접근하기 전에 접근하고자 하는 논리 주소가 한계 레지스터보다 작을지를 검사하며, 높다면 인터럽트(트랩)이 발생



- 캐시메모리 : CPU의 연산속도와 메모리의 접근 속도를 줄이기 위한 저장 장치
 - CPU와 메모리사이, 레지스터보다 큰 용량, 메모리보다 빠른 SRAM 기반
 - CPU의 연산속도는 메모리에서 데이터를 호출하는 것 보다 빠르기에 유익 시간이 발생
 - 참조 지역성의 원리에 따라 캐시 히트를 발생시켜 캐시 적중률을 높임
- 저장 장치 계층 구조(memory hierarchy) : 각 다른 용량과 성능의 저장 장치를 계층화
 - CPU와 얼마나 가까운가를 기준으로한 저장장치
 - 용량 : L1 < L2 < L3, 속도 L1 > L2 > L3
 - 분리형 캐시 : L1 - 명령어. 데이터만 저장하면 이를 L1d, L1d 캐시로 분리해 저장함



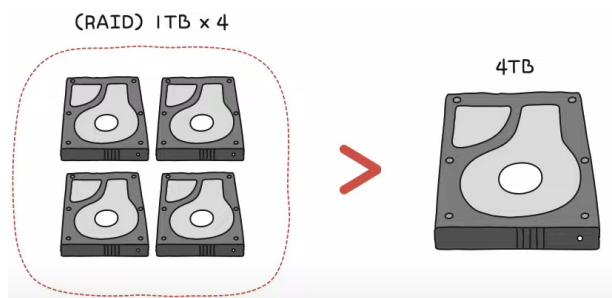
- 캐시 히트(cashe hit) : 메모리는 보조기억장치의 일부를, 캐시메모리는 CPU가 사용할 데이터를 예측해 저장함 <-> 캐시 미스
 - 캐시 적중률 : 높을 수록 메모리 접근 횟수 감소
 - 캐시 히트 횟수 / (캐시 히트 횟수 + 캐시 미스 횟수)
- 참조 지역성의 원리(principle of locality) : 시공간 지역성
 - 시간 지역성(temporal locality) : 최근 접근한 메모리 공간에 재접근 성향
 - 공간 지역성(spatial locality) : 접근한 공간 근처로 접근하려는 성향

Chapter 7. 보조기억장치

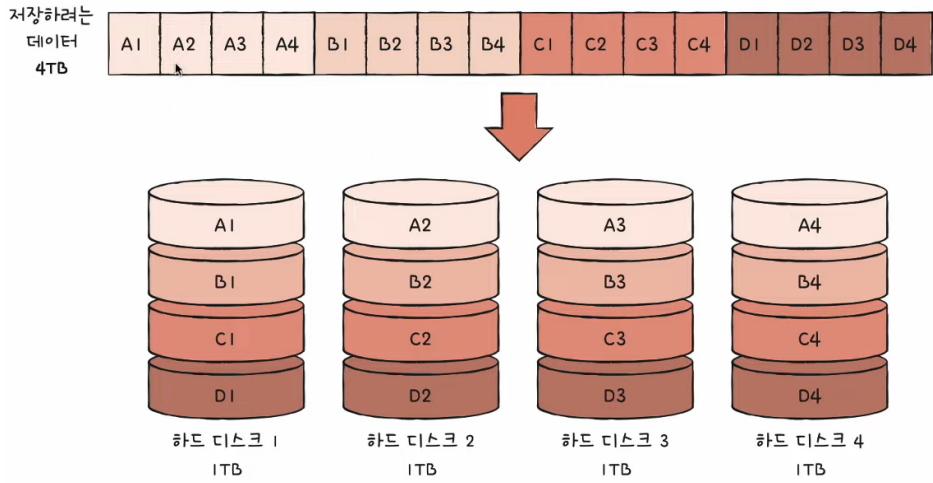
- 보조 기억장치(Secondary Memory, Auxiliary Memory) : CPU는 직접 접근불가
 - HDD(Hard Disk Driver)
 - 플래시 메모리 : SSD(Solid State Drive), USB, SD카드
 - CD-ROM등 CPU는 직접 접근불가
- HDD : 자기 디스크(magnetic disk)
 - 헤드 : 플래터를 대상으로 읽고 쓰며 원하는 위치로 이동하는 디스크암이 부착됨
 - 플래터(platter) : 원판이 자기 물질로 덮여 수 많은 N극과 S극이 0과 1을 수행
 - 여러겹의 플래터가 존재하며, 스피드들이 위아래로 달음
 - 스피드(spindle) : 분당 회전수(RPM, revolution per minute)로 플래터를 돌림
 - 15,000/minute
 - 데이터 접근시간 : 탐색 시간, 회전 지연, 전송시간 등 상세 내용은 불필요하므로 생략
- 플래시 메모리 : NAND, NOR 연산을 수행하는 회로 혹은 게이트 기반
 - 셀 : 플래시 메모리에서 데이터를 저장하는 가장 작은 단위

구분	SLC	MLC	TLC
셀당 bit	1bit	2bit	3bit
수명	길다	보통	짧다
읽기/쓰기 속도	빠르다	보통	느리다
용량 대비 가격	높다	보통	낮다

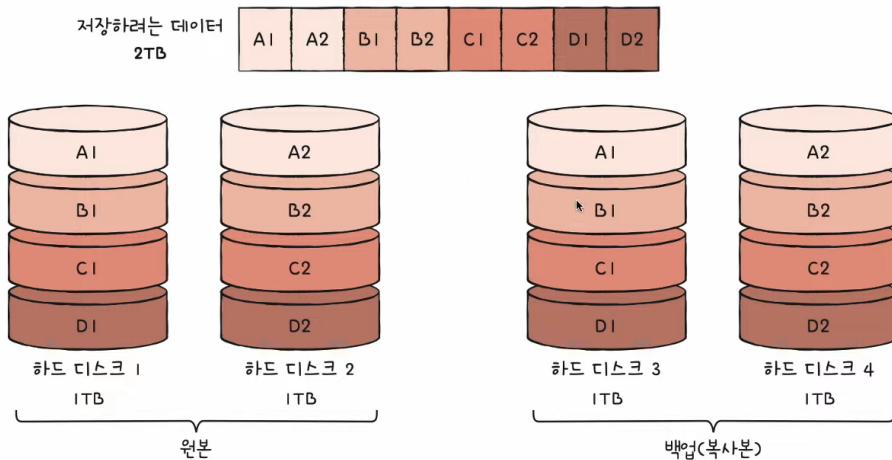
- 한 셀에 1비트 = Single Level Cell, 2 = Multiple, 3 = TripleLC, 4=Q
- 단위 : 셀 < 페이지 < 블록 < 플레이인 < 데이
 - 읽기와 쓰기는 페이지 단위, 삭제는 블록 단위
- 상태 : Free, Valid(유효 데이터 저장중), Invalid(쓰레기 데이터 보유중)
 - 플래시는 덮어쓰기 불가능
 - 이를 해결하기 위해 SSD는 garbage collection 기능으로 보완
- RAID(Redundant Array of Independent Disks) : HDD, SSD 주로 사용, 여러개의 물리적 보조장치를 논리적으로 하나로 사용해서 안전성 증가



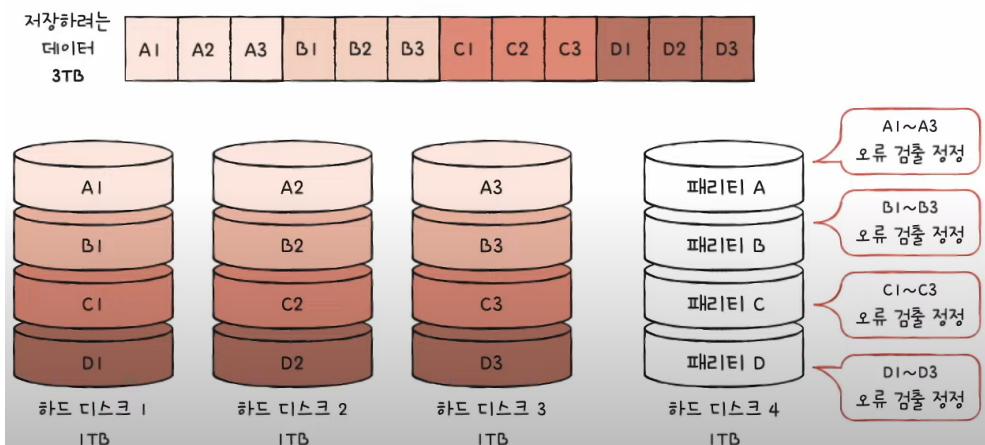
- RAID 0 : 1TB * 4 개(병렬분산저장), 1/4씩 나누어 저장하는 Stripe 방식, 4TB보다 4배 빠름
 - 단점 : 하나라도 문제시 이슈



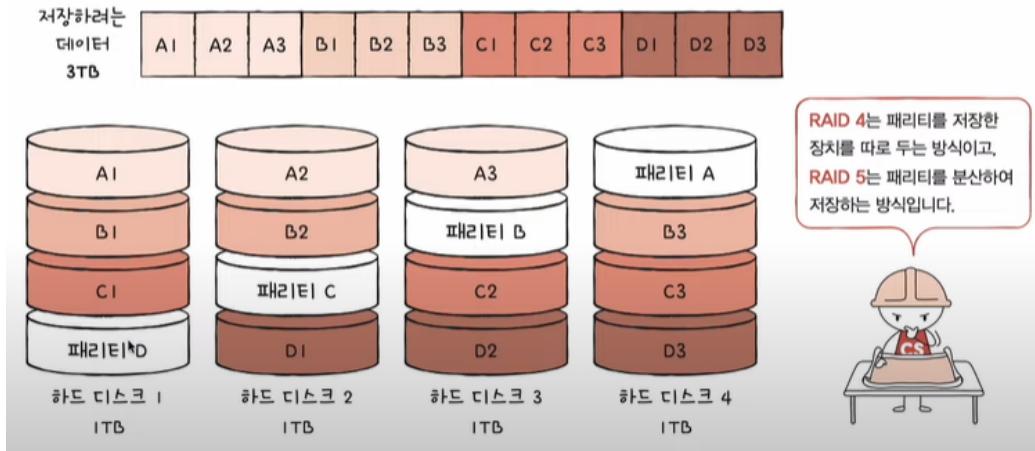
- RAID 1 : 1TB * 2 * 2(미러링방식) , Stripe 방식은 쓰며 복구가 가능
 - 단점 : 가용량 절반, 비용증가



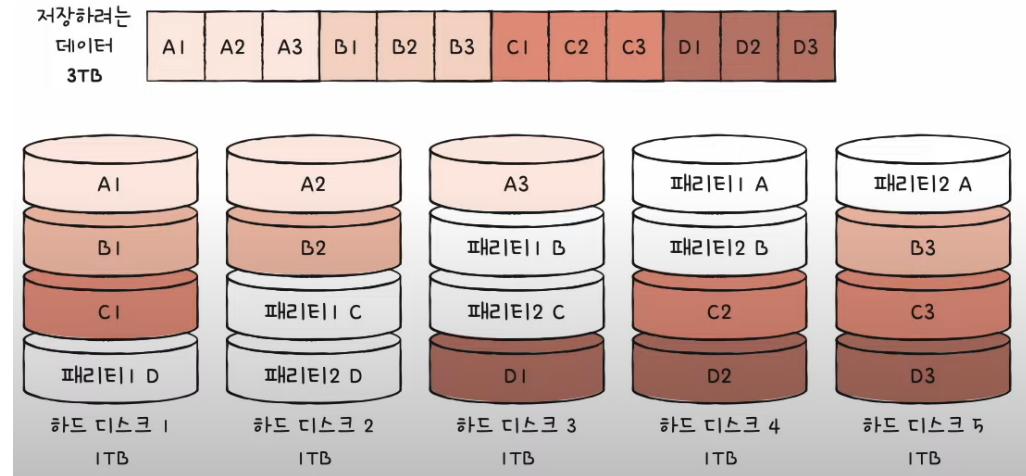
- RAID 4 : 1TB * 3 * 1(백업), 패리티 비트(parity bit), 백업용으로 검출, 복구 가능
 - 단점 : 패리티에도 이중으로 작성하기에 병목 현상이 발생



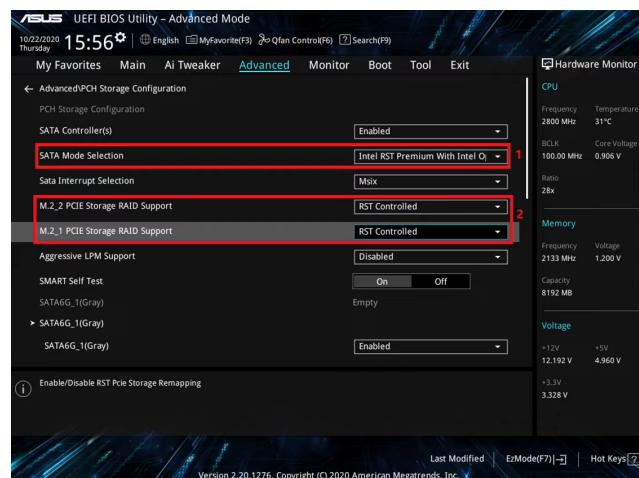
- RAID 5 : 패리티 자체를 분산 저장, 병목 완화



- RAID 6 : 패리티 자체를 분산 저장의 이중화, 5보다 쓰기는 느리나 높은 안전성



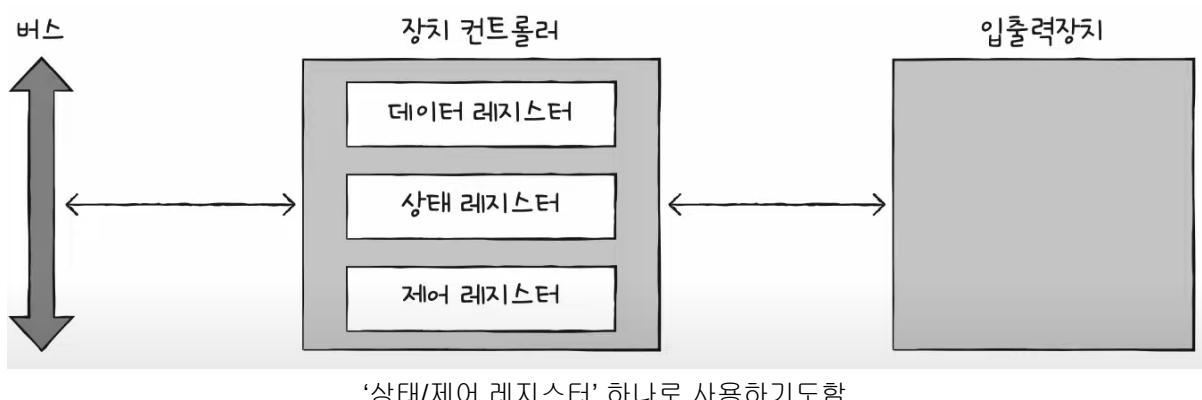
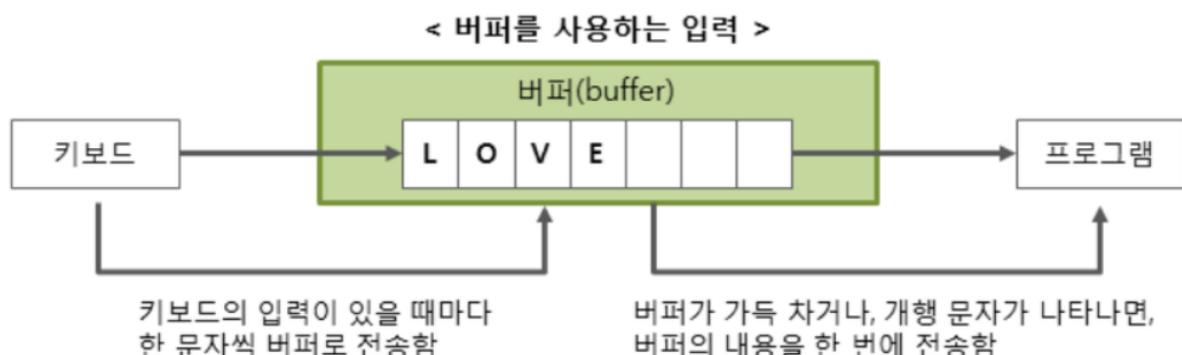
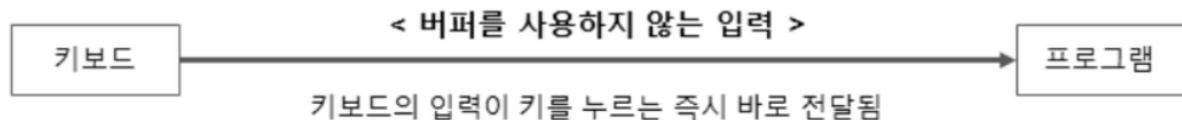
- RAID 10 : RAID 0과 1 혼합
- RAID 50 : RAID 0과 5 혼합
- Nested RAID : 여러 RAID 레벨을 혼합한 방식



BIOS에서 RAID 설정

Chapter 8. 입출력장치

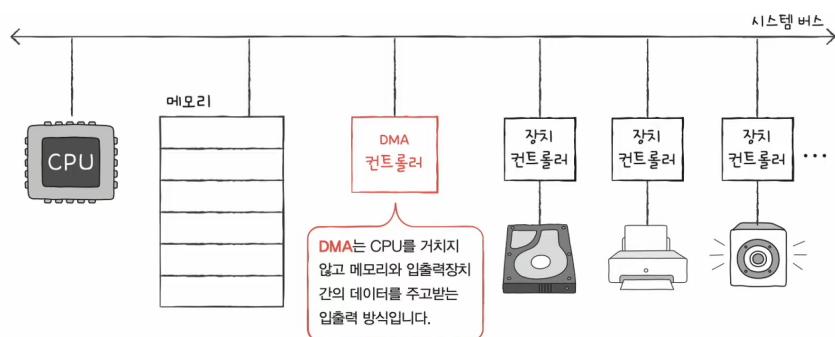
- 입출력 장치 : 장치 컨트롤러라는 하드웨어를 통해 연결되어 컴퓨터 내부와 정보 소통
 - 특징 : 다양한 종류, 낮은 데이터 전송률
 - 입출력 제어기, 입출력 모듈등의 이름으로도 불림
 - 키보드, 모니터, 마우스, USB, 스피커, 마이크등
- 장치 컨트롤러(device controller) : CPU와 입출력장치간의 통신 중개, 오류 검출, 데이터 버퍼링
- 버퍼링(buffering) : 전송률이 높은 장치와 낮은 장치 사이에 주고 받는 데이터를 버퍼라는 임시 저장 공간에 저장하여 전송률을 비슷하게 맞추는 방법



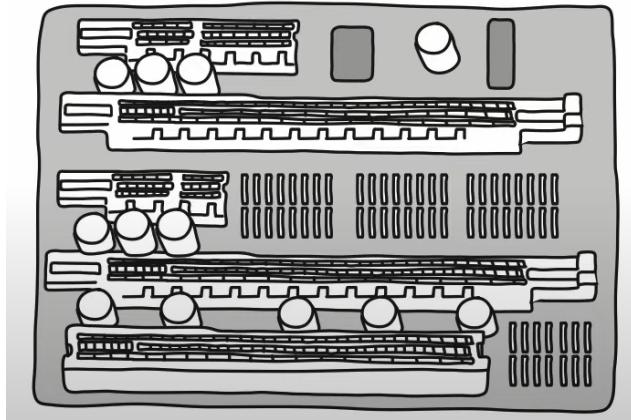
- 데이터 레지스터 : CPU와 입출력장치 사이에 주고 받을 데이터가 저장되는 곳, 버퍼 역할
 - 주고받는 데이터가 많은 입출력장치에서는 레지스터 대신 RAM을 사용하기도 함
- 상태 레지스터 : 입출력 장치의 작업준비 상태, 작업 완료, 오류등의 상태 정보가 저장
- 제어 레지스터 : 입출력 장치가 수행할 내용에 대한 제어 정보와 명령을 저장
 - 이 레지스터들은 버스를 타고 입출력장치로 전달되며, 장치컨트롤러를 통해 입출력 장치로 전달
- 장치 드라이버 : 장치 컨트롤러의 동작을 감지, 제어, 정보 송수신을 돋는 프로그램
 - 프로그램이기에 실행 과정에서 메모리에 저장
 - 장치 컨트롤러는 하드웨어적 통로, 장치 드라이버는 소프트웨어적 통로

- 장치 컨트롤러가 CPU와 정보를 송수신하는 방법 : 프로그램, 인터럽트 기반, DMA 입출력
 - 프로그램 입출력(program I/O) : 프로그램 속 명령어로 입출력 장치를 제어하는 방법
 - CPU 가 하드 디스크 컨트롤러의 제어 레지스터에 쓰기 명령어 보냄
 - 하드 디스크 컨트롤러는 하드 디스크의 상태 확인, 상태 레지스터에 준비완료 표시
 - CPU는 상태 레지스터를 주기적 확인, 데이터 레지스터에 데이터 저장
 - 쓰기(백업)이 미완료시 1번부터 반복하며 끝나면 작업종료
 - CPU가 각 장치 컨트롤러 속 레지스터의 주소를 아는 방법 : 메모리 맵, 고립형 입출력
 - 메모리 맵 입출력(memory-mapped I/O) : 메모리에 접근하기 위한 주소 공간과 입출력 장치에 접근하기 위한 주소 공간을 하나의 주소 공간으로 간주
 - 고립형 입출력(isolated I/O) : 메모리를 위한 주소 공간과 입출력 장치를 위한 주소 공간 분리
-
- | 메모리 맵 입출력 | 고립형 입출력 |
|--------------------------|--------------------------|
| 메모리와 입출력장치는 같은 주소 공간 사용 | 메모리와 입출력장치는 분리된 주소 공간 사용 |
| 메모리 주소 공간이 축소됨 | 메모리 주소 공간이 축소되지 않음 |
| 메모리와 입출력장치에 같은 명령어 사용 가능 | 입출력 전용 명령어 사용 |

- 인터럽트 기반 입출력(interrupt-Driven I/O) : CPU가 장치 컨트롤러에 명령어를 전달하면 다른 작업을 수행할 수 있고, 장치 컨트롤러가 해당 작업을 마치고 인터럽트 요청(완료신호)를 보내면 CPU는 기존 작업을 백업하고 인터럽트 서비스 루틴을 실행하는 방식
 - 여러 입출력 장치로부터 인터럽트가 몰려오는 경우 CPU는 우선 순위 처리 가능
 - NMI(Non-Maskable Interrupt) : 인터럽트 비트 비활성화도 무시 불가, 최우선 인터럽트
 - 프로그래머블 인터럽트 컨트롤러(PIC, Programmable Interrupt controller) : 다중 인터럽트를 처리하는 하드웨어로 CPU가 우선 처리할 순서를 알려주는 장치
 - PCI는 인터럽트 비트까지가 유효하게 발동
- DMA(Direct Memory Access) 입출력 : 앞선 두 입출력 방식은 입출력 장치와 메모리가 CPU를 거쳐 상호작용해야 하나 DMA는 직접 메모리에 접근하는 입출력 방식
 - 시스템 버스에 연결된 DMA 컨트롤러가 CPU에게 정보를 받아 완료시 보고



- 사이클 스태リング(cycle stealing) : CPU와 DMA 컨트롤러가 시스템 버스 동시사용불가, CPU에게 허락을 받는 방식
- 입출력 버스 : DMA가 잡은 시스템 버스 사용시의 문제가 되므로 입출력버스를 추가해 각 장치 컨트롤러와 소통
 - PCI 버스, PCI Express(PCIe) 버스등



입출력 장치들을 PCIe 버스와 연결해주는 PCIe 슬롯의 모습



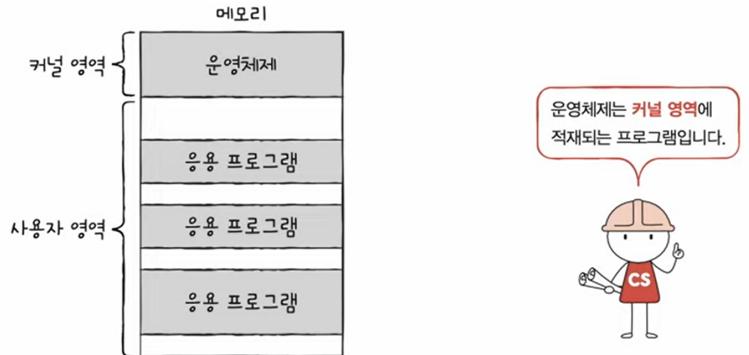
- PCIe 버스가 있어도 여전히 인출, 해석, 실행의 상당부분은 CPU의 몫
- 입출력 전용 CPU가 있는 컴퓨터는 DMA 컨트롤러 역할
 - = 입출력 프로세서(I/O Processor), 입출력 채널(I/O Channel)⁶

6

<https://velog.io/@jobmania/CH08-%EC%9E%85%EC%B6%9C%EB%A0%A5%EC%9E%A5%EC%B9%98>

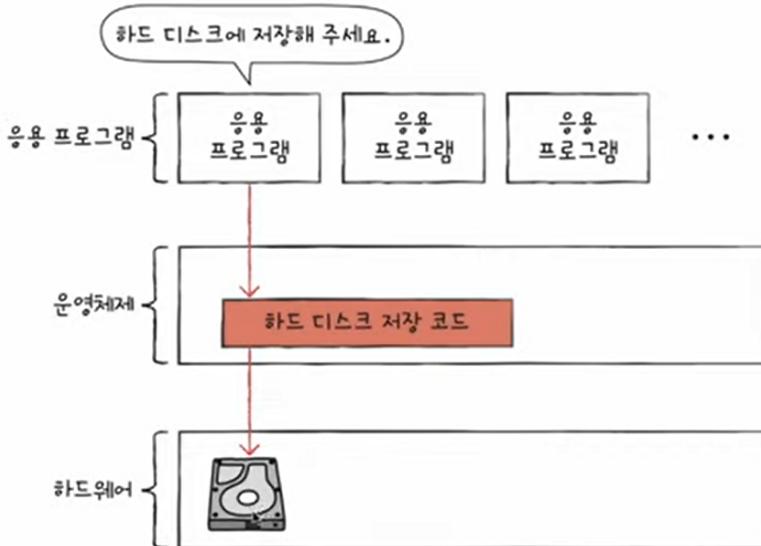
Chapter 9. 운영체제

- 운영체제 : 실행할 프로그램에 필요한 자원을 할당하여 올바른 실행을 돋는 프로그램, 정부
 - 커널 영역(kernel space) : 운영체제의 중요도가 매우 높기에 부팅시 별도의 공간에 적재되어 실행됨
 - 사용자 영역 : 운영체제 이외의 응용 프로그램을 적재하는 공간



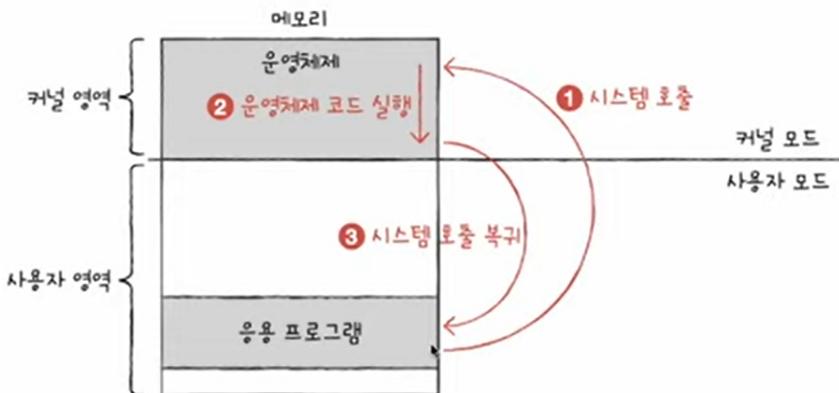
응용 프로그램(application software)?
사용자가 특정 목적을 위해 사용하는 일반적인 프로그램

- 여러 응용 프로그램들이 적재될 메모리 주소 관리
- CPU가 우선 운용될 프로그램 지정, 하드웨어 우선 순위 관리
- CPU가 각 프로그램마다 할당할 자원과 시간 관리
- 대다수 오류 메시지의 근원, 작성한 소스 코드를 하드웨어가 미실행시 OS가 알려줌
- 응용 프로그램들이 사용할 자원에 대한 접근시 OS의 허락이 필요
 - 운영체제에 도움을 요청 = 운영체제 코드를 실행
- 응용 프로그램들의 요청을 받은 OS는 응용프로그램 대신 자원에 접근, 대신 작업 수행



- 커널(kernel) : 운영체제의 핵심 기능, 자원에 접근 및 조작, 프로그램의 올바른 실행, 심장
- 사용자 인터페이스(UI, User Interface) : 사용자와 컴퓨터의 상호작용, 운영체제가 제공
 - 그래픽 유저 인터페이스(GUI, Graphical UI) : 윈도우 바탕화면, 스마트폰
 - 커맨드 라인 인터페이스(CLI, Command Line I) : 명령어 기반
- 운영체제가 제공하는 서비스중 커널에 포함되지 않는 서비스도 있음

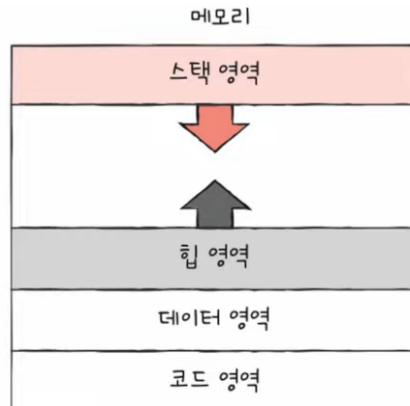
- 이중 모드(dual mode) : CPU가 명령어를 실행시 사용자 모드, 커널 모드로써 실행 가능
 - 사용자 모드 : 운영체제의 서비스를 제공 받을 수 없는 실행 모드
 - 커널 영역의 코드 실행 불가
 - 일반적인 응용 프로그램들의 기본 실행 모드
 - 사용자 모드로 실행중인 CPU는 입출력 명령어(하드웨어 자원 접근) 불가
 - 사용자 모드로 실행중인 일반적인 응용 프로그램도 자원 접근 불가
 - 커널 모드 : 운영체제의 서비스 제공 받을 수 있는 실행 모드
 - 시스템 호출(system call) : 사용자모드에서 커널 모드로 전환, 소프트웨어적 인터럽트



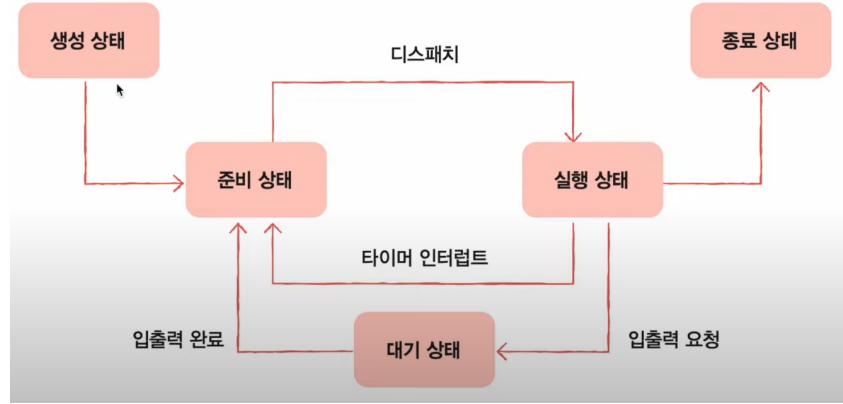
- 운영체제의 핵심 서비스 : 프로세스 관리, 자원 접근 및 할당, 파일 시스템 관리
 - 프로세스 : 실행 중인 프로그램
 - 자원 접근 및 할당
 - CPU : CPU는 하나의 프로세스만 실행하므로 번갈아 실행함
 - 메모리 : 운영체제가 새로운 프로세스를 적재할 주소 결정
 - 입출력장치 : 운영체제가 제공하는 기능으로 커널 영역에 존재, 인터럽트 서비스 루틴 실행
 - 파일 시스템 관리 : 파일의 루트인 디렉토리 관리
- 가상 머신 : 소프트웨어적으로 새로운 운영체제와 응용 프로그램 실행 가능, 사용자모드
 - 하이퍼 바이저 모드 : 별도의 커널 모드가 작용되는 것

Chapter 10. 프로세스와 스레드

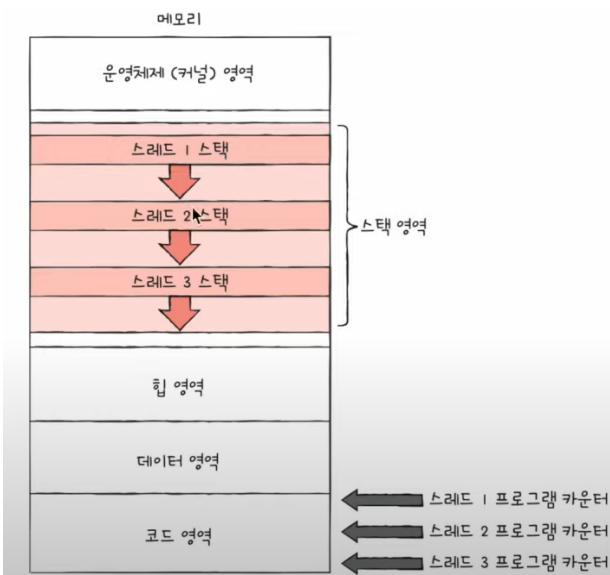
- 포그라운드 프로세스(foreground process) : 사용자에게 보이는 프로세스 <-> 백그라운드
 - 백그라운드 중 유닉스 체계는 데몬, 윈도우 체계는 서비스라고 명명
- 타이머 인터럽트 : 정해진 시간 만큼 CPU를 대기
- 프로세스 제어 블록(PCB) : 플래그처럼 프로세스와 관련된 정보를 저장하는 자료 구조
 - 커널 영역에 생성되었다가 종료시 폐기됨
 - 프로세스 아이디(PID) : 특정 프로세스를 식별하기 위해 부여하는 고유 번호
 - 같은 일을 수행하는 프로세스라도 둘다 실행시 각 아이디가 생성됨
 - 레지스터 값 : 자기 차례가 다가오면 PCB에 저장한 중간값을 호출, 미완료시 저장
 - 프로세스 상태 : 완료, 대기, 사용중인 상태가 PCB에 저장됨
 - CPU 스케줄링 정보 : 언제, 어떤 순서로 CPU를 할당 받을지에 대한 정보 기록
 - 메모리 관리 정보 : 어떤 주소에 저장, 베이스, 한계, 테이블 레지스터 값 저장
 - 사용한 파일과 입출력 장치 목록 : 파일 사용 및 입출력 장치 할당도 기록
- 문맥 교환(context switching) : 기존 프로세스의 문맥을 PCB에 백업, 새 프로세스 문맥을 복구
- 프로세스의 사용자 영역 : 대기중인 프로세스는 사용자 영역에 나뉘어 저장됨
 - 코드 영역(code, text segment) : 기계어로 이루어진 명령어가 저장, 읽기 전용
 - 데이터 영역 : 프로그램이 실행되는 동안 유지할 전역 변수(global variable)
 - 코드와 데이터 영역은 크기가 고정된 정적 할당 영역
 - 힙 영역(heap) : 사용자가 직접 할당할 수 있는 저장 공간, 반환이 지정해주지 않으면 메모리 누수(leak)가 발생
 - 스택 영역 : 데이터 일시 저장 공간, 함수의 실행이 끝나면 사라지는 매개 변수, 지역 변수, 스택영역에 일시 데이터가 push 되고, 삭제시 pop 됨
 - 힙과 스택은 크기가 변동적인 동적 할당 영역
 - 힙은 메모리가 낮은 주소에서 높은 주소에 할당, 스택은 높은에서 낮은으로



- 생성 상태(new) : 방금 메모리에 적재되어 PCB를 할당 받고 준비 상태가 되어 CPU 할당 대기
- 준비 상태(ready) : CPU를 할당 받으면 실행되지만 대기중인 상태
 - 디스패치(dispatch) : 준비에서 실행으로 넘어감
- 실행 상태(running) : CPU 할당 받아 실행 중, 시간 종료시 준비 상태
- 대기 상태(blocked) : 프로세스 실행중 입출력 장치 사용시(이벤트), 입출력 장치 완료까지 기다리고, 완료되면 다시 준비 상태
- 종료 상태(terminated) : 프로세스 종료, 운영체제는 PCB와 프로세스가 사용한 메모리 정리

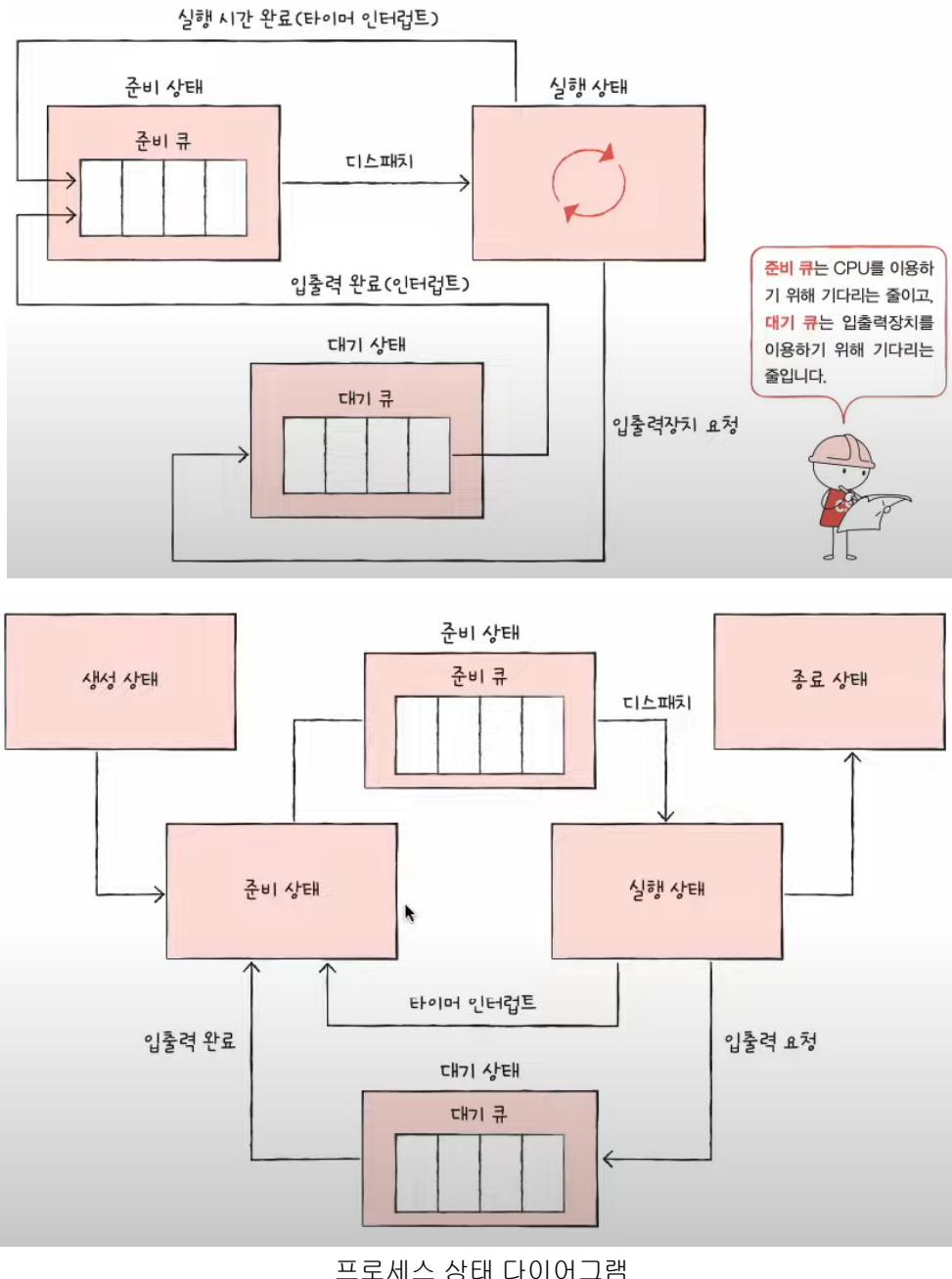


- 프로세스 계층 구조 : 새 프로세스를 생성시 부모와 자식 프로세스가 되는 것
 - 부모 프로세스는 `fork`를 통해 자신의 복사본을 자식 프로세스로 생성, 각종 정보 상속
 - 자식 프로세스는 `exec`를 통해 자신의 메모리 공간을 다른 프로그램으로 교체
- 스레드(thread) : 프로세스를 구성하는 실행의 흐름 단위, 하나의 프로세스가 여러 스레드 실행
 - 프로세스의 스레드들은 실행에 필요한 최소한의 정보만을 유지한채 프로세스 자원을 공유하며 실행됨
 - 리눅스는 프로세스와 스레드를 구분하지 않고 task라고 명명



- 멀티 프로세스 : 프로세스를 동시에 실행
- 멀티 스레드 : 여러 스레드로 프로세스를 동시에 실행
 - 프로세스끼리는 자원 공유 x, 스레드는 자원 공유 o
 - 프로세스를 `fork` 하면 각종 정보 상속되면 메모리에 동일 내용 중복 및 낭비 발생
 - 멀티 스레드는 하나의 스레드가 문제 발생시 다른 스레드도 영향 받음
 - 멀티 프로세스로 해결하거나 프로세스간 통신으로 해결
- 프로세스 간 통신(IPC, Inter-process communication) : 프로세스끼리 자원 및 정보 공유
 - 컴퓨터내의 시스템끼리의 파일 송수신을 통한 통신
- 공유 메모리 : 프로세스들이 서로 공유하는 메모리 영역에서 데이터 송수신
 - 공유 주소에 A가 전역 변수 저장시 B가 호출 가능
- 이외에도 소켓, 파일 등을 통해 프로세스들은 통신이 가능

- 입출력 버스트 : 입출력을 위한 대기 상태가 많은 입출력 집중 프로세스
- CPU 버스트 : 대기 상태보다 실행 상태가 더 많은 CPU 집중 프로세스
- 스케줄링 큐 : CPU, HDD, 입출력장치등 쓰고 싶은 프로세스들마다 출점, 준비 큐, 대기 큐



- 선점형 스케줄링(preemptive) : 실행중인 프로세스의 자원을 빼앗을 수 있음, 비독점, 고른 자원 배분이 가능하나 오버헤드 발생 가능, 대부분의 운영체제가 사용중
- 비선점형(non-preemptive) : 빼앗을 수 없음, 독점
 - 오래 대기하는 호위효과(convoy effect) 발생, 일정 시간만 사용하는 타임 슬라이스, 우선 순위 높은 것만 처리시 기아 현상 발생, 장기 대기시 에이징으로 순위 높임

Chapter 12. 프로세스 동기화

- synchronization : 특정 자원에 한 프로세스만 접근하거나 올바른 순서대로 실행되어야 함
 - 실행 순서 제어 : 프로세스를 올바른 순서대로 실행하기
 - 상호 배제(mutual exclusion) : 동시에 접근해서 안되는 자원에 하나의 프로세스만 허용

프로세스 A	프로세스 B	현재 잔액
잔액을 읽어 들인다		10만 원
읽어 들인 값에서 2만 원을 더한다		10만 원
문맥 교환		10만 원
	잔액을 읽어 들인다	10만 원
	읽어 들인 값에서 5만 원을 더한다	10만 원
	문맥 교환	10만 원
더한 값 저장		12만 원
	더한 값 저장	15만 원
		최종 잔액 = 15만 원?

잔액이라는 자원을 상호 배제가 안될 시 잘못 계산되는 경우

- 공유 자원(shared resource) : 동시에 접근 가능한 자원
 - 전역 변수, 파일, 입출력장치, 보조기억장치등 가능
 - 두 개 이상의 프로세스를 동시에 실행시 의도치 않은 문제가 발생 가능
- 임계 구역(critical resource) : 동시에 접근시 문제가 발생하는 자원에 접근하는 코드 영역
 - 두 개 이상의 프로세스가 임계 구역에 진입시 둘중하나는 대기하지 않으면 문제
- 레이스 컨디션(race condition) : 임계 구역에서 동시에 다발적으로 코드 실행하는 경우
 - 1줄의 고급언어는 3줄의 저급언어로 실행되기에 문맥교환시 문제 발생 주의

프로세스 A	프로세스 B	현재 총합	현재 r1	현재 r2
$r1 = \text{총합}$		10	10	
$r1 = r1 + 1$		10	11	
문맥 교환		10	11	
	$r2 = \text{총합}$	10	11	10
	$r2 = r2 - 1$	10	11	9
	문맥 교환	10	11	9
$\text{총합} = r1$		11	11	9
문맥 교환		11	11	9
	$\text{총합} = r2$	9	11	9
				최종 총합 = 9

- 상호 배제를 위한 동기화 세가지 원칙
 - 상호 배제(mutual exclusion) : 한 프로세스가 임계구역 진입시 타 프로세스 진입불가
 - 진행(progress) : 임계구역에 어떤 프로세스가 미진입시 진입 허용
 - 유한대기(bounded waiting) : 한 프로세스가 임계구역 진입시 언젠가는 허용해야 함

- 유텍스락(mutex lock, MUTual Exclusion lock) : 상호 배제를 위한 동기화 장치
 - lock : 자물쇠 역할, 프로세스들이 공유하는 전역 변수
 - acquire : 임계구역을 잠그는 역할
 - 프로세스가 임계 구역에 진입전에 호출하는 함수
 - 열릴때 까지 대기 및 확인 반복(busy wait 상태라고 함)
 - 열려있으면 잠금(lock 를 true로)
 - release : 임계구역의 잠금을 해제
 - 작업 끝나면 임계 구역을 오픈(lock를 false로)

```

public class MutexExample {
    private static Boolean lock = false; // 공유하는 락 객체

    void acquire() {
        synchronized (lock) { // synchronized 블록으로 임계 구역에 진입
            // 임계 구역에 들어가기 전에 필요한 초기화나 확인 작업 수행

            // 락이 해제될 때까지 기다림 (lock이 false일 때)
            while (lock) {

            }
        }
        lock = true; // 락을 획득
    }

    void release() {
        synchronized (lock) {
            // 임계 구역을 빠져나간 후에 필요한 정리 작업 수행
            lock = false; // 락을 해제

        }
    }
}
}

acquire(); // 잠겨있는지 확인, 잠겨있지 않다면 잠그고 들어가기
// 임계구역 진입 ( 이전 장의 '총합' 변수 접근)
release(); // 자물쇠 반환!

```

- 바쁜 대기 (busy waiting) : 실제로는 c/c++, python 사용자가 직접 구현하지 않아도 유텍스락 기능 제공, 보다 더 정교하게 설계되어있음. 실제로 구현할 일이 별로 없음..

```

// 락이 해제될 때까지 기다림 (lock이 false일 때)
while (lock) {
    try {
        lock.wait(); // 락을 해제하고 다른 스레드로부터의 알림을 기다림
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

- 세마포(semaphore) : 상호 배제를 위한 동기화 장치, 공유장치 여러개에도 적용 가능
 - 전역변수 S : 임계 구역에 진입할 수 있는 프로세스 개수(사용가능한 공유자원수)
 - wait 함수 or P : 진입 or 대기여부를 알려줌
 - signal 함수 or V : 진입 허용
 - busy wait 방지위해 PCB를 세마포를 위한 대기 큐에 넣음

```
wait()
// 임계구역
signal()
```

```
wait(){ // 만일 임계 구역에 진입할 수 있는 프로세스 갯수가 0 이하라면
    while ( S <= 0 ){
        } // 지속 확인
    S--; // 임계 구역에 진입할 수 있는 프로세스 갯수가 하나 이상이면 S를 1 감소시키고 임계 구역 진입!
}
```

```
signal(){
    S++ // 임계구역 작업을 끝낸후 S 를 1 증가
}
```

- 바쁜 대기 해결 방법 : 그래서 실제로는 $S \leq 0$ 일 경우, 즉 사용할 수 있는 자원이 없는 경우 대기상태로 만든다.(해당 프로세스의 PCB를 대기 큐에 삽입 시킴)

이후, 자원이 생겼을 때 대기 큐의 프로세슬르 준비상태로 만듬.

(해당 프로세스 PCB를 대기큐에서 꺼내 준비 큐에 삽입)

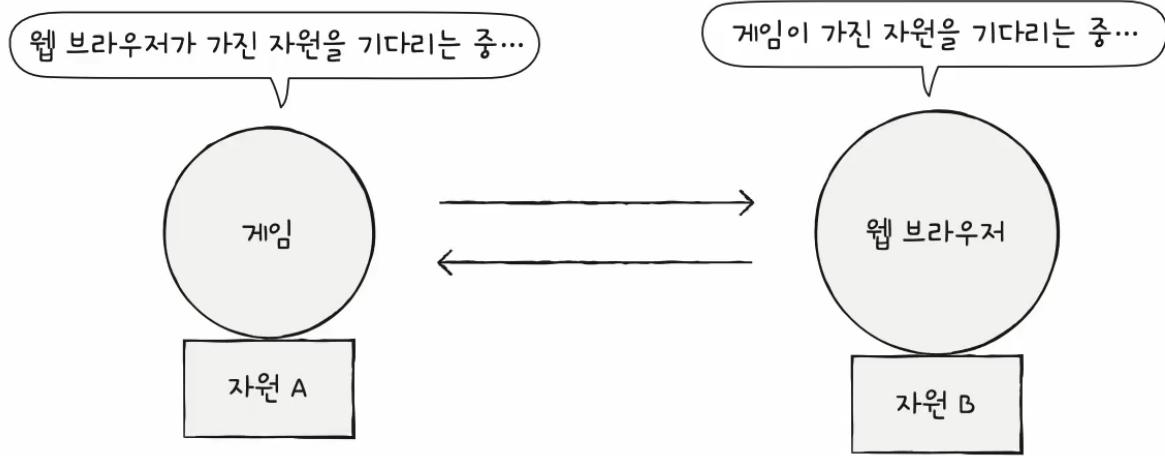
```
void wait(Process p) {
    S--;
    if (S < 0 ) {
        addToQueue(p); // add this process to Queue, 해당 프로세스를 대기큐에 삽입
        sleep(); // 대기상태로 만든다.
    }
}

void signal(Process p) {
    S++;
    if (S <= 0 ) {
        removeQueue(); // 대기큐에서 제거.
        wakeup(p); // 프로세스를 준비큐로 옮긴다.
    }
}
```

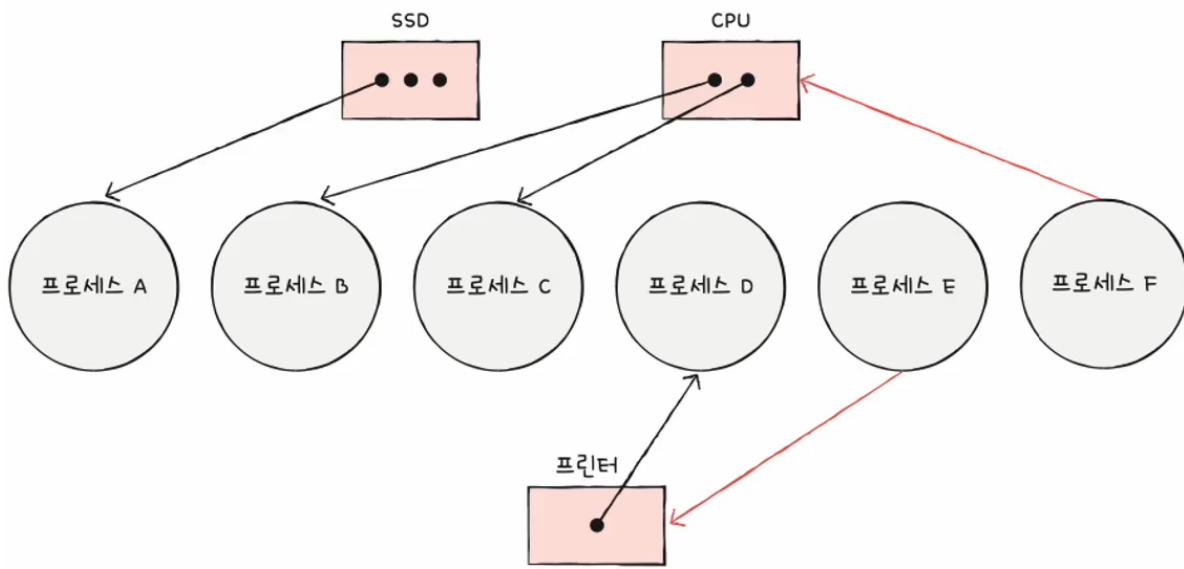
- 모니터(monitor) : 세마포에 비해 사용자가 사용하기 편리한 동기화 장치
 - 인터페이스만을 통해 공유자원에 접근 가능
 - 모니터에 진입하기 위한 큐 관리

Chapter 13. 교착 상태

- 교착 상태(deadlock) : 일어나지 않을 사건을 기다리며 진행이 멈춤



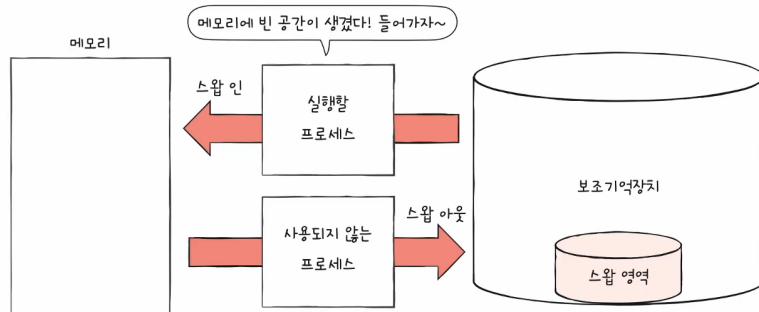
- 상호배제, 점유와 대기, 비선점, 원형 대기가 모두 만족시 교착 상태 발생
- 자원할당그래프(resource-allocation graph) : 사용 자원, 대기 자원을 표현



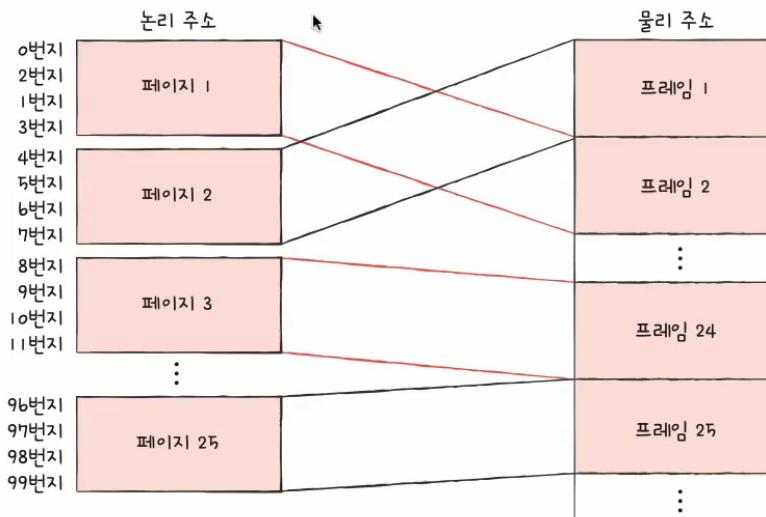
- 예방 : 위 조건 4개중 하나를 충족 시킴, 충족시도시 어떠한 방법을 써도 비효율, 단점 극명함
- 회피 : 안전 상태를 유지하는 경우에만 자원 할당
- 교착 상태의 검출 후 회복 : 프로세스에게 모든 자원할당, 교착상태 주기적 검사, 발생시 회복
 - 선점을 통한 회복 : 한 프로세스만 몰빵
 - 강제종료를 통한 회복 : 모두 or 하나씩 프로세스를 종료, 오버헤드 문제
- 무시 : 타조 알고리즘(ostrich algorithm)

Chapter 14. 가상 메모리

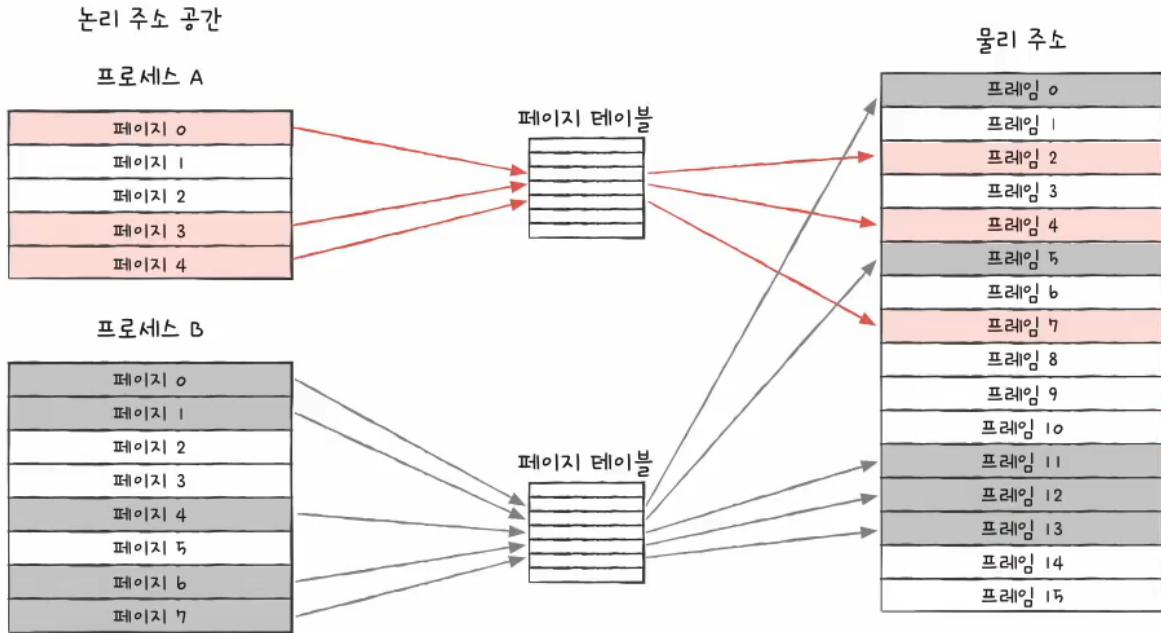
- 스와핑(swapping) : 미실행된, 끝 상태의 프로세스를 보조기억장치로 보내 다른 프로세스 적재
 - 유닉스, 리눅스, MacOS에서 free, top 명령어를 통해 스왑 영역의 크기 확인 가능
 - 크기와 사용 여부는 사용자가 임의 설정 가능



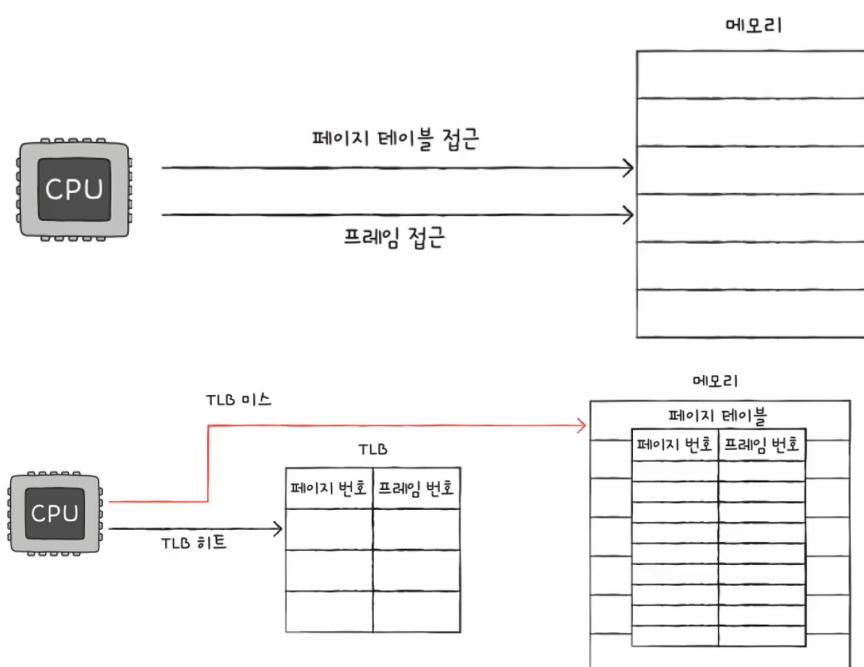
- 메모리 할당 : 랜덤하게 빈 공간에 프로세스를 연속적으로 할당하는 방식
 - 최초 적합(first fit) : 순서대로 검색후 적재, 검색 최소화, 빠른 할당 가능
 - 최적 적합(best fit) : 모든 빈 공간중 가장 작은(요구치와 동일한) 공간에 할당
 - 최악 적합(worst fit) : 모든 빈 공간중 가장 큰 공간에 할당
- 외부 단편화(external fragmentation) : 연속 메모리 할당의 문제, 빈 공간을 합하면 적재되지만 애매하게 나누어져 있어서 적재 불가
- 압축(compaction) : 메모리 조각 모음
 - 외부 단편화 해결 가능하나 시스템 중지 필요, 옮기는 과정의 오버헤드 야기
- 가상메모리 : 실행하고자 하는 프로그램의 일부만 메모리에 적재하여 실제 물리 메모리 크기보다 더 큰 프로세스를 실행
- 페이지 : 메모리의 물리 주소 공간을 프레임 단위로 자르고, 프로세스의 논리 주소 공간을 페이지 단위로 자른뒤 각 페이지를 프레임에 할당, 현재 대부분의 운영체제가 사용
 - 프로세스 실행을 위해 프로세스 전체가 메모리에 적재되지 않아도 가능



- 페이지 테이블 : 프로세스가 불연속적으로 배치되더라도 논리 주소에서 연속적으로 배치되도록 일정의 이정표를 제공



- 페이지 테이블 베이스 레지스터(PTBR) : CPU내 PTBR은 각 프로세스의 페이지 테이블이 적재된 주소를 가리키고 있음, PCB에 기록됨
- 내부 단편화 문제 : 메모리 낭비가 일어날수도 있으나 설정보다 더 큰 페이지를 허용하는 대형 페이지를 통해 해결
- TBL(Translation Lookaside Buffer) : CPU가 메모리 접근 시간 두배(페이지 테이블 접근, 프레임 접근)로 증가를 막기 위해 캐시 메모리를 둠
 - CPU에서 발생한 논리 주소에 대한 페이지 번호 있으면 히트 or 미스

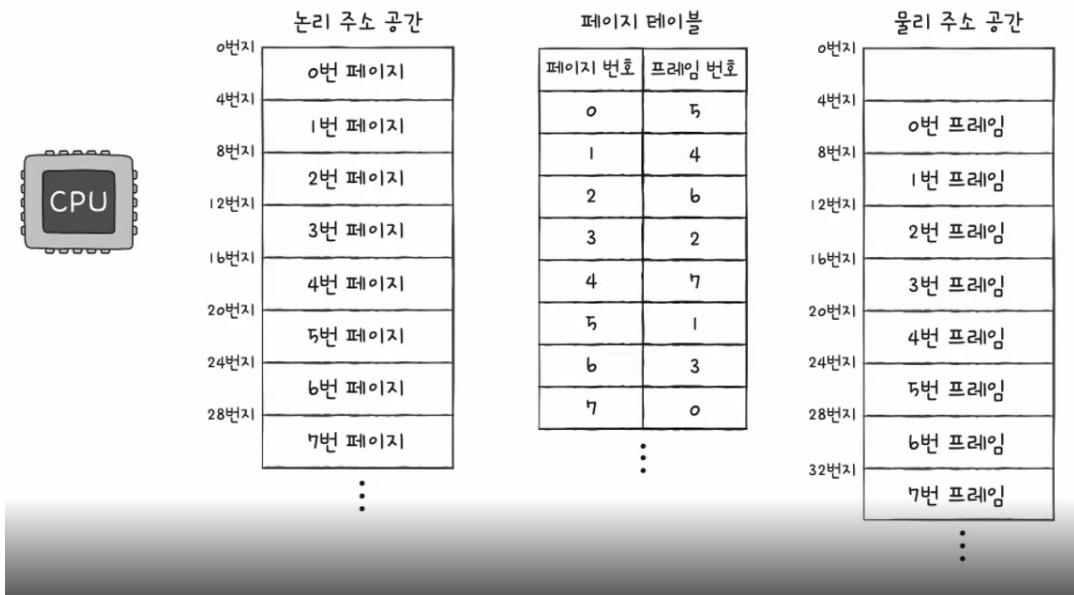


- 페이지에서의 주소 변환 : 페이지 번호와 변위를 통해 물리 주소가 물리 주소로 변환
 - 페이지 번호(page num) : 접근하고자 하는 페이지나 프레임
 - 변위(offset) : 해당 페이지나 프레임과 주소까지의 거리



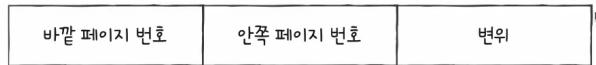
CPU가 32bit 의 주소를 보냈다면, 페이지 번호 + 변위 = 32bit

- 논리 주소의 페이지 번호 및 변위는 페이지 테이블에 의해 물리 주소의 프레임 번호, 변위로 변환되며 같은 동일



- CPU가 5번 페이지 및 변위 2 접근시
 - 페이지 테이블에 의해 5번 페이지의 프레임 번호는 1
 - 물리 주소 공간의 1번 프레임은 8 ~ 12 번지 사이, CPU는 10번지에 근접하게됨
- 페이지 테이블 엔트리 : 행(엔트리)에는 유효 비트, 보호 비트, 참조 비트, 수정 비트가 담김
 - 유효 비트(Valid Bit)** : 접근 가능 여부
 - 페이지가 메모리에 적재되면 유효 비트 1, 아니면 유효 비트 0
 - 보호 비트(protection bit)** : RWX 가능여부를 0과 1로 표현
 - 참조 비트(reference bit)** : CPU가 페이지에 접근한 유무를 확인
 - 수정 비트(modified bit)** : 데이터를 쓴 적이 있는지의 수정 여부를 확인
 - = 더티 비트(dirty bit) : 1 이면 변경, 0 이면 미접근 or 읽기만
 - 페이지가 메모리에서 사라질 때 보조기억장치에 R 작업 필요성 여부 확인
 - R만 한 페이지라면 메모리에 저장된 값과 보조기억장치 값이 동일(덮어쓰기)만 진행
 - W 작업을 수행한 경우 보조기억장치와 값이 다름(R 작업 필요)
- 페이지 폴트 : 유효 비트 0 일때 접근시, 프로세스가 가용할 프레임 수가 적을 때 발생하는 예외(exception)

- 계층적 페이징 : =다단계 페이지 테이블, 페이지 테이블 자체도 페이징



- 요구 페이징 : 필요한 페이지만을 유효 비트가 0이라도 1로 바꾸고 메모리에 적재
 - 페이지 교체와 프레임 할당이 안정적이어야 함
 - 유효 비트가 0 일 경우 페이지 폴트, 보조기억장치에서 가져오므로 메모리보다 느림
- 페이지 교체 알고리즘 : 요구 페이징 남발시에 발생할 페이지 폴트 방지
 - 페이지 참조열(접근 가능한 중복 페이지 제거)을 통해 페이지 폴트 횟수를 알아야 함
 - FIFO : 가장 먼저 올라온, 오래 머문 페이지 방출
 - 2차 기회 페이지 알고리즘 : 참조 비트로 적재 시간 재설정해 유지함
 - Optimal(최적) : 낮은 사용 빈도의 페이지 방출, 구현 어려움, 성능 평가용
 - LRU(Least Recently Used) : 오랫동안 사용하지 않은 페이지를 교체
- 스레싱(thrashing) : 프로세스의 실행보다 페이징에 소모 시간이 많아 성능저하 발생, 프로세스가 필요로 하는 최소한의 프레임 수가 미보장시 발생
- 프레임 할당
 - 정적 할당
 - 균등 : 각 프로세스마다 프레임 균등 할당
 - 비례 할당 : 프로세스 크기에 비례
 - 단, 프로세스 크기와 필요 프레임이 비례하지 않은 문제
 - 동적 할당
 - 작업 집합(working set) : 프로세스가 참조 페이지 기억, 빈번한 페이지 중지
 - 페이지 폴트 빈도 : 폴트율 상, 하한선을 지정, 발생 빈도에 따라 프레임 할당

- 파일(file) : 보조기억장치에 저장된 관련 정보의 집합, 의미있는 정보의 논리적 단위
 - 구성 : 이름, 실행 정보, 부가 정보(속성 or 메타데이터)
 - 속성 : 유형, 크기, 보호, 생성 날짜, 마지막 접근 날짜, 마지막 수정 날짜, 생성자, 소유자, 위치
 - 파일 유형 : 이름뒤 확장자(extention)를 통해 운영체제가 인식
 - 실행 파일 : exe, com, bin
 - 목적 파일 : obj, o
 - 소스 파일 : c, cpp, cc, java, asm, py
 - 워드 프로세서 파일 : xml, rtf, doc, docx
 - 라이브러리 파일 : lib, a, so, dll
 - 멀티미디어 파일 : mpeg, mov, mp3, mp4, avi
 - 백업/보관 파일 : rar, zip, tar
 - 주체 : 운영체제에 의해서만 실행(생성, 삭제, 열기, 닫기, 읽기, 쓰기)
 - 디렉토리(directory) : 파일의 집합, 관리를 위해 트리 구조로 형성
 - 루트(root) : 최상위 트리 C:\
 - 주체 : 운영체제에 의해서만 실행(생성, 삭제, 열기, 닫기, 읽기)
 - . : 현재 작업중인 디렉토리를 표시
 - .. : 현재 디렉토리의 상위 디렉토리
 - 경로(path) : 디렉토리를 이용해 파일 위치, 파일 이름을 특정 짓는 정보
 - 절대 경로(absolute path) : 루트 디렉토리부터 시작하는 경로, 현재 위치와 상관 없음
 - 상대 경로(relative path) : 현재 디렉토리부터 시작하는 경로
 - 디렉토리 엔트리 : 운영체제는 디렉토리도 특별한 형태의 파일로 간주
 - 운영체제는 디렉토리 테이블로 파일 이름, 위치를 유추할 정보를 저장, 속성을 명시
 - 파일 시스템 : 파일과 디렉토리를 보조기억장치에 저장 및 접근하기 위한 운영체제 내부 프로그램, FAT 파일 시스템과 유닉스 파일 시스템이 있음
 - 섹터 : HDD 의 가장 작은 저장 단위
 - 블록 : 하나 이상의 섹터, 운영체제는 블록 단위로 파일과 디렉토리를 관리
 - 파티셔닝(partitioning) : 저장 장치의 논리 영역을 구획하는 작업
 - 파티션(partition) : 논리적으로 나누어진 영역
 - 포맷팅(formatting) : 관리할 파일의 설정 및 새로운 데이터를 쓸 준비 작업
 - 파일 할당 방법
 - 연속 할당(contiguous allocation) : 순차적 할당, 외부 단편화 문제, 구형 방식
 - 불연속 할당
 - 연결 할당(linked) : 각 블록 일부에 다음 블록의 주소 저장하여 연결
 - C 언어의 포인터 개념과 동일, 외부 단편화 문제 해결
 - 반드시 첫 번째 블록부터 차례대로 읽음, 임의 접근 속도 느림
 - 하드웨어 고장, 오류시 다음 블록 접근 불가
 - 이를 해결하기 위해 FAT 파일 시스템 채용
 - 색인 할당(indexed) : 모든 블록 주소를 색인 블록에 모아 관리
 - 이를 기반한 것이 유닉스 파일 시스템

- FAT 파일 시스템 : 연결 할당의 단점 보완
 - 파일 할당 테이블(FAT, File Allocation Table) : 블록을 주소를 테이블로 모음
 - MS-DOS, USB, SD 카드등 저용량 저장 장치용에 채용
 - FAT 12,16,32등 해당 숫자는 비트(클러스터)수를 의미
 - 예약 영역 / FAT 영역 / 루트 디렉토리 영역 / 데이터 영역 순서로 저장됨
 - 하드 디스크의 시작 부분에 있어서 메모리에 캐시되는 속도가 빨라짐
- 유닉스 파일 시스템 : 색인 블록(i-node, index-node) 기반
 - i-node : 파일 속성 정보와 열다섯 개의 블록 주소가 저장
 - 예약 영역 / i-node 영역 / data 영역 순서로 저장
 - 12개는 직접 블록 주소(직접 블록)를 저장
 - 13번째부터 단일 간접 블록의 주소를 저장(별도의 주소만 저장된 것)
 - 14번째는 이중 간접 블록 주소를 저장(별도의 별도)
 - 15번째는 삼중 간접 블록 주소를 저장
- NF 파일 시스템(NFTS) : 윈도우 운영체제에서 사용, 저널링 기능 지원
- ext 파일 시스템 : 리눅스 운영체제에서 사용, 저널링 기능 지원
- 저널링 파일 시스템(journaling) : 시스템 크래시가 발생시 빠른 복구 방법
 - 작업 직전 파티션의 로그 영역에 수행하는 작업(변경)에 대한 로그를 기록
 - 로그를 남긴 후 작업 수행
 - 작업 완료후 로그 삭제
 - 크래시 발생시 로그만 검사
 - 예전에는 파일 시스템 전체 검사 및 복구
 - 리눅스의 fsck, 윈도우의 scandisk 프로그램
- 마운트 : 저장 장치의 파일 시스템에서 다른 저장 장치의 파일 시스템으로 접근할 수 있도록 파일 시스템을 편입시키는 작업, 운영체제에서 저장 장치를 마운트한다고 표현
 - 트리 구조의 디렉토리를 PC 내에 현존하는 디렉토리의 하위 디렉토리화, 연결
- [The Linux Kernel Archives](#)