

# 컴퓨터의 이해

---

## COMPUTATIONAL THINKING

2017.06.19  
최동훈

# 강사소개 - 최동훈

email : [sapsalddog@gmail.com](mailto:sapsalddog@gmail.com)

linkedin : [linkedin.com/in/sapsalddog](https://www.linkedin.com/in/sapsalddog)

당신의 커리어 전환점 패스트캠퍼스

## 프로젝트

터치터치 틀린그림 찾기 클라이언트

SKT 마이샵 2.0 백오피스 프론트엔드

LG webos TV Alljoyn API 제작 외 다수

## 학력

고려대학교 기계공학과 졸

고려대학교 컴퓨터학과 졸



## 번역

Unity 5.x Game AI Programming Cookbook

Building an RPG with Unity 5.x

Learning Functional Data Structures And Algorithms

2017.06.19  
최동훈

# 1시간 동안 운영체제 만들어보기

2017.06.19  
최동훈

## 바이너리 에디터

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

# 1. 바이너리 에디터 사용해보기

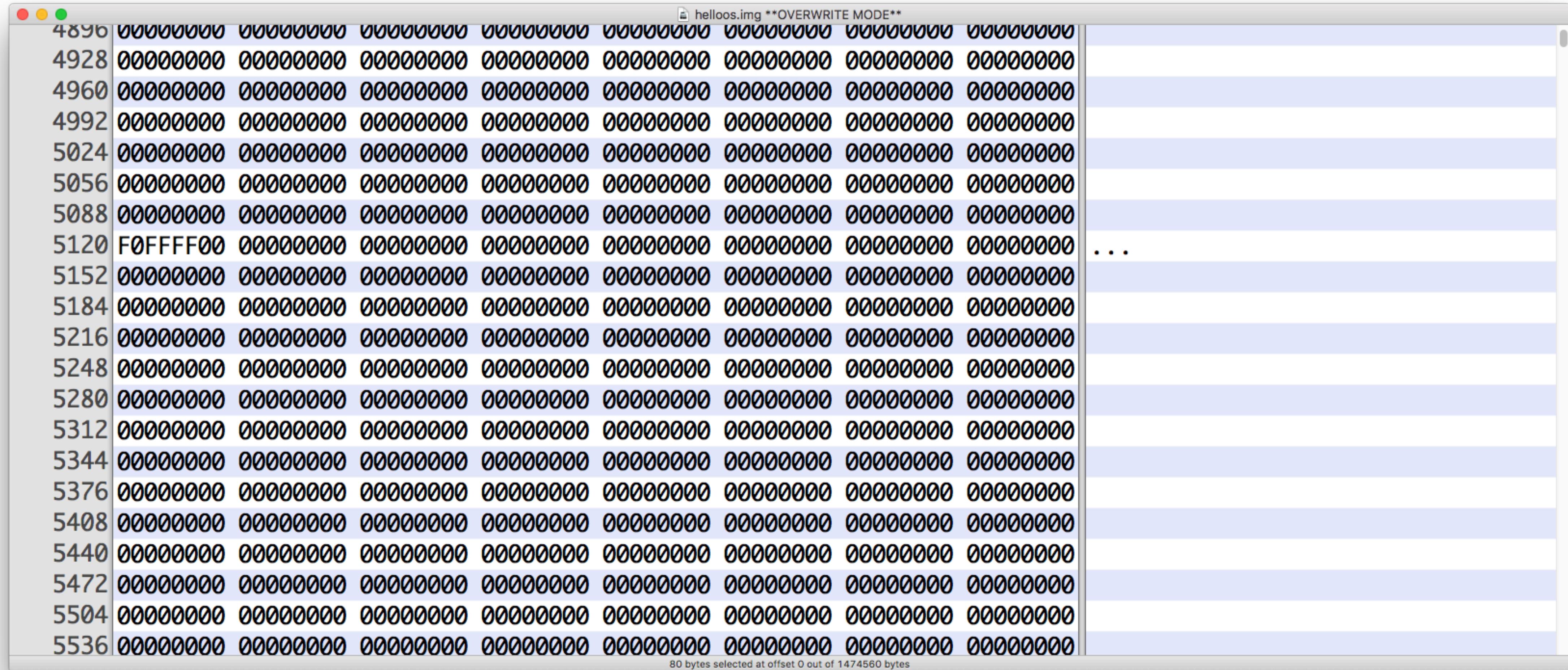
macOS - iHex (앱스토어)

windows - HxD (<https://mh-nexus.de/en/hxd/>)

2017.06.19  
최동훈



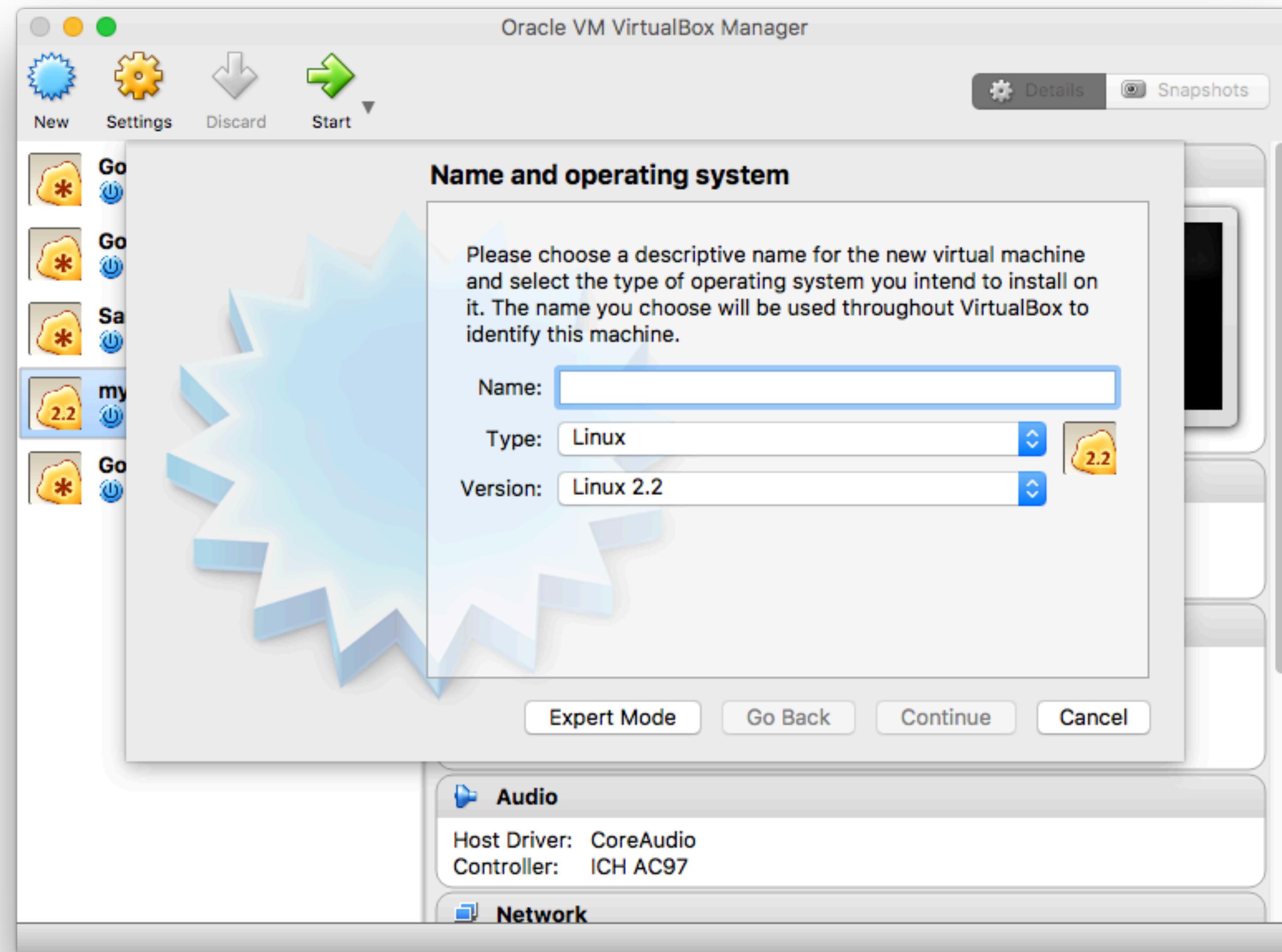
2017.06.19  
최동훈

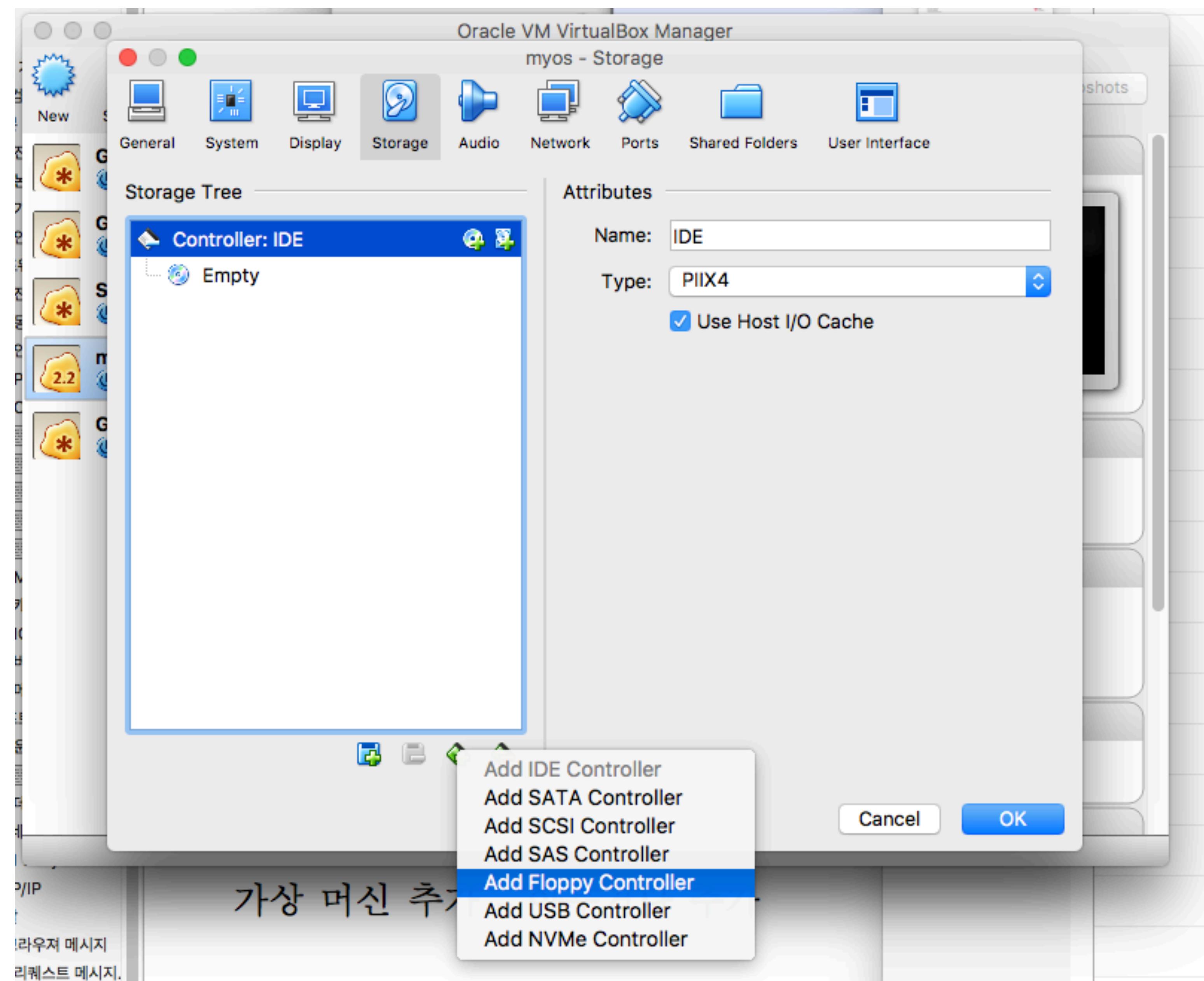


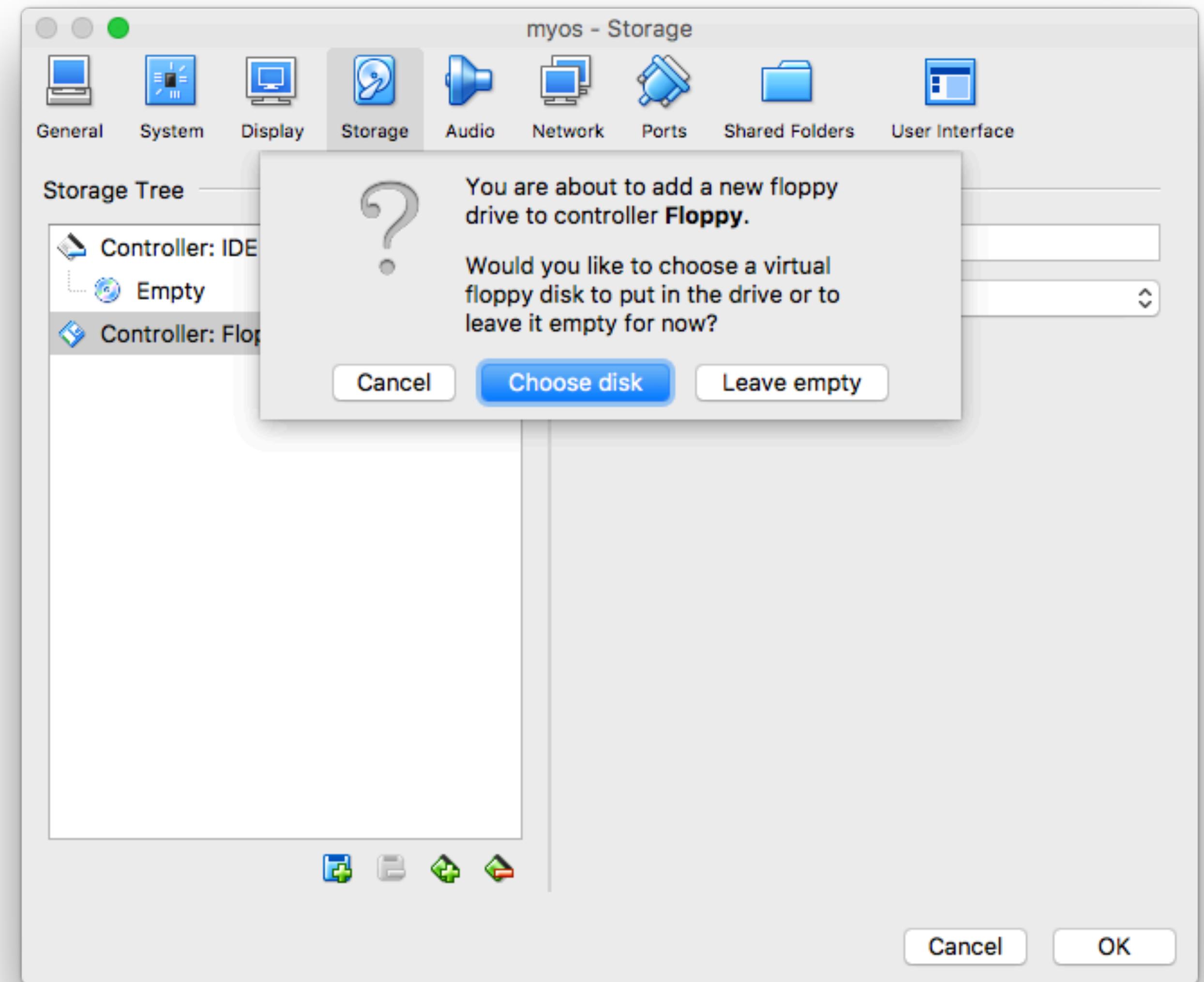
당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈







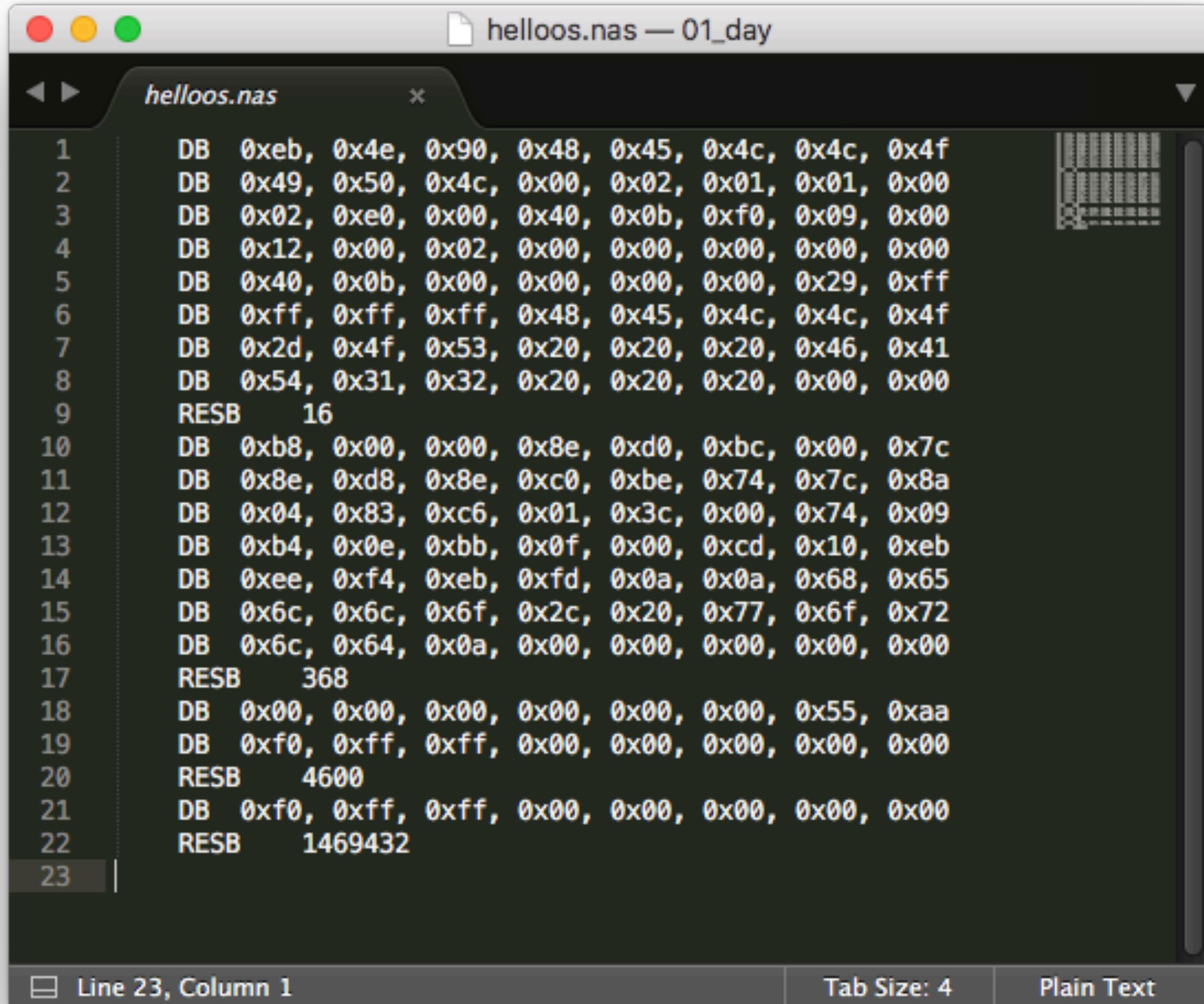
## ASCII 코드표

2017.06.19  
최동훈

DEC	HEX	OCT	Char	DEC	HEX	OCT	Char	DEC	HEX	OCT	Char
0	00	000	Ctrl-@ NUL	43	2B	053	+	86	56	126	V
1	01	001	Ctrl-A SOH	44	2C	054	.	87	57	127	W
2	02	002	Ctrl-B STX	45	2D	055	-	88	58	130	X
3	03	003	Ctrl-C ETX	46	2E	056	.	89	59	131	Y
4	04	004	Ctrl-D EOT	47	2F	057	/	90	5A	132	Z
5	05	005	Ctrl-E ENQ	48	30	060	0	91	5B	133	[
6	06	006	Ctrl-F ACK	49	31	061	1	92	5C	134	₩
7	07	007	Ctrl-G BEL	50	32	062	2	93	5D	135	]
8	08	010	Ctrl-H BS	51	33	063	3	94	5E	136	^
9	09	011	Ctrl-I HT	52	34	064	4	95	5F	137	_
10	0A	012	Ctrl-J LF	53	35	065	5	96	60	140	`
11	0B	013	Ctrl-K VT	54	36	066	6	97	61	141	a
12	0C	014	Ctrl-L FF	55	37	067	7	98	62	142	b
13	0D	015	Ctrl-M CR	56	38	070	8	99	63	143	c
14	0E	016	Ctrl-N SO	57	39	071	9	100	64	144	d
15	0F	017	Ctrl-O SI	58	3A	072	:	101	65	145	e
16	10	020	Ctrl-P DLE	59	3B	073	:	102	66	146	f
17	11	021	Ctrl-Q DCI	60	3C	074	<	103	67	147	g
18	12	022	Ctrl-R DC2	61	3D	075	=	104	68	150	h
19	13	023	Ctrl-S DC3	62	3E	076	>	105	69	151	i
20	14	024	Ctrl-T DC4	63	3F	077	?	106	6A	152	j
21	15	025	Ctrl-U NAK	64	40	100	@	107	6B	153	k
22	16	026	Ctrl-V SYN	65	41	101	A	108	6C	154	l
23	17	027	Ctrl-W ETB	66	42	102	B	109	6D	155	m
24	18	030	Ctrl-X CAN	67	43	103	C	110	6E	156	n
25	19	031	Ctrl-Y EM	68	44	104	D	111	6F	157	o
26	1A	032	Ctrl-Z SUB	69	45	105	E	112	70	160	p
27	1B	033	Ctrl-[ ESC	70	46	106	F	113	71	161	q
28	1C	034	Ctrl-\ FS	71	47	107	G	114	72	162	r
29	1D	035	Ctrl-] GS	72	48	110	H	115	73	163	s
30	1E	036	Ctrl-^ RS	73	49	111	I	116	74	164	t
31	1F	037	Ctrl_ US	74	4A	112	J	117	75	165	u
32	20	040	Space	75	4B	113	K	118	76	166	v
33	21	041	!	76	4C	114	L	119	77	167	w
34	22	042	"	77	4D	115	M	120	78	170	x
35	23	043	#	78	4E	116	N	121	79	171	y
36	24	044	\$	79	4F	117	O	122	7A	172	z
37	25	045	%	80	50	120	P	123	7B	173	{
38	26	046	&	81	51	121	Q	124	7C	174	
39	27	047	'	82	52	122	R	125	7D	175	}
40	28	050	(	83	53	123	S	126	7E	176	)

# 어셈블러 도전

2017.06.19  
최동훈



The screenshot shows a terminal window titled "helloos.nas — 01\_day". The window contains assembly code for a program named "helloos.nas". The code includes memory allocations (DB) and a large block of RESB (reserved space) at the end. The assembly code is as follows:

```
1      DB  0xeb, 0x4e, 0x90, 0x48, 0x45, 0x4c, 0x4c, 0x4f
2      DB  0x49, 0x50, 0x4c, 0x00, 0x02, 0x01, 0x01, 0x00
3      DB  0x02, 0xe0, 0x00, 0x40, 0x0b, 0xf0, 0x09, 0x00
4      DB  0x12, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00
5      DB  0x40, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x29, 0xff
6      DB  0xff, 0xff, 0xff, 0x48, 0x45, 0x4c, 0x4c, 0x4f
7      DB  0x2d, 0x4f, 0x53, 0x20, 0x20, 0x20, 0x46, 0x41
8      DB  0x54, 0x31, 0x32, 0x20, 0x20, 0x20, 0x00, 0x00
9      RESB  16
10     DB  0xb8, 0x00, 0x00, 0x8e, 0xd0, 0xbc, 0x00, 0x7c
11     DB  0x8e, 0xd8, 0x8e, 0xc0, 0xbe, 0x74, 0x7c, 0x8a
12     DB  0x04, 0x83, 0xc6, 0x01, 0x3c, 0x00, 0x74, 0x09
13     DB  0xb4, 0x0e, 0xbb, 0x0f, 0x00, 0xcd, 0x10, 0xeb
14     DB  0xee, 0xf4, 0xeb, 0xfd, 0xa, 0xa, 0x68, 0x65
15     DB  0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x77, 0x6f, 0x72
16     DB  0x6c, 0x64, 0xa, 0x00, 0x00, 0x00, 0x00, 0x00
17     RESB  368
18     DB  0x00, 0x00, 0x00, 0x00, 0x00, 0x55, 0xaa
19     DB  0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
20     RESB  4600
21     DB  0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
22     RESB  1469432
23
```

At the bottom of the terminal window, it says "Line 23, Column 1" and "Tab Size: 4".

# 어셈블러

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL



최초의 Human readable 프로그래밍 언어



대소문자 가리지 않음



0x를 붙이면 16진수, 붙이지 않으면 10진수

db 명령어 - ‘data byte’ 약자, 파일에 내용을 1바이트 만큼 쓴다

resb 명령어 - ‘reserve byte’ 약자, 0을 뒤에 나오는 숫자 갯수만큼 채운다.

주석은 :

```
2. bash

-P<file>    pre-includes a file
-D<macro>[=<value>] pre-defines a macro
-U<macro>    undefines a macro
-X<format>   specifies error reporting format (gnu or vc)
-w+foo        enables warnings about foo; -w-foo disables them

where foo can be:
  macro-params          macro calls with wrong no. of params (default off)
  macro-selfref         cyclic macro self-references (default off)
  orphan-labels         labels alone on lines without trailing `:' (default
off)
  number-overflow       numeric constants greater than 0xFFFFFFFF (default o
n)
  gnu-elf-extensions   using 8- or 16-bit relocation in ELF, a GNU extensio
n (default off)

response files should contain command line parameters, one per line.

For a list of valid output formats, use -hf.
For a list of debug formats, use -f <form> -y.
sapsalogs-MacBook-Pro:helloos1 sapsaldog$ nasm helloos.nas -o myos.img
helloos.nas:9: warning: uninitialised space declared in .text section: zeroing
helloos.nas:17: warning: uninitialised space declared in .text section: zeroing
helloos.nas:20: warning: uninitialised space declared in .text section: zeroing
helloos.nas:22: warning: uninitialised space declared in .text section: zeroing
sapsalogs-MacBook-Pro:helloos1 sapsaldog$
```

2017.06.19  
최동훈

helloos.nas — 01\_day

```
1 ; hello-os
2 ; TAB=4
3
4 ; 이하는 표준적인 FAT12 포맷 플로피 디스크를 위한 기술
5
6     DB      0xeb, 0x4e, 0x90
7     DB      "HELLOIPL" ; boot sector의 이름은 자유롭게 써도
8     ; 좋다(8바이트)
9     DW      512      ; 1섹터의 크기(512로 해야 함)
10    DW      1         ; 클러스터의 크기(1섹터로 해야 함)
11    DW      1         ; FAT가 어디에서 시작될까(보통은 1섹터째부터)
12    DW      2         ; FAT의 개수(2로 해야 함)
13    DW      224       ; 루트 디렉토리 영역의 크기(보통은 224엔트리로 한다)
14    DW      2880      ; 드라이브 크기(2880섹터 해야 함)
15    DB      0xf0       ; 미디어 타입(0xf0해야 함)
16    DW      9          ; FAT영역 길이(9섹터로 해야 함)
17    DW      18         ; 1트랙에 몇개의 섹터가 있을까(18로 해야 함)
18    DW      2          ; 헤드 수(2로 해야 함)
19    DD      0          ; 파티션을 사용하지 않기 때문에 여기는 반드시 0
20    DD      2880      ; 드라이브 크기를 한번 더 write
21    DB      0,0,0x29   ; 잘 모르지만 이 값으로 해 두면 좋은 것 같다
22    DD      0xffffffff ; 아마, 볼륨 시리얼 번호
23    DB      "HELLO-OS" ; 디스크 이름(11바이트)
24    DB      "FAT12"   ; 포맷 이름(8바이트)
25    RESB    18         ; 우선 18바이트를 비어둔다
26
27 ; 프로그램 본체
28
29    DB      0xb8, 0x00, 0x00, 0x8e, 0xd0, 0xbc, 0x00, 0x7c
30    DB      0x8e, 0xd8, 0x8e, 0xc0, 0xbe, 0x74, 0x7c, 0x8a
31    DB      0x04, 0x83, 0xc6, 0x01, 0x3c, 0x00, 0x74, 0x09
32    DB      0xb4, 0x0e, 0xbb, 0x0f, 0x00, 0xcd, 0x10, 0xeb
33
34 ; 메세지 부분
35
36    DB      0xa, 0xa      ; 개행을 2개
37    DB      "hello, world"
38    DB      0xa          ; 개행
39    DB      0
40
41    RESB    0x1fe-$        ; 0x001fe까지를 0x00로 채우는 명령
42
43    DB      0x55, 0xaa
44
45 ; 이하는 boot sector이외의 부분을 기술
46
47    DB      0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
48    RESB    4600
49    DB      0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
50    RESB    1469432
51
```

Line 51, Column 1      Tab Size: 4      Plain Text

# Floopy disk

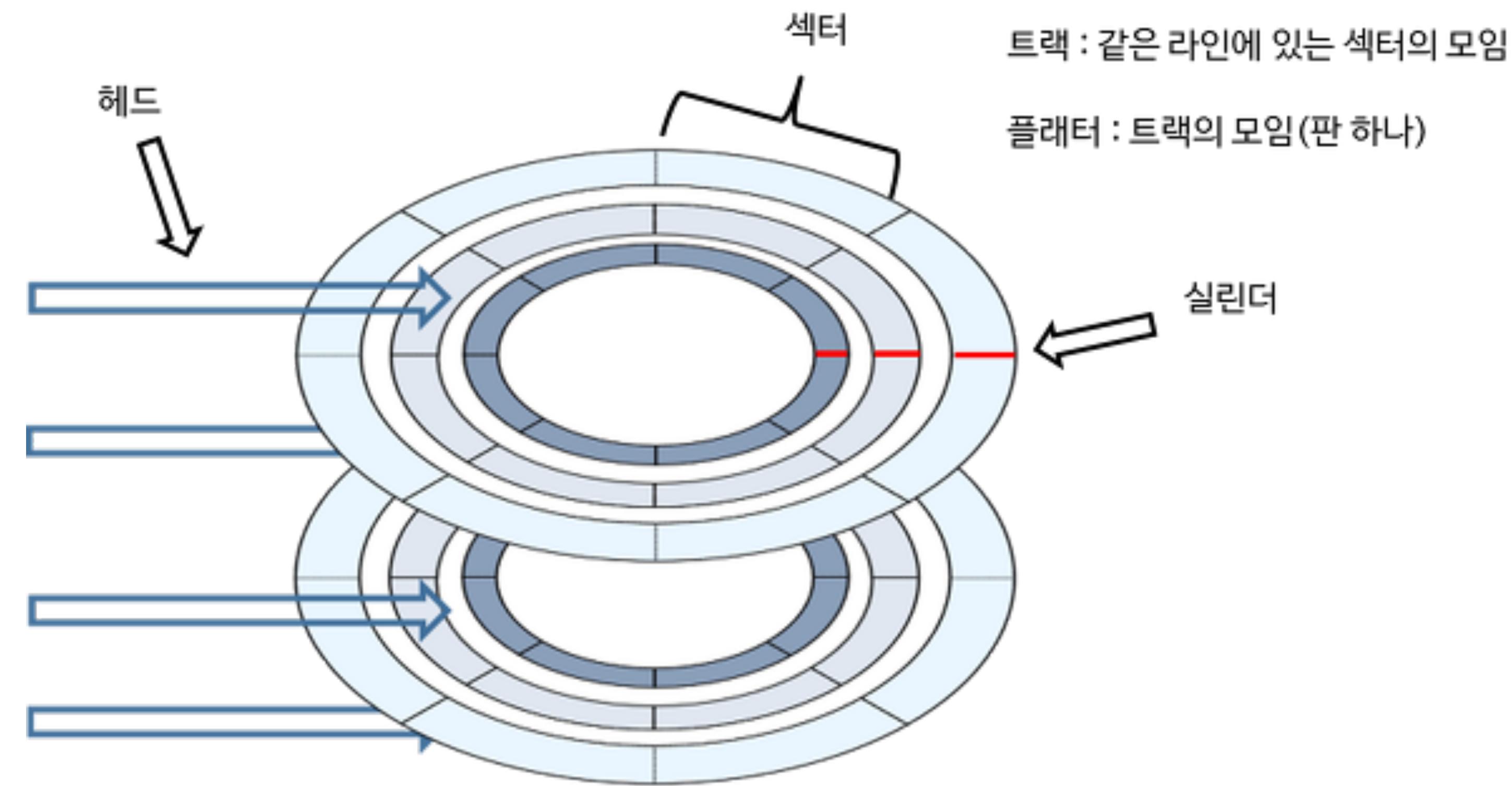
- 컴퓨터 보조 기억 장치
- 껍데기 안에 자성체로 덮여 있는, 회전할 수 있는 원판이 들어 있음

위키 참조 : [https://ko.wikipedia.org/wiki/%ED%94%8C%EB%A1%9C%ED%94%BC\\_%EB%94%94%EC%8A%A4%ED%81%AC](https://ko.wikipedia.org/wiki/%ED%94%8C%EB%A1%9C%ED%94%BC_%EB%94%94%EC%8A%A4%ED%81%AC)

당신의 커리어 전환점 패스트캠퍼스



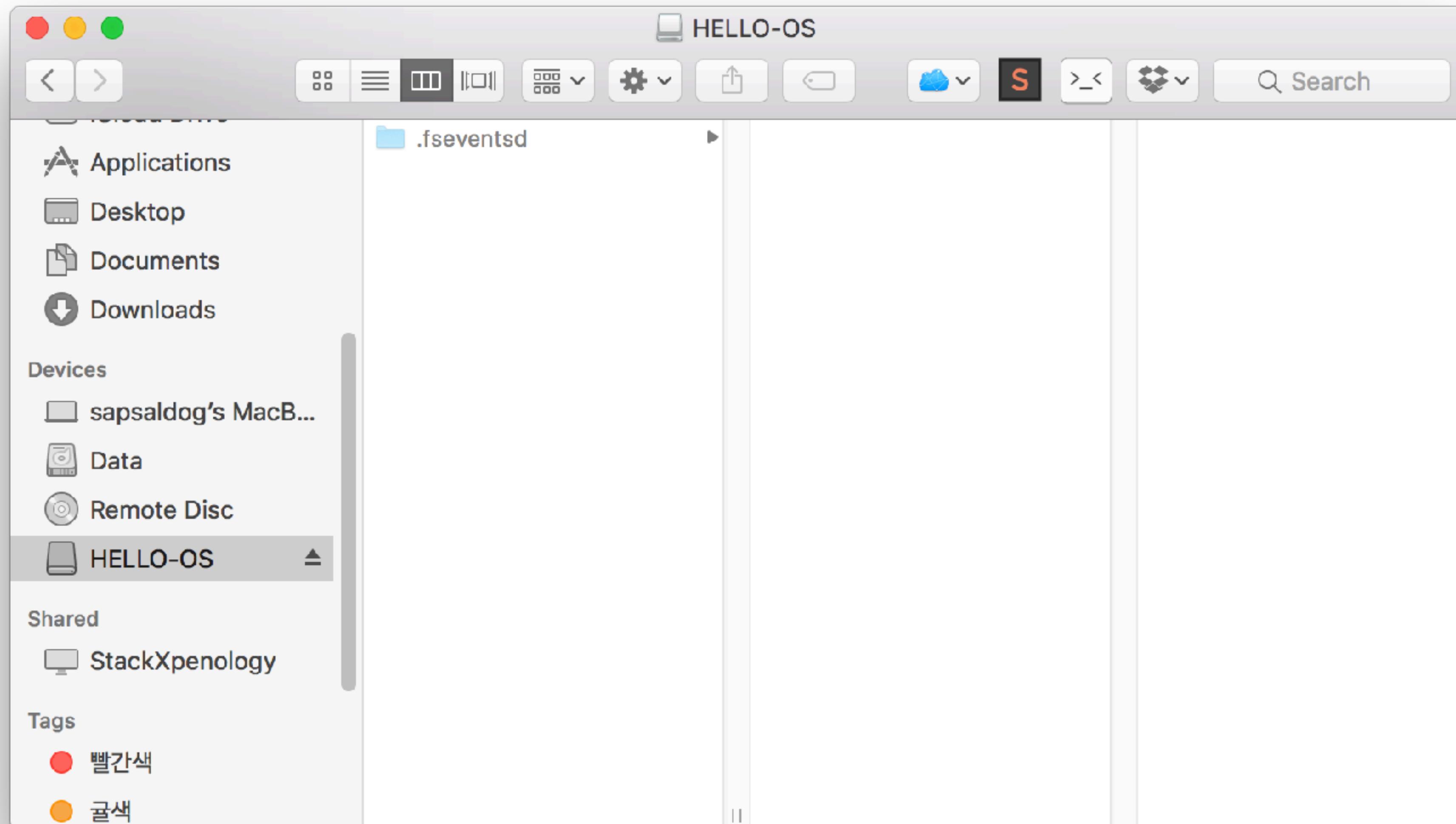
2017.06.19  
최동훈



# 플로피 디스크의 논리적 구조

섹터 / 트랙 / 플래터 / 클러스터

2017.06.19  
최동훈

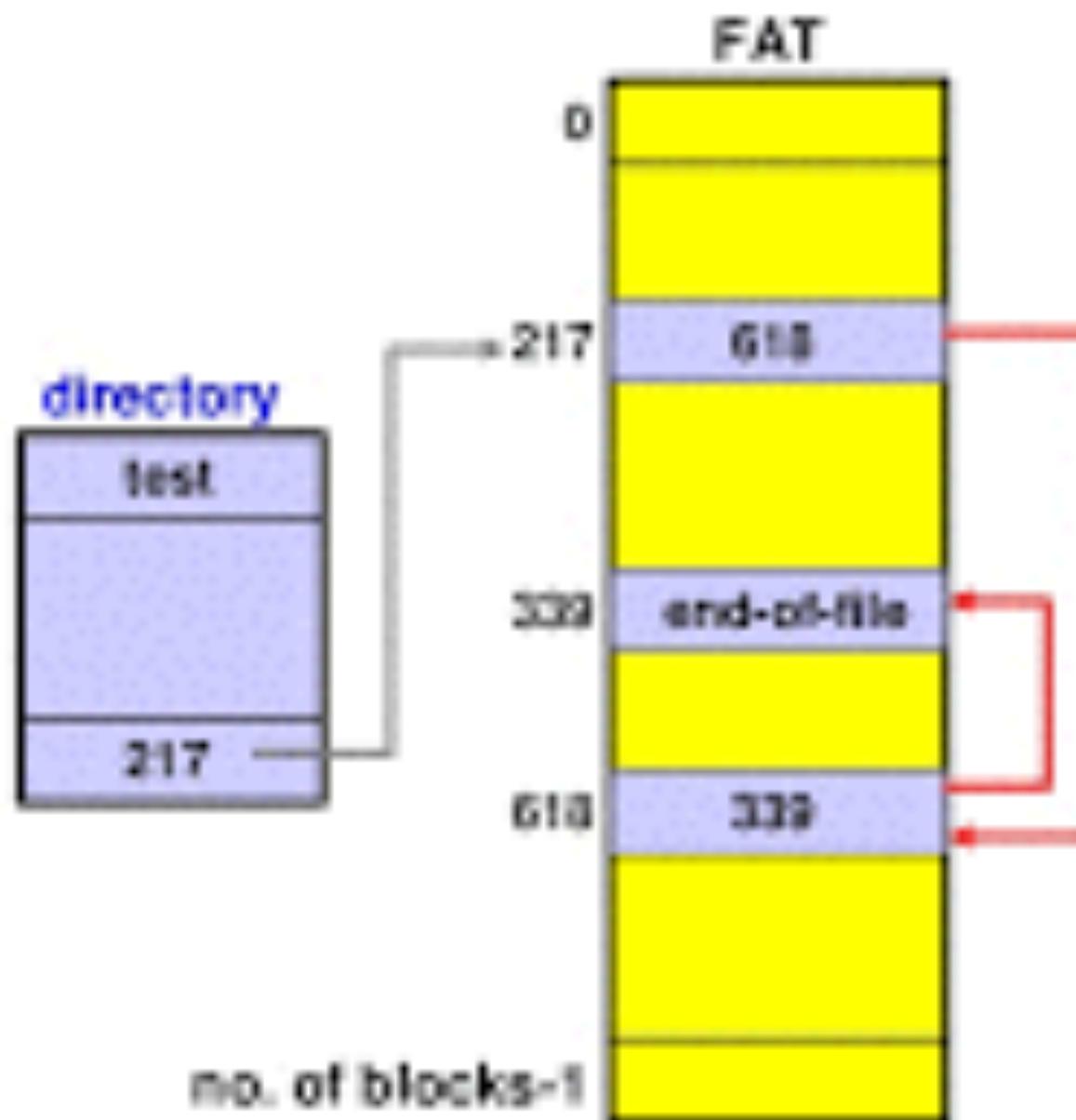


2017.06.19  
최동훈

# File Allocation Table

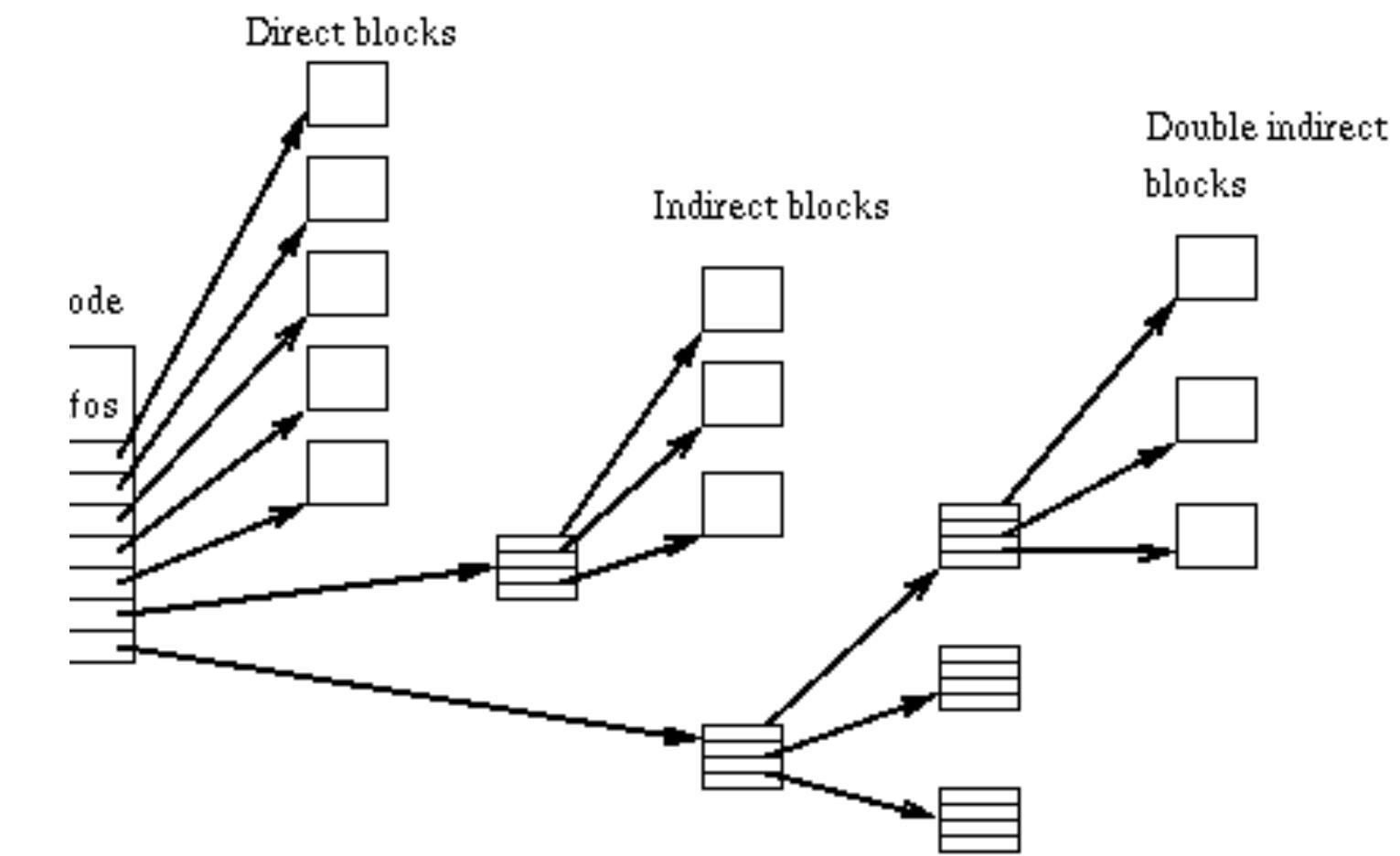
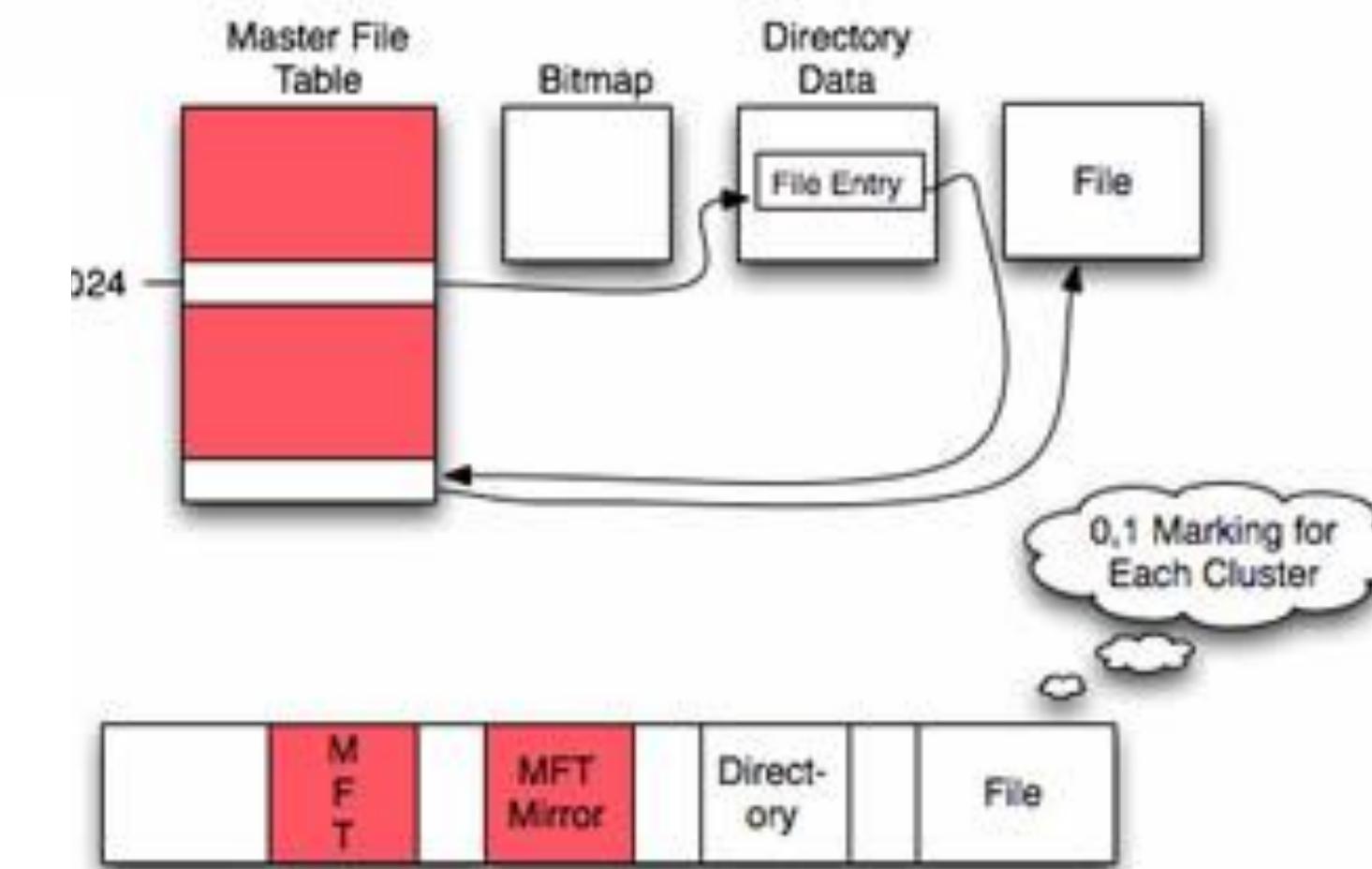
- 파일 할당 테이블(FAT)은 컴퓨터 시스템에 쓰이는 컴퓨터 파일 시스템 구조이다.
- FAT에는 여러가지 종류가 있으며 운영체제의 종류에 따라 지원하는 FAT의 종류에 차이가 있다.
- 운영체제는 FAT를 통해 파일관련 서비스를 제공 한다.

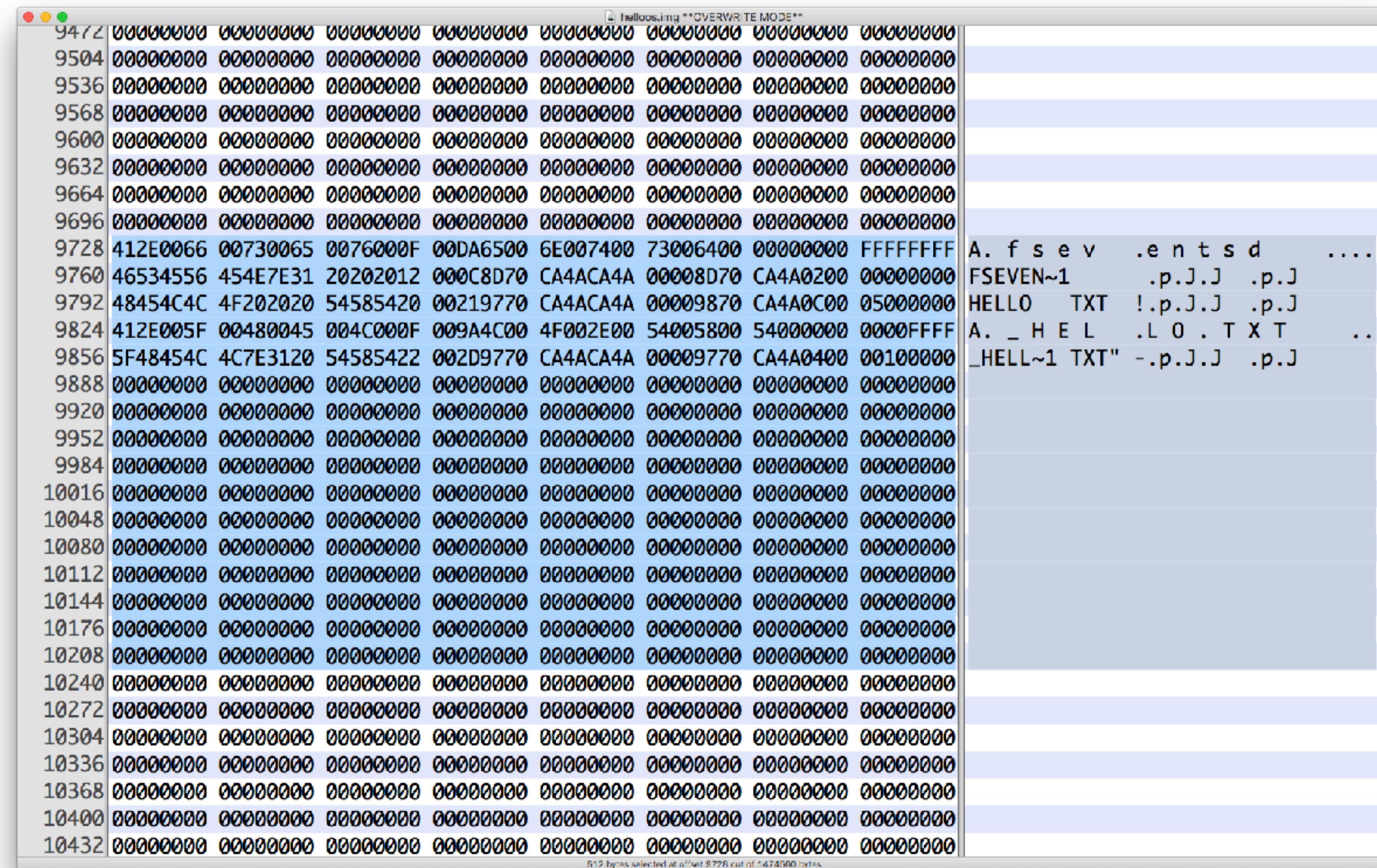
## File Allocation Table (FAT)



- This is a variation of the linked allocation by pulling all pointers into a table, the **file allocation table (FAT)**.
- Large no. of disk seeks.
- Can do direct access.
- FAT needs space.
- The left diagram shows file test has its first block at **217**, followed by **618, 339** (end of file).
- What if FAT is damaged? We all know it well!

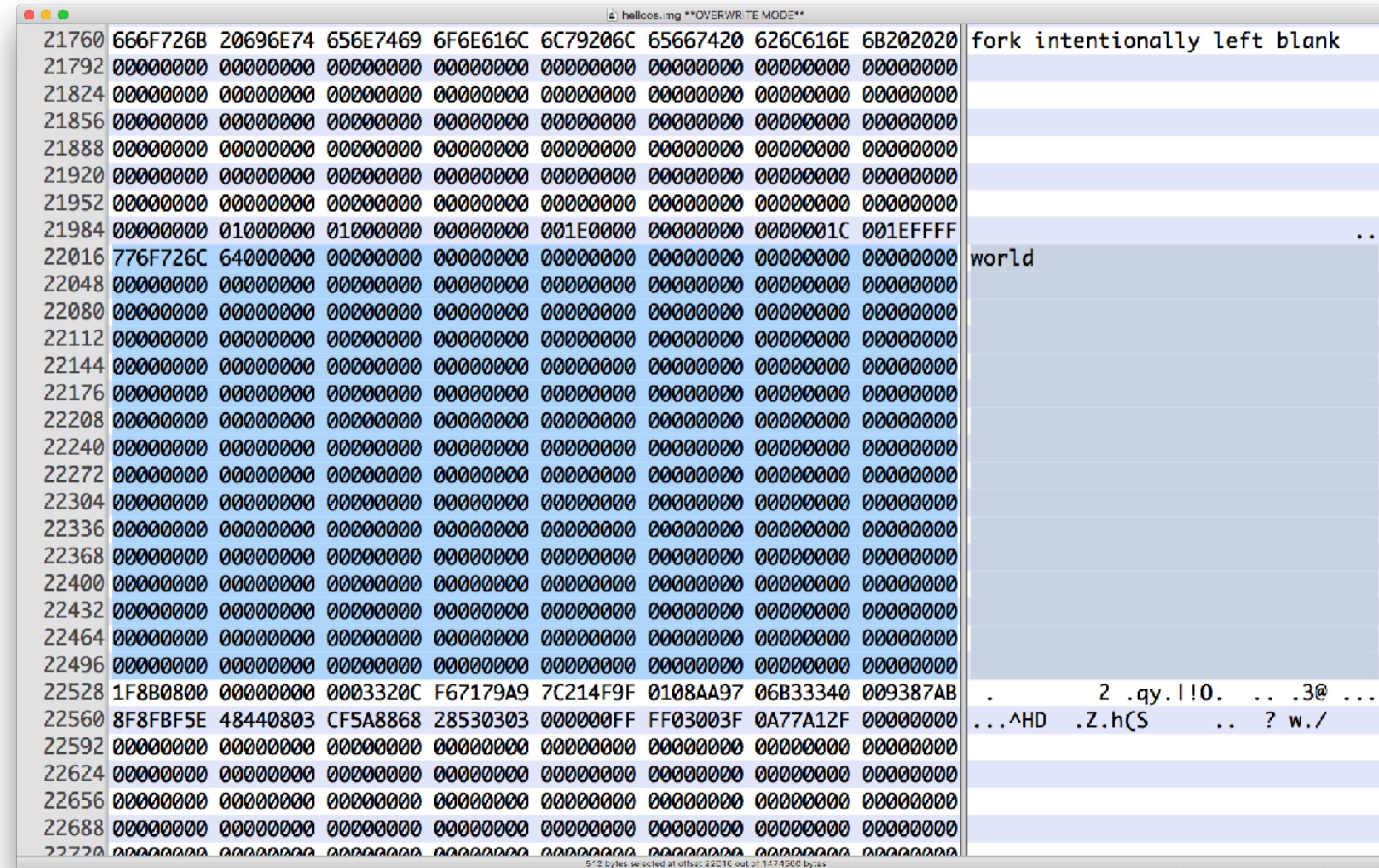
## NTFS FILE SYSTEM





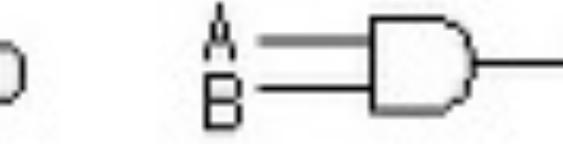
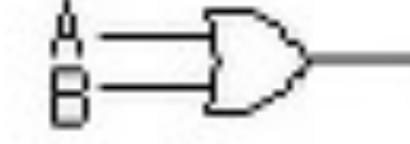
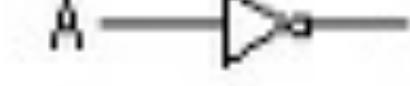
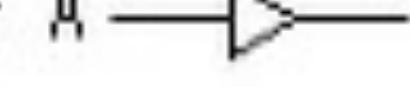
2017.06.19  
최동훈

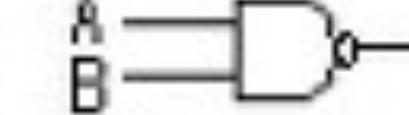
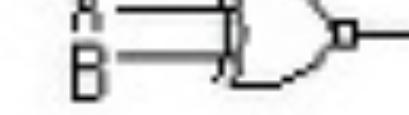
FAT 12 SPEC : <http://www.dfists.ua.es/~gil/FAT12Description.pdf>



2017.06.19  
최동훈

# 논리 게이트

명칭	그래픽 기호	함수식	진리치표															
AND		$X = AB$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	K	0	0	0	0	1	0	1	0	0	1	1	1
A	B	K																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$X = A+B$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	K	0	0	0	0	1	1	1	0	1	1	1	1
A	B	K																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$X = A^+$	<table border="1"> <thead> <tr> <th>A</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	K	0	1	1	0									
A	K																	
0	1																	
1	0																	
Buffer		$X = A$	<table border="1"> <thead> <tr> <th>A</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	K	0	1	1	0									
A	K																	
0	1																	
1	0																	

명칭	그래픽 기호	함수식	진리치표															
NAND		$X=(AB)^+$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	K	0	0	1	0	1	1	1	0	1	1	1	0
A	B	K																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$X=(A+B)^+$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	K	0	0	1	0	1	0	1	0	0	1	1	0
A	B	K																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$X=(A \oplus B)$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	K	0	0	0	0	1	1	1	0	1	1	1	0
A	B	K																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$X=(A \odot B)$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	K	0	0	1	0	1	0	1	0	0	1	1	1
A	B	K																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

# 논리 게이트 연습문제

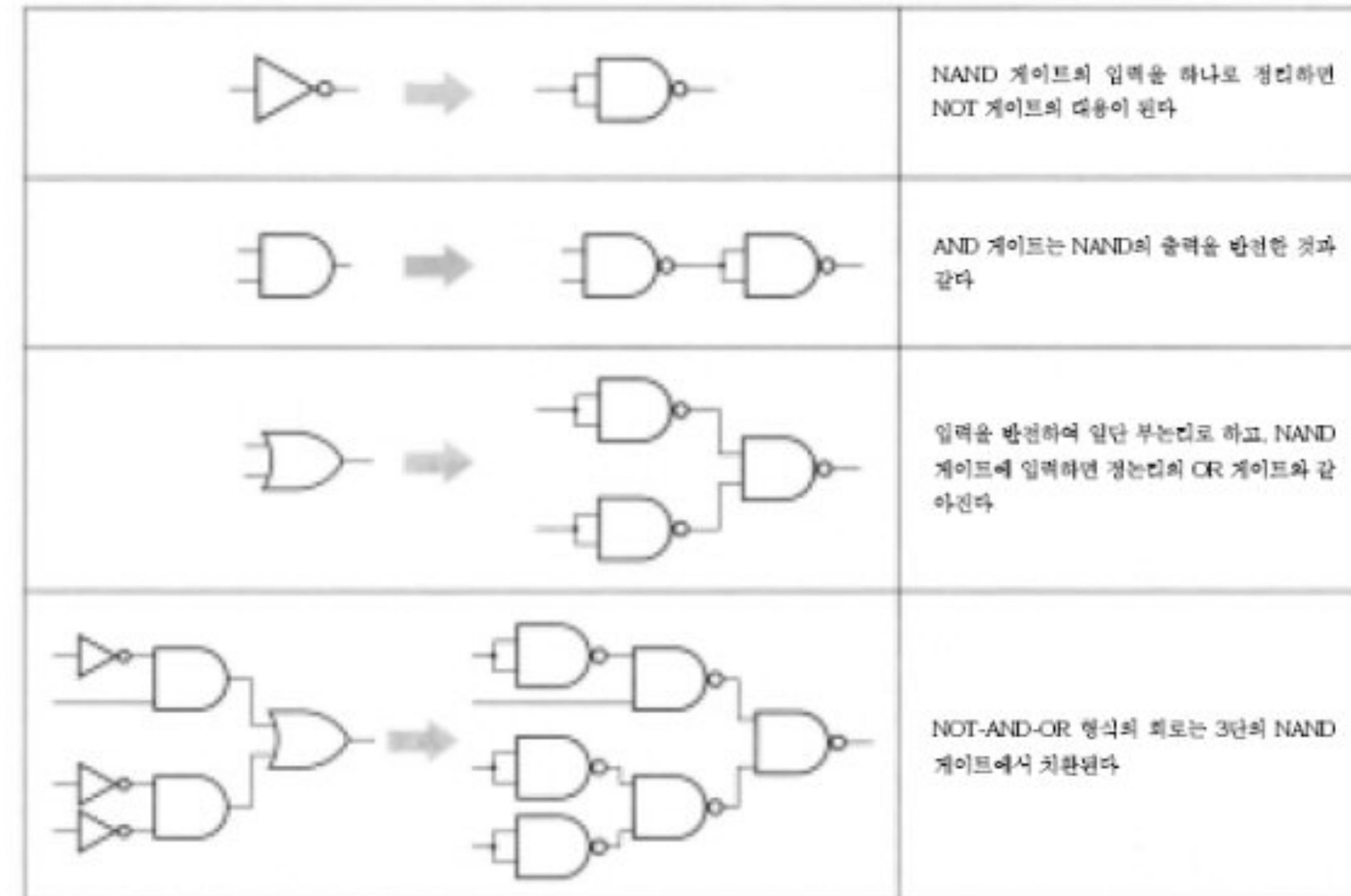
- 오른쪽 진리표를 AND/OR/NOT으로 구성해보자.
- NAND 게이트로 AND/OR/NOT을 만들 어보자

진리표 #1		
A	B	X
0	1	0
0	0	0
1	1	1
1	0	0

진리표 #2		
A	B	X
0	1	1
0	0	0
1	1	1
1	0	0

# NAND로 AND/OR/NOT 만들기

표 4. NOT, AND, OR를 NAND만인 회로에서 치환할 수 있다



# CPU

당신의 커리어 전환점 패스트캠퍼스

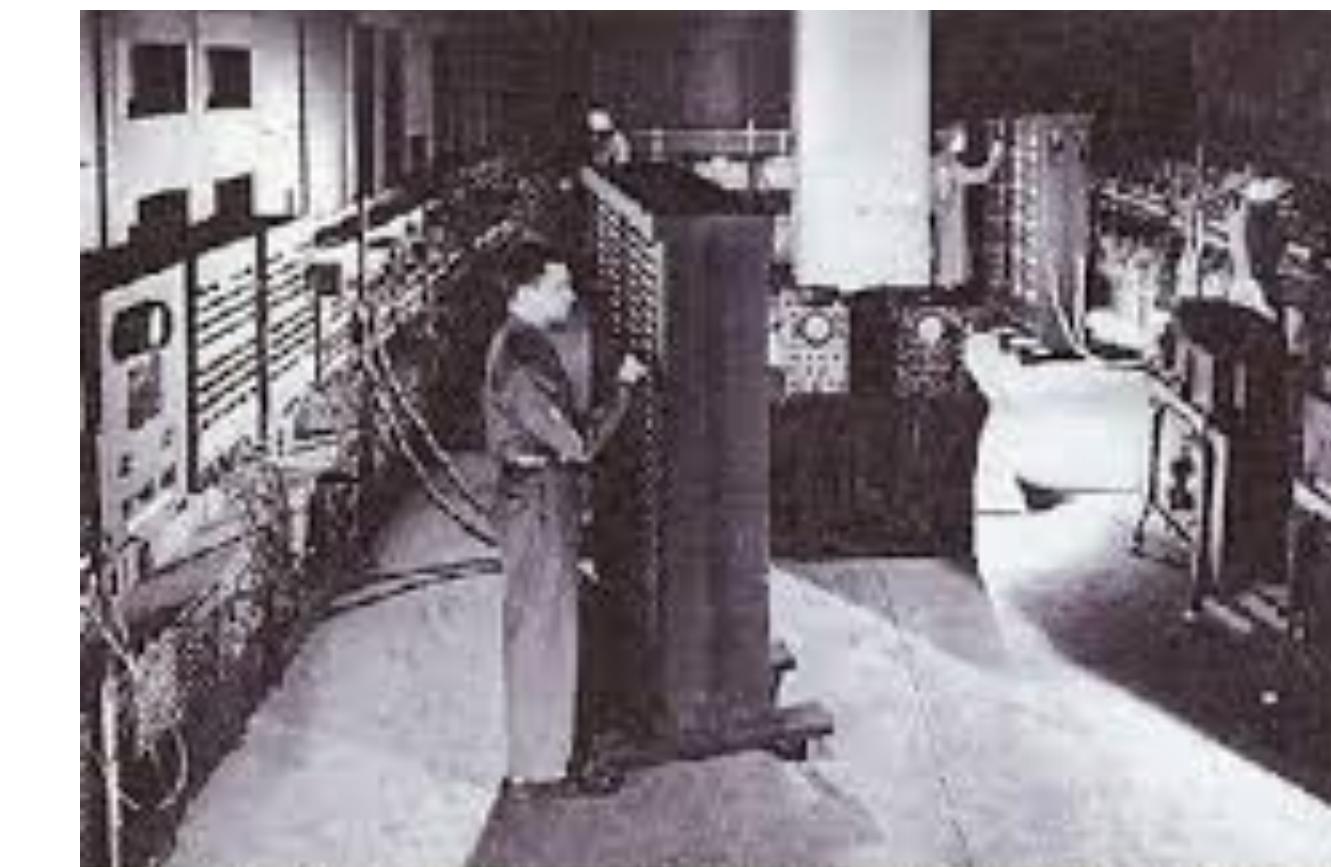
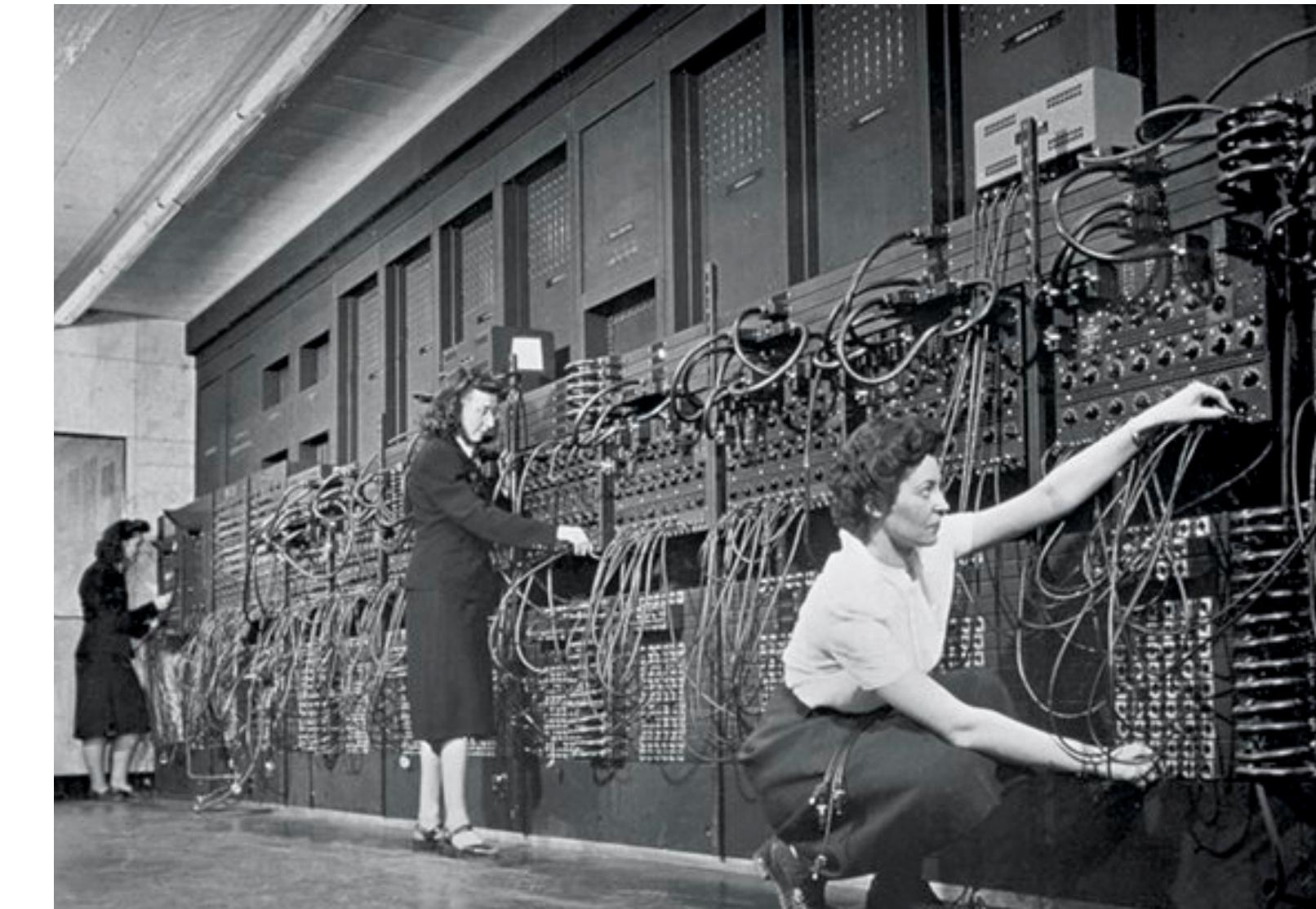
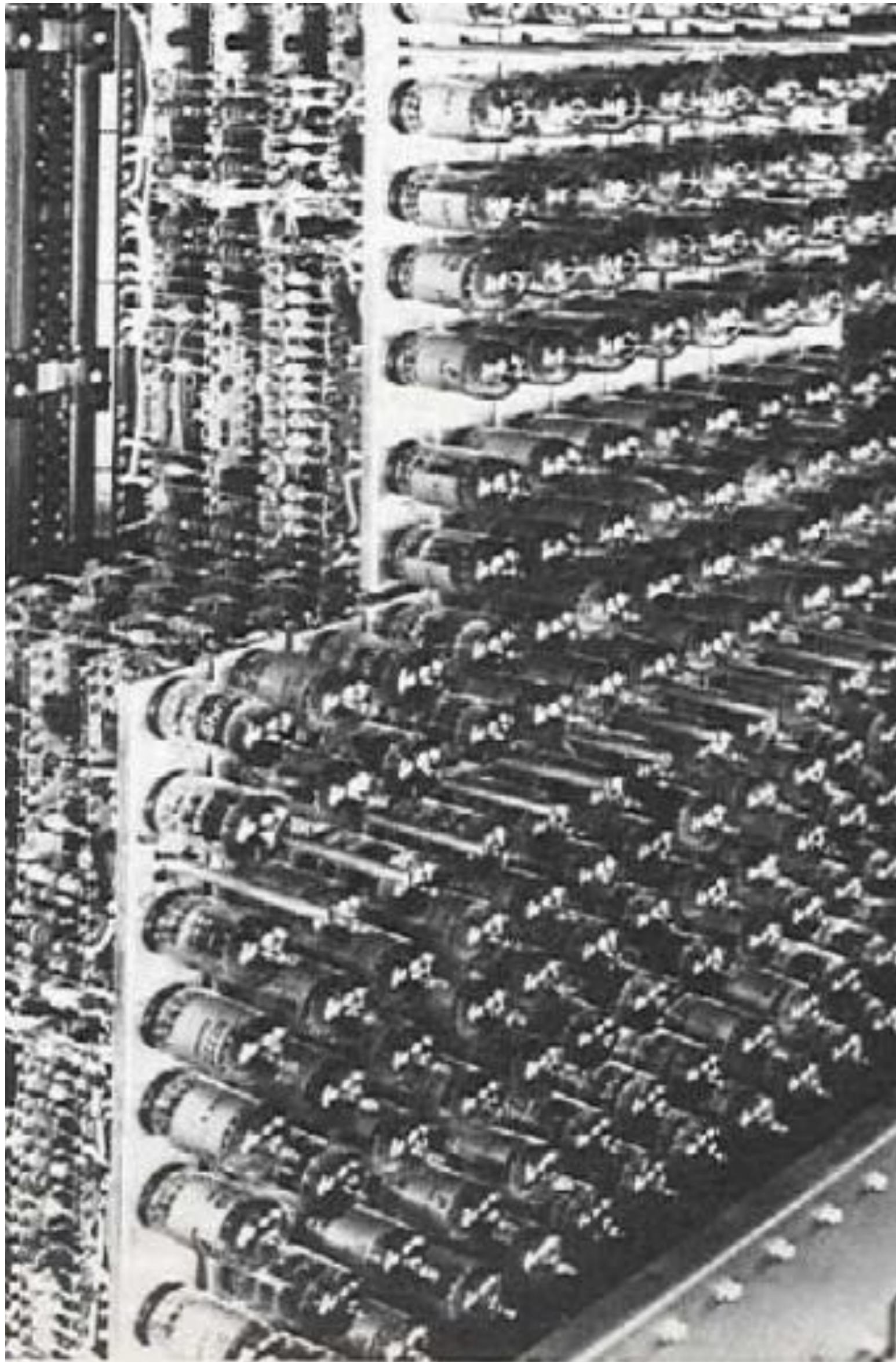
FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈

# 진공관 - 트랜지스터 - IC

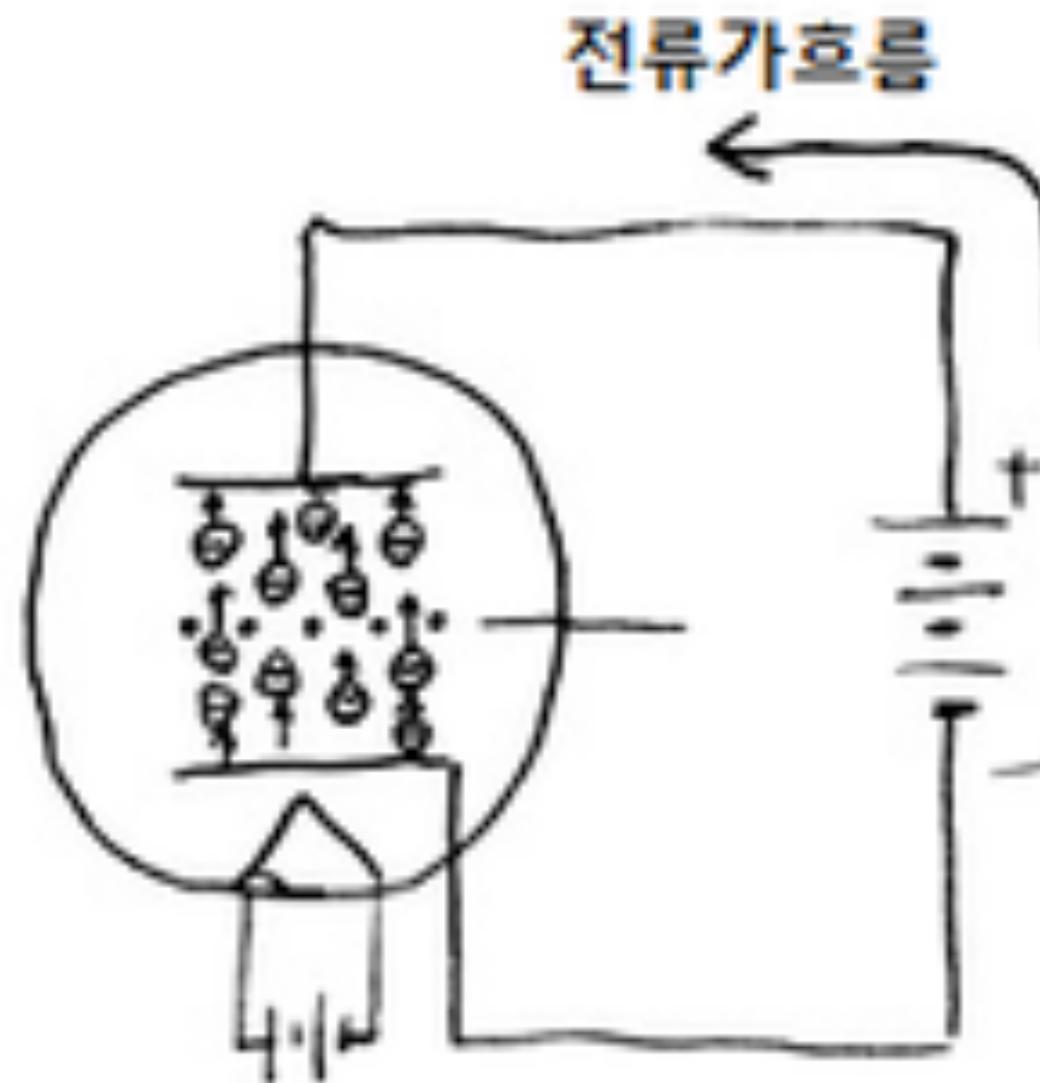
2017.06.19  
**최동훈**

2017.06.19  
최동훈

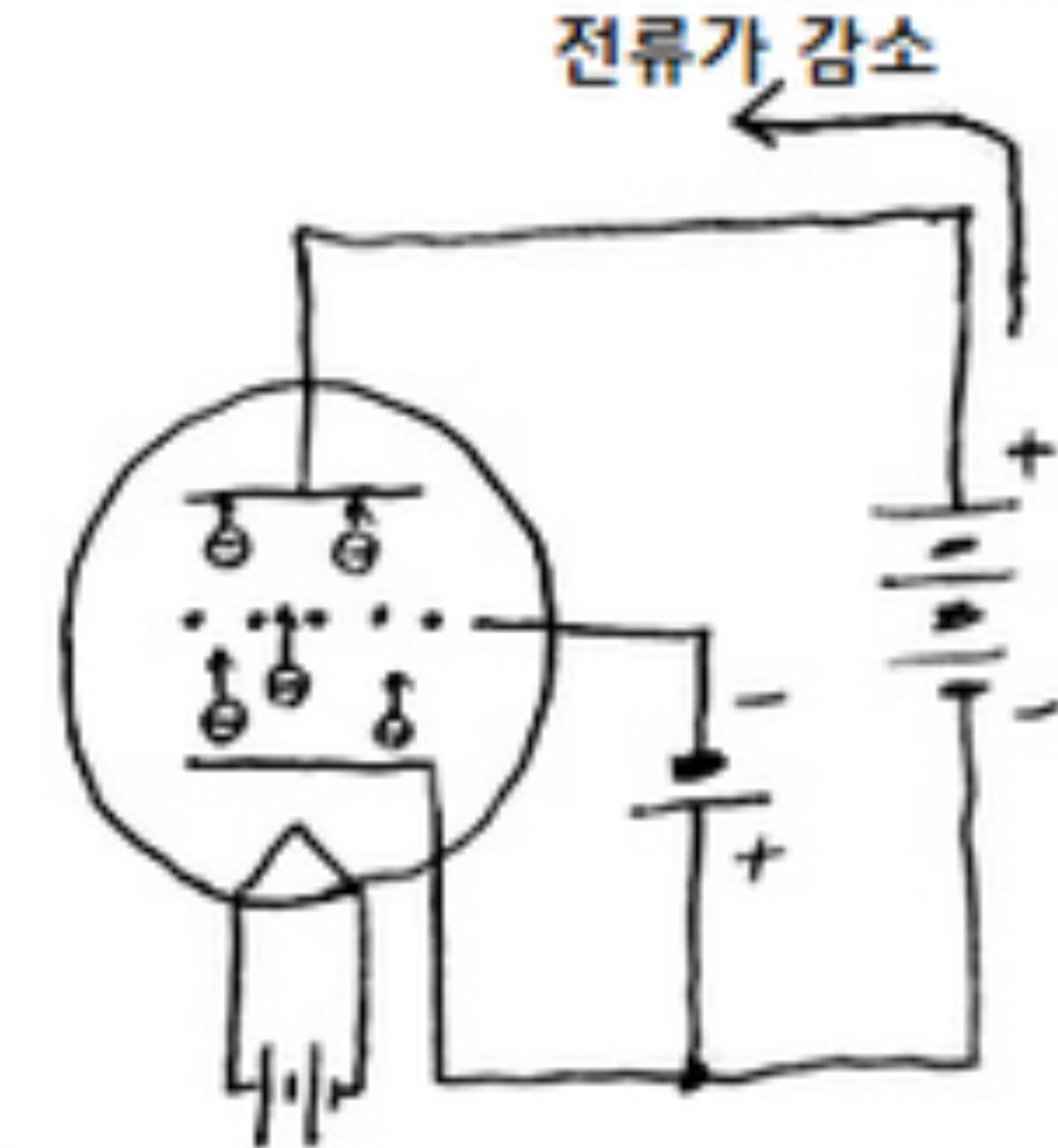


▲ 애니악은 군사용으로 개발되었으나 개발 도중인 1945년 전쟁이 끝나버리는 바람에 최종 완성된 1946년에는 본래 목적인 군사용으로 사용되지 못했다.

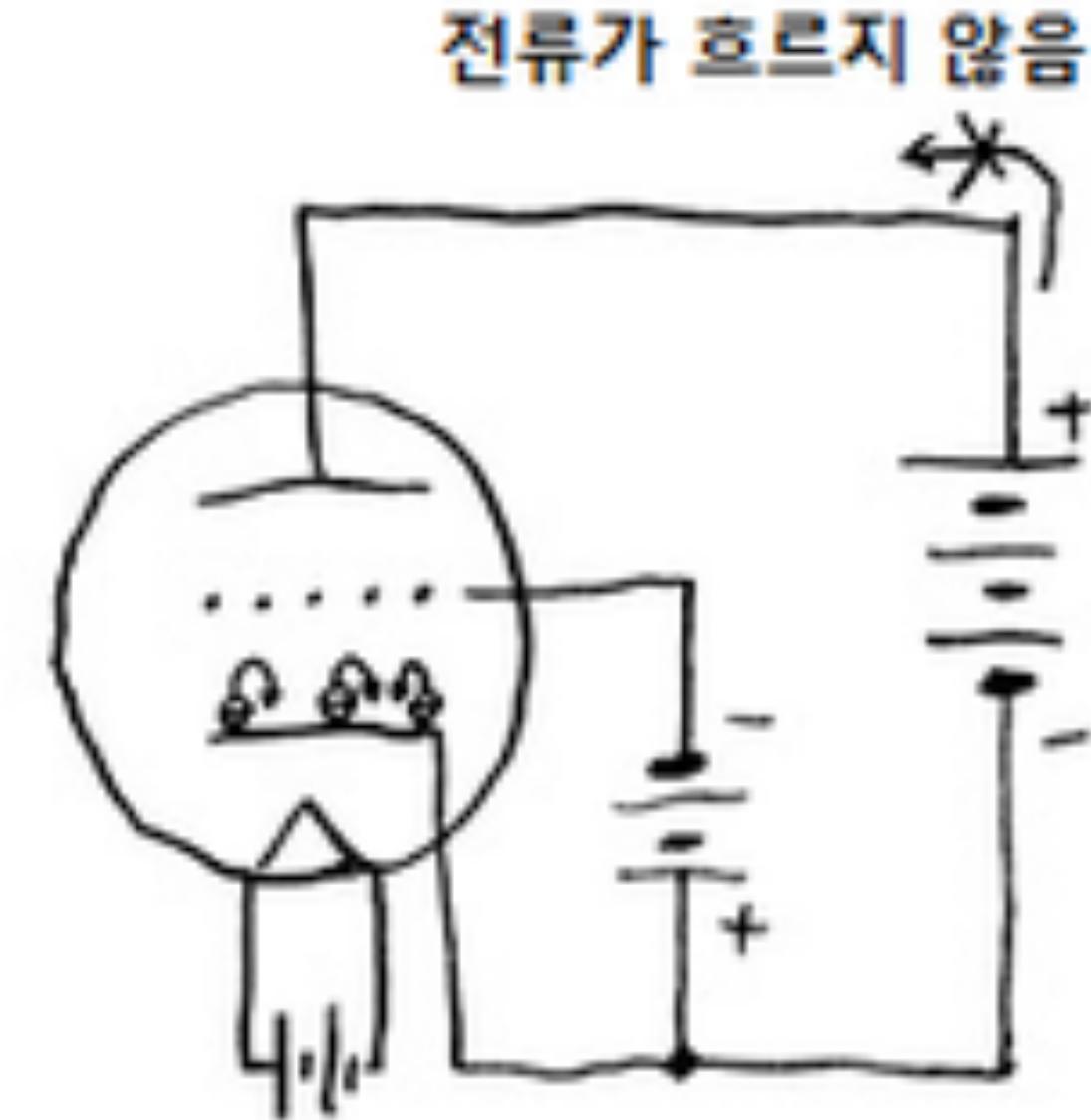
# 진공관



(a) 그리드에 아무것도  
가하지 않을 때



(b) 그리드에 작은(-) 전압을 걸 때



(c) 그리드에 큰(-) 전압을 걸 때

# 트렌지스터

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈

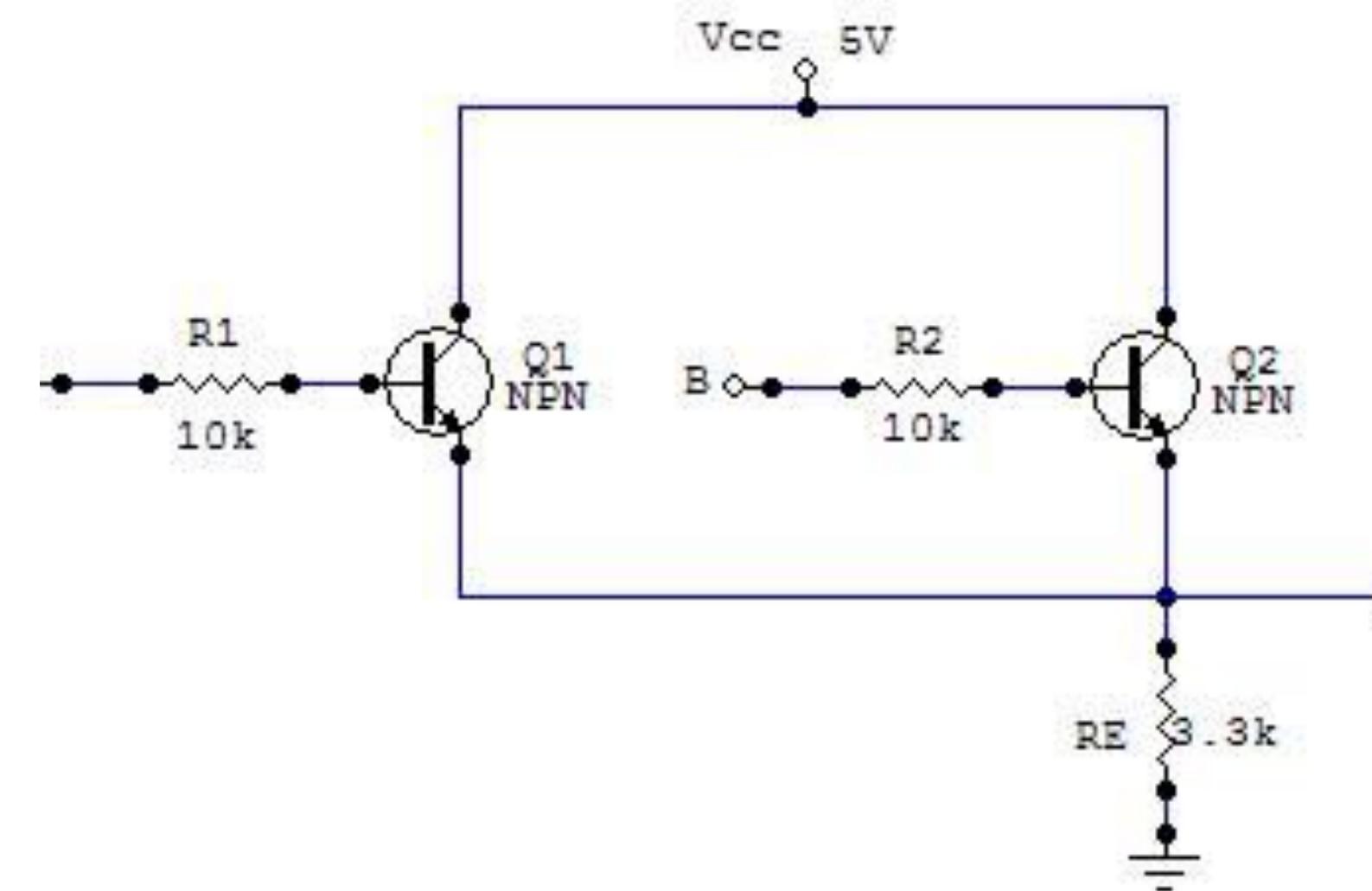
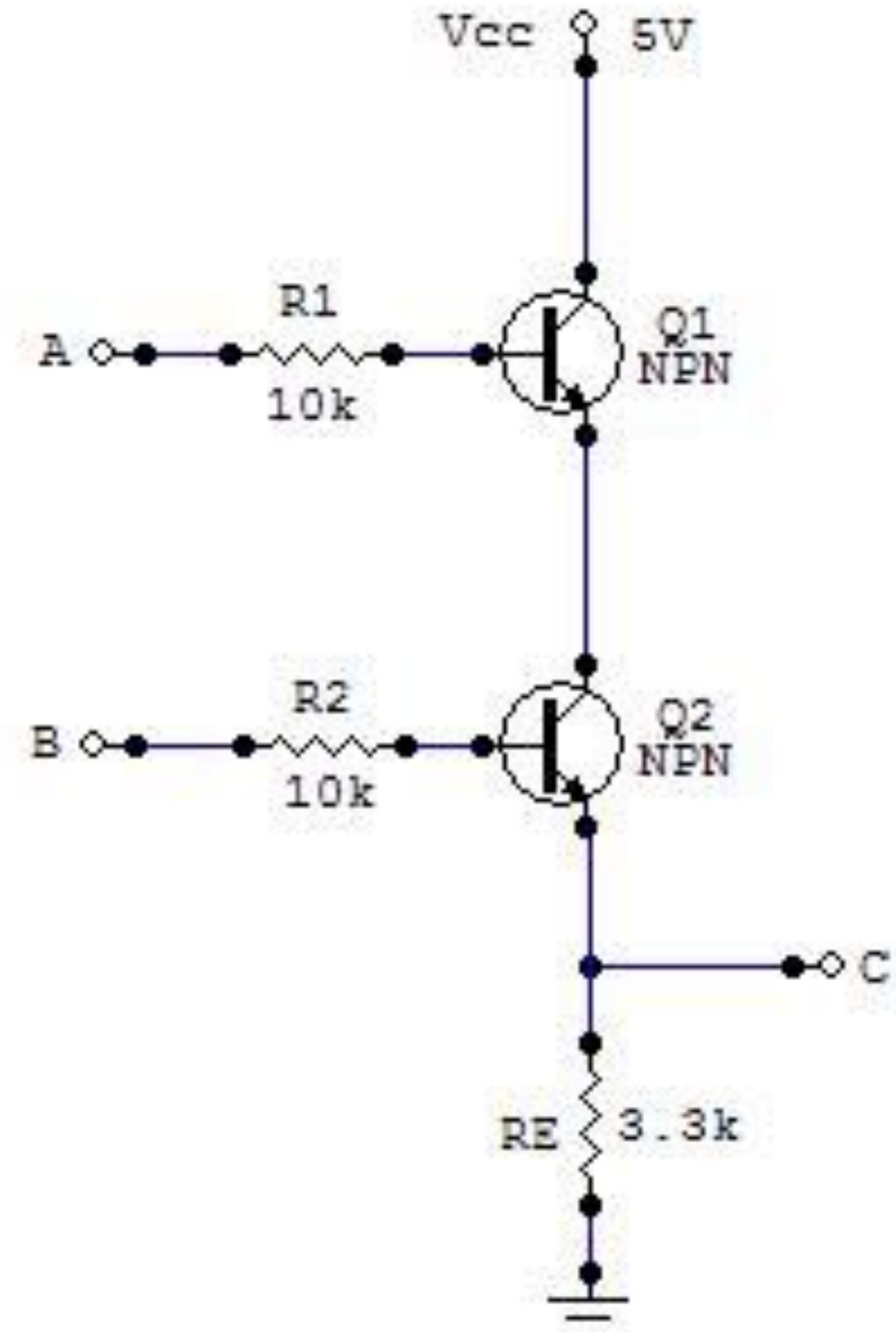
# 트렌지스터

당신의 커리어 전환점 패스트캠퍼스



FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈



2017.06.19  
최동훈

# 반가산기 설계

See How Computers  
Add Numbers  
In One Lesson

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈

# 반가산기 설계

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈

# 연습문제 - 반가산기 설계

2017.06.19  
최동훈

# 폰 노이만 구조

- 폰 노이만이 고안한 “내장 메모리 순차” 처리 방식
- 데이터와 메모리가 구분되어 있지 않고 하나의 버스를 가지고 있음

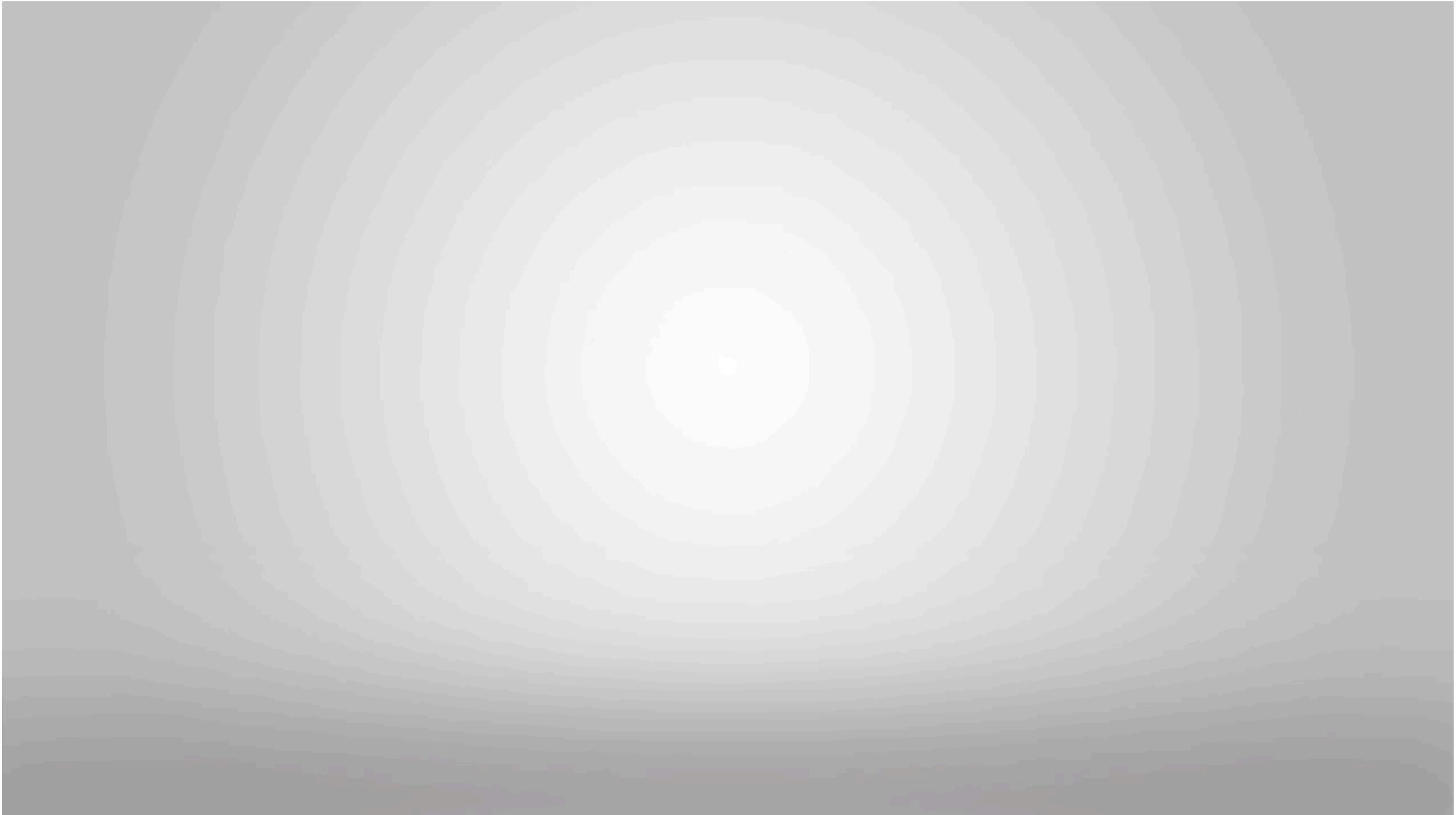
# 하버드 구조

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

- 하버드 마크I(Harvard Mark I)에서 나온 구조
- 메모리와 데이터의 버스가 구분되어 있다.

당신의 커리어 전환점 패스트캠퍼스



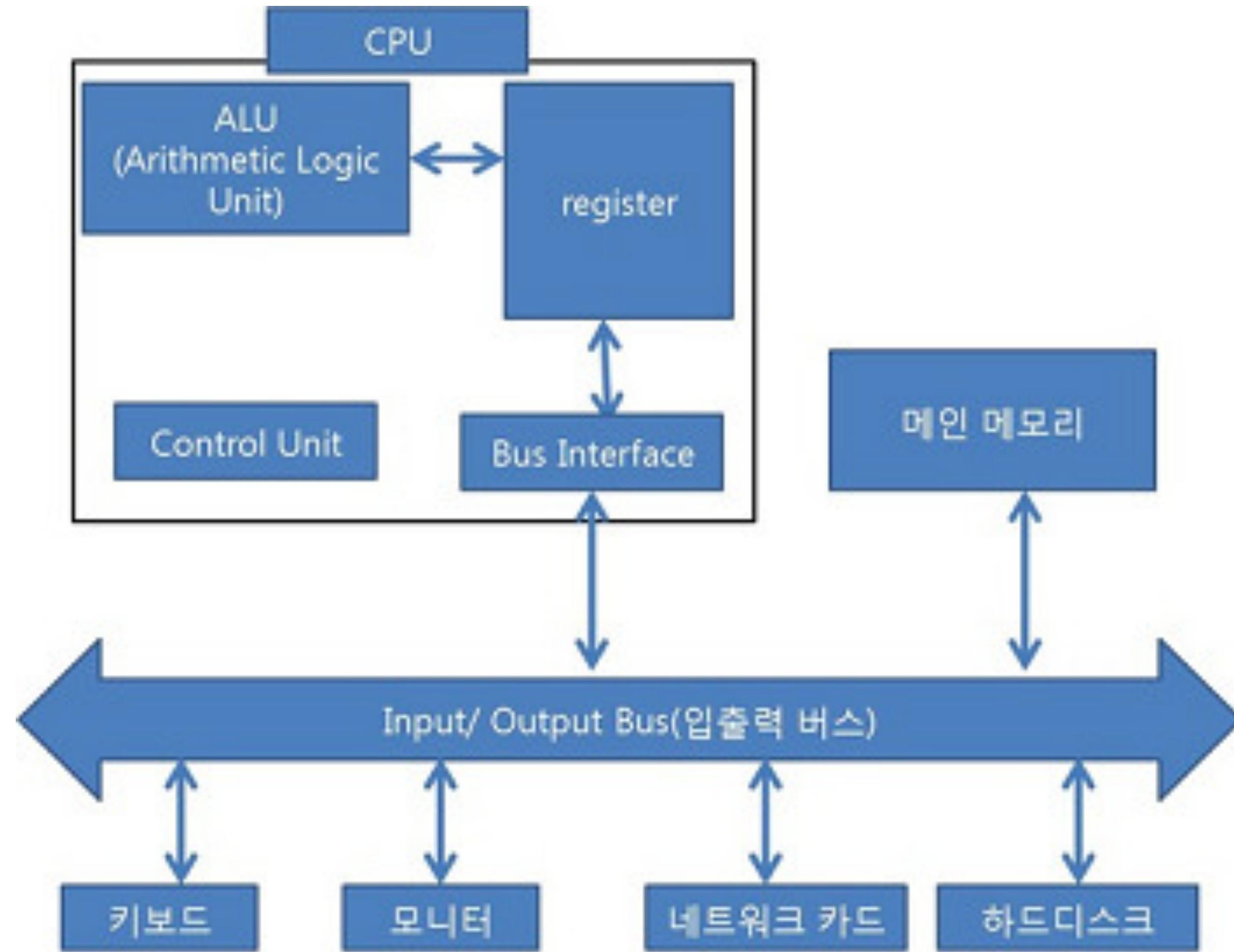
FRONTEND PROGRAMMING SCHOOL

2017.06.19  
**최동훈**

# CPU 구조

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL



2017.06.19  
최동훈

# 비트 연산자 (C와 JAVA)

- | (or 연산자) :  $1101|0110 = 1111$
- & (and 연산자) :  $0101&1100 = 0100$
- ^ (xor 연산자) :  $1010^0001 = 1000$
- ~ (not 연산자) :  $\sim 0111 = 1000$

# 비트 연산자 연습문제

- $1101^1010$
- $\sim(1110&1000)$
- $(\sim1010)|1101^0011$
- $111111|101010\&010101\&000000$
- $\sim((10100011|01100111)\&(10111011^00011111))$

# 16진수-10진수 변환 연습문제

- 다음 16진수를 10진수로 혹은 10진수를 16진수로 변환하세요
  - 0xff
  - 0xac0
  - 0xe5b2
  - 0xff2dac33
  - 255
  - 256
  - 355223
  - 10022383

# 16진수-10진수 변환 연습문제

- $1101^1010$
- $\sim(1110\&1000)$
- $(\sim1010)|1101^0011$
- $111111|101010\&010101\&000000$
- $\sim((10100011|01100111)\&(10111011^00011111))$

# 클럭과 CISC / RISC

- 매 클럭시마다 머신(CPU)의 상태는 변화됨
- CISC (Complex Instruction Set Computer)
- RISC (Reduced Instruction Set Computer)

# 클럭 이야기

- 486 시절 : SX(33Mhz) / DX(33Mhz) / DX2(66Mhz) / DX4(100Mhz)
- SX / DX 차이 : FPU(Floating Point Unit) 유무
- DX까지는 CPU와 주변 기기의 클럭차이가 없었음
- DX2 부터는 클럭 차이가 생김
- 클럭이 높아지면서 생기는 이야기 -> 이후 버스에서

# 동기 / 비동기

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL



2017.06.19  
**최동훈**

---

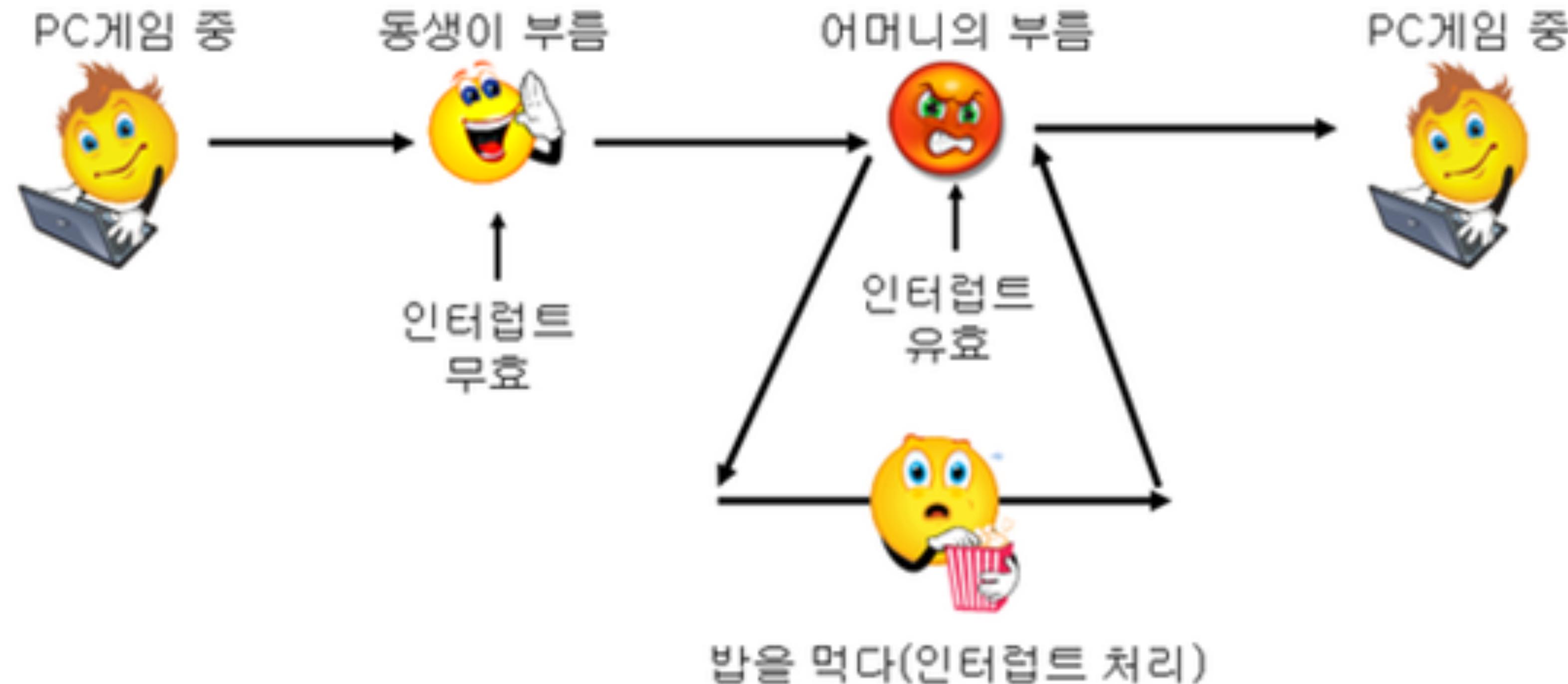
# 동기 / 비동기

- 동기(synchronization) 는 대상이 클럭에 의존함을 의미
- 비동기는 클럭에 의존하지 않음
- 블록 / 논블록(javascript콜)과 유사

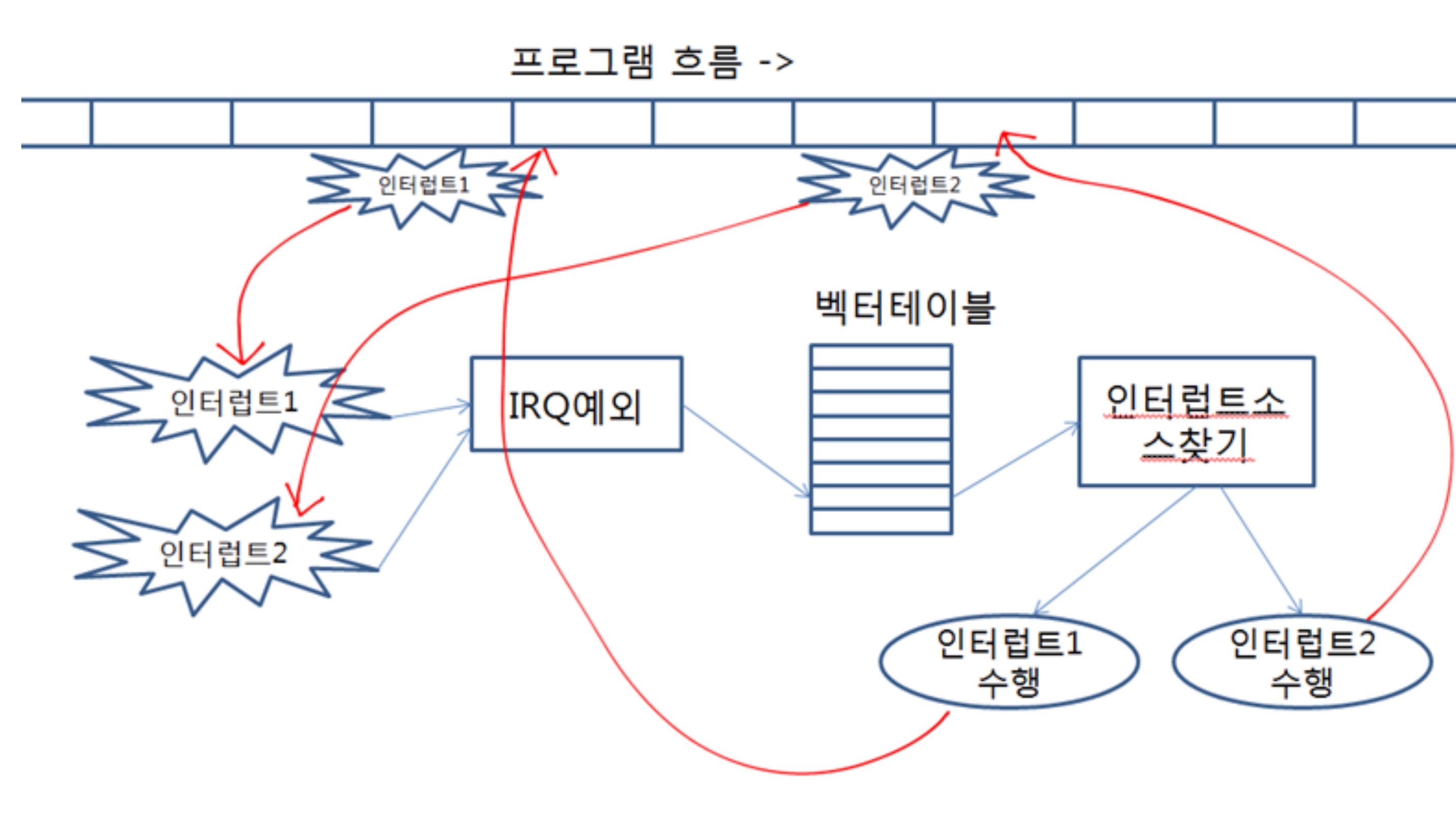
# 인터럽트

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL



# 인터럽트



하드웨어 인터럽트 / 소프트웨어 인터럽트

인터럽트 벡터 : 일종의 함수 테이블

인터럽트는 주어진 임무를 완수하고 이전에 하던 작업으로 복귀 해야 한다 -> 컨텍스트 스위칭

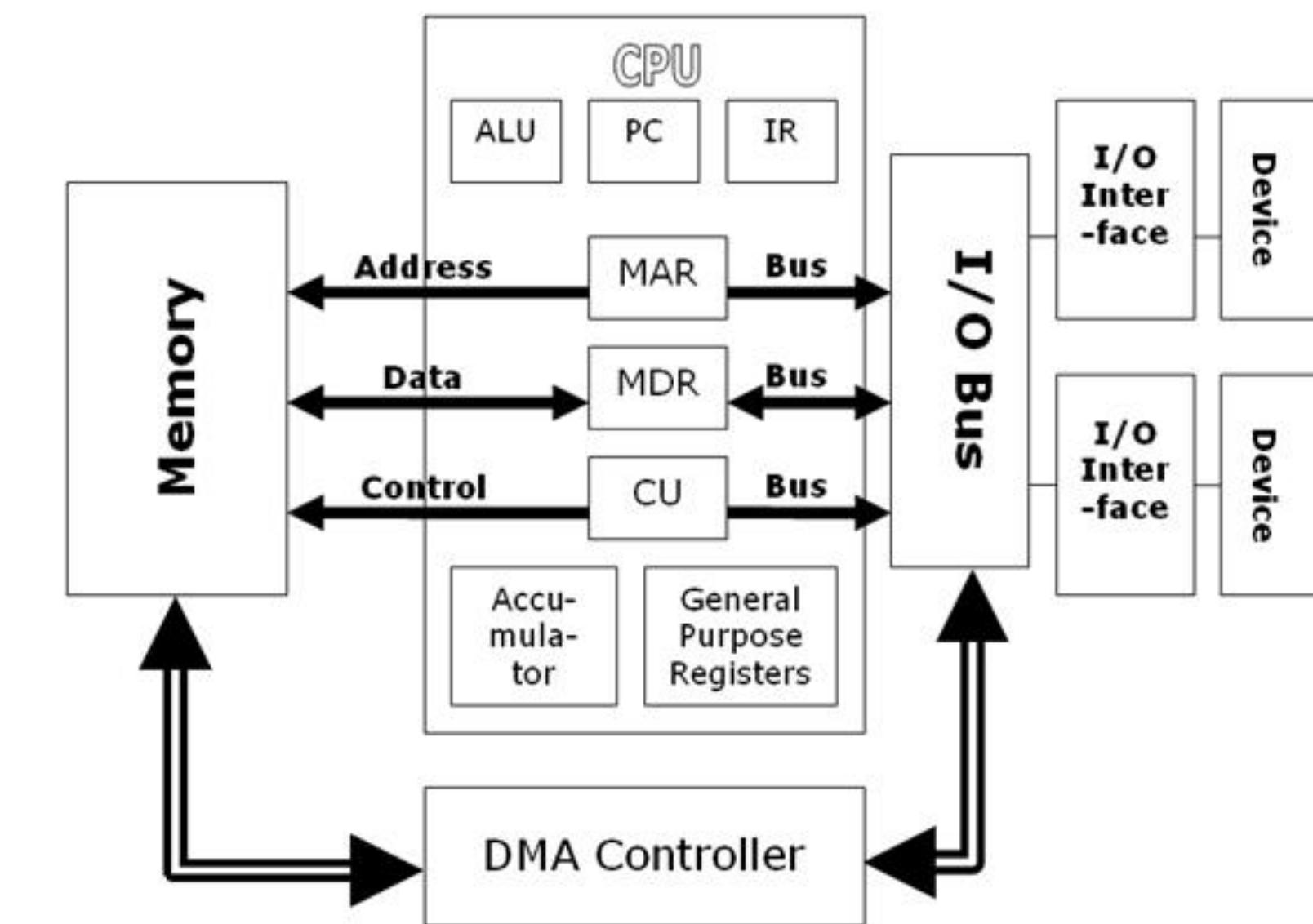
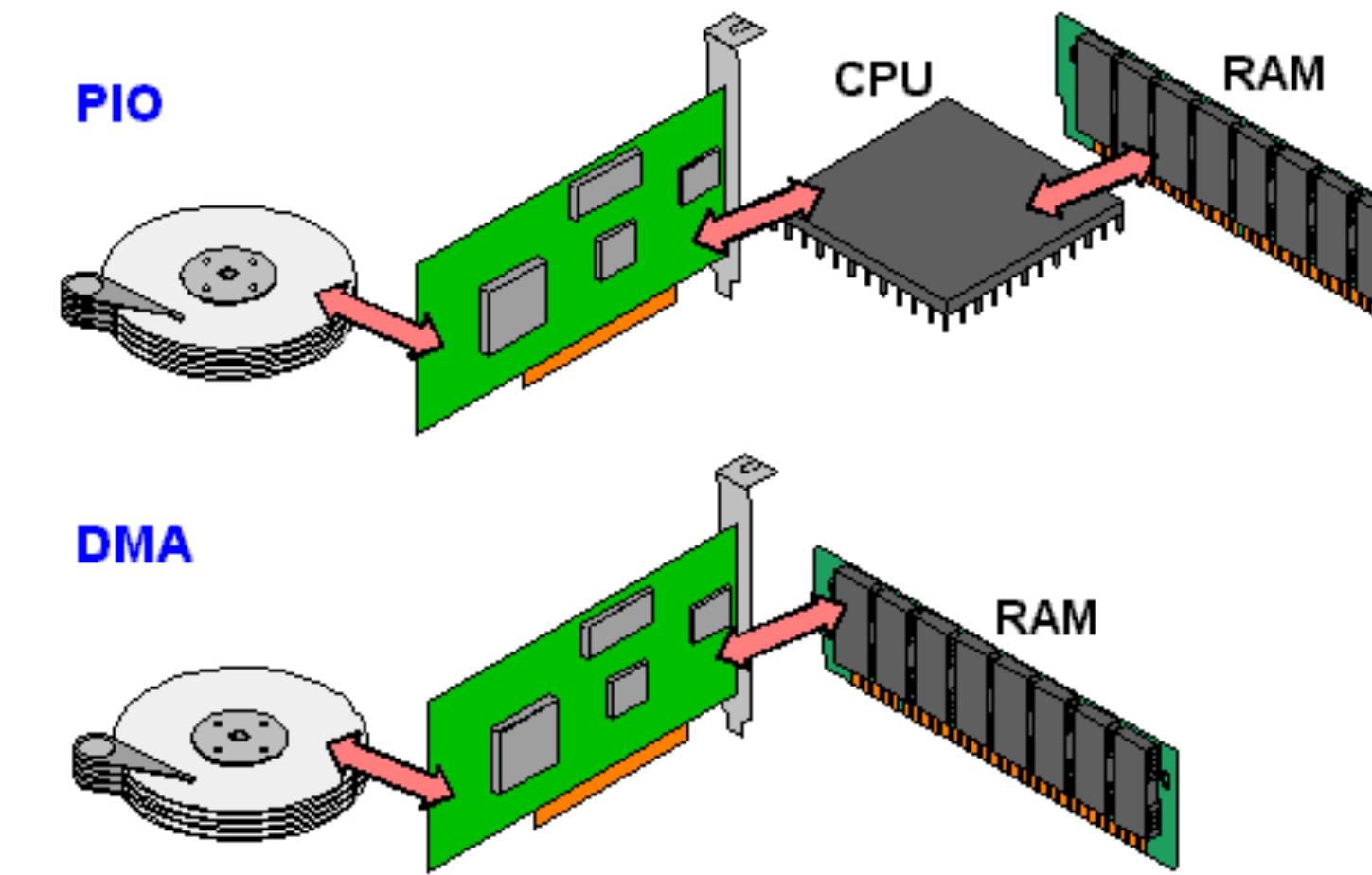
2017.06.19  
최동훈

# PIO/DMA

PIO(Programmed Input/Output)  
DMA(Direct Memory Access)

둘중에 어느것이 빠를까?

\* DMA와 인터럽트



# 메모리

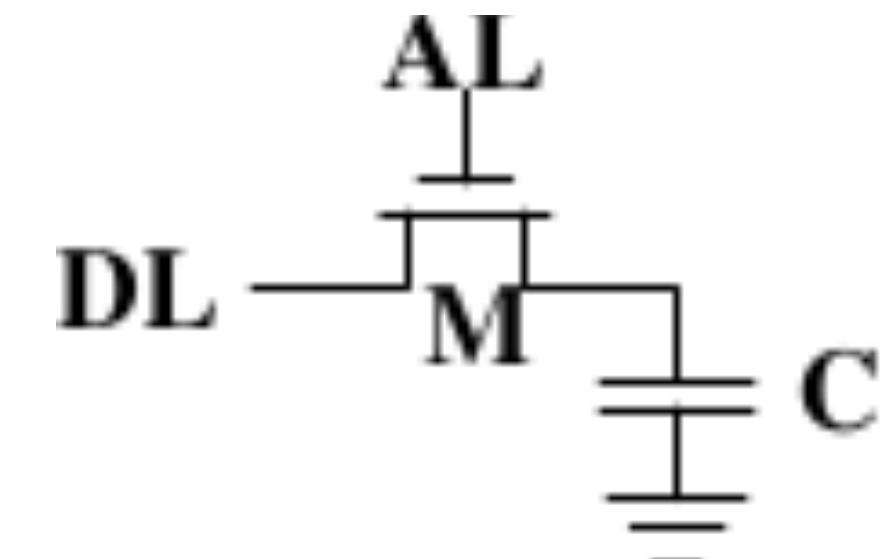
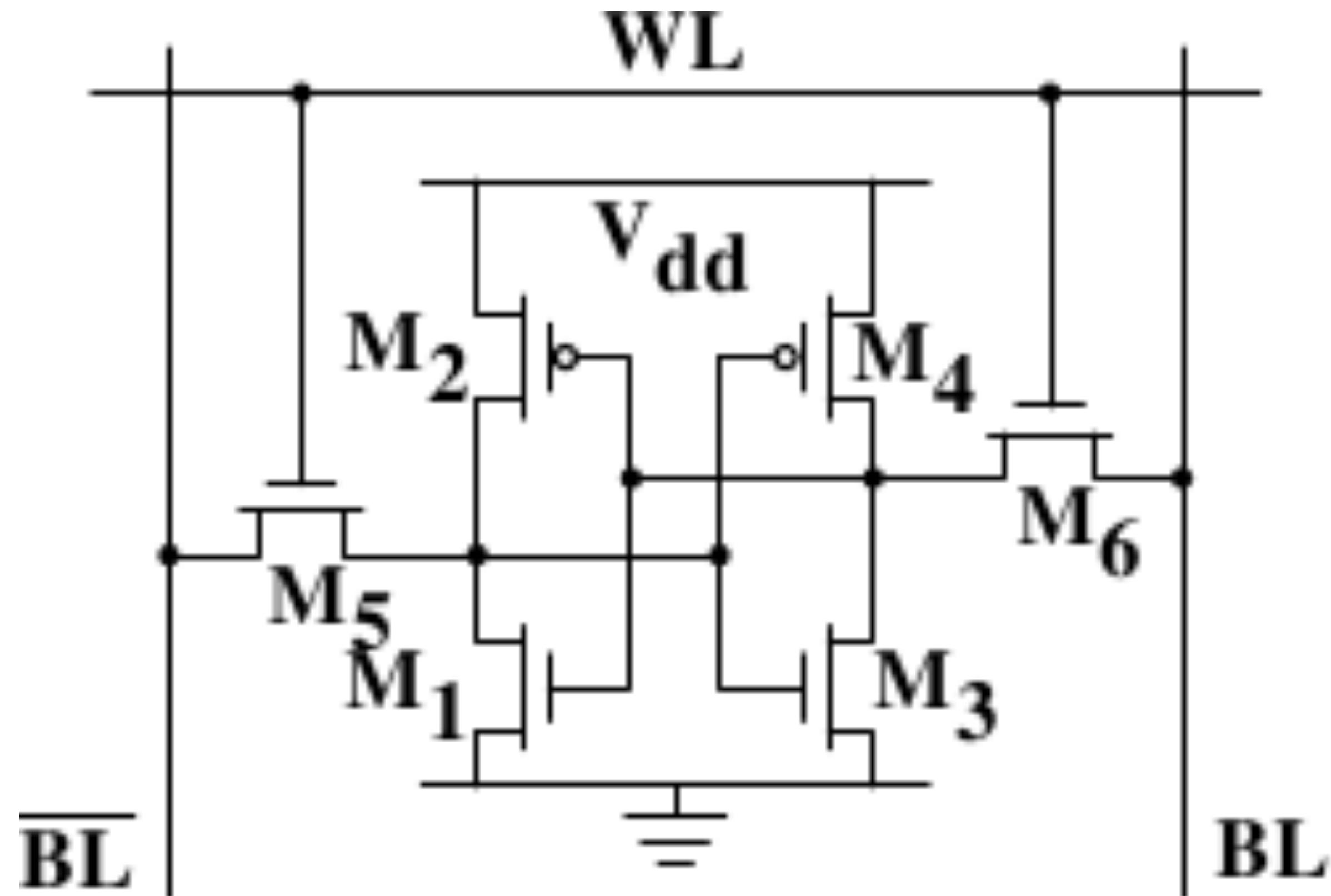
당신의 커리어 전환점 패스트캠퍼스

2017.06.19  
최동훈

# SRAM / DRAM

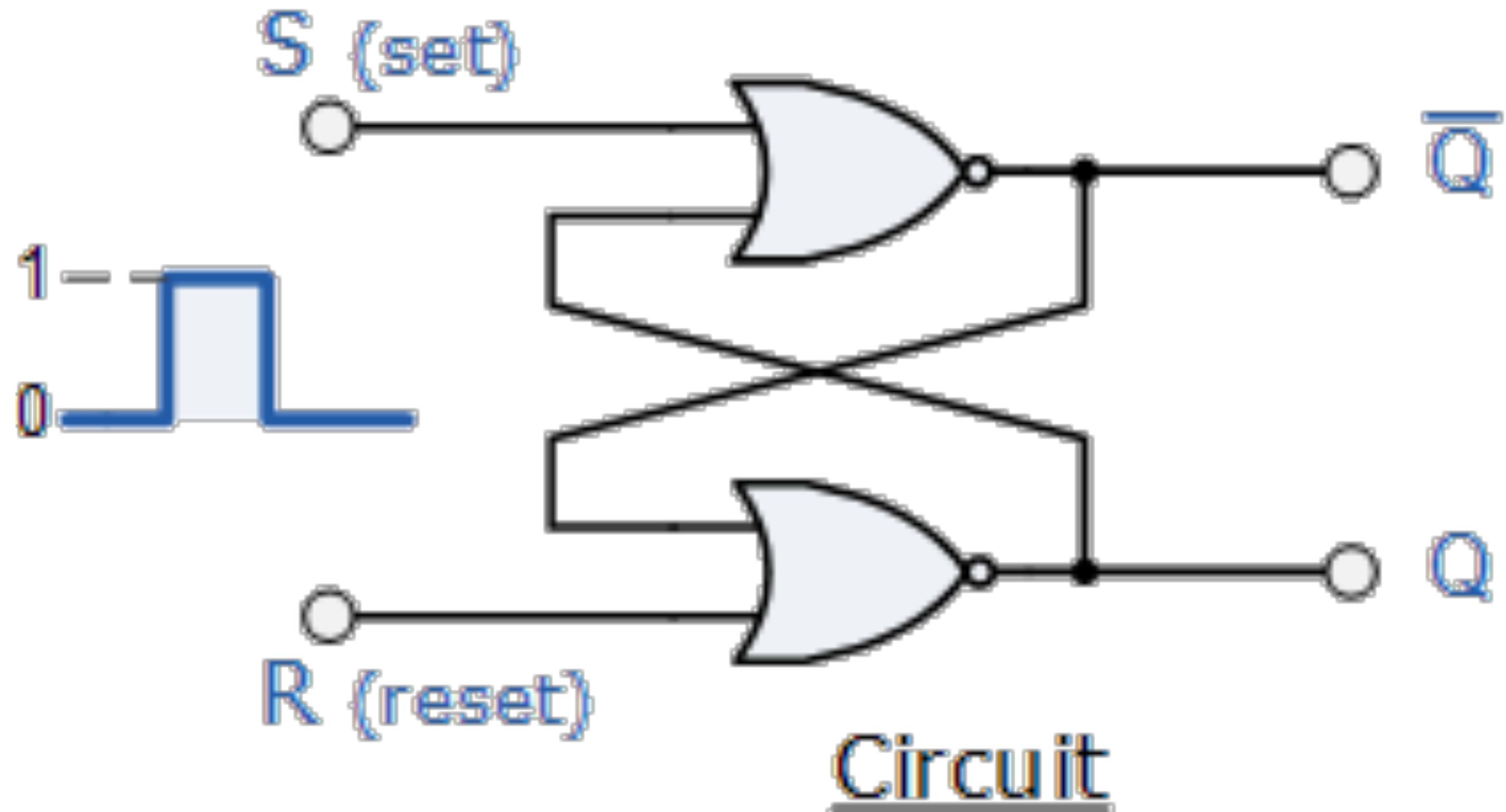
당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL



# SR 래치/플립플롭

당신의 커리어 전환점 패스트캠퍼스



S	R	Q	$\bar{Q}$
0	0	No change	
0	1	0	1
1	0	1	0
1	1	0	0
			(Invalid)

FRONTEND PROGRAMMING SCHOOL

# 캐시메모리 / 캐시 히트

- 캐시 히트란? 한 객체가 캐시에서 특정 데이터를 가져오려고 시도를 할때 특정 데이터가 존재 하는 경우
- 캐시 미스란? 한 객체가 캐시에서 특정 데이터를 가져오려고 시도를 할때 특정 데이터가 존재하지 않는 경우 -> 캐시 갱신 필요
- 생각해볼 문제 : 캐시는 CPU에만 존재할까?

# I/O

당신의 커리어 전환점 패스트캠퍼스

2017.06.19  
**최동훈**

FRONTEND PROGRAMMING SCHOOL

# 디스크

- 순차적 접근 - HDD (앞의 플로피 디스크 참조)
- 비순차적 접근 - SSD
- 차이의 근본적인 원인 - 기구적 설계
- 좋은걸 놔두고 안좋은걸 쓰는 이유 - 돈

# 직렬 / 병렬 통신

- 직렬 : 한번에 하나의 비트를 전송
- 병렬 : 한번에 다중 비트를 전송
- 컴퓨터의 역사를 살펴보면 병렬 통신이 직렬 통신에 자리를 내준다. 이유는 무엇일까?

# 버스의 역사

당신의 커리어 전환점 패스트캠퍼스

FRONTEND PROGRAMMING SCHOOL

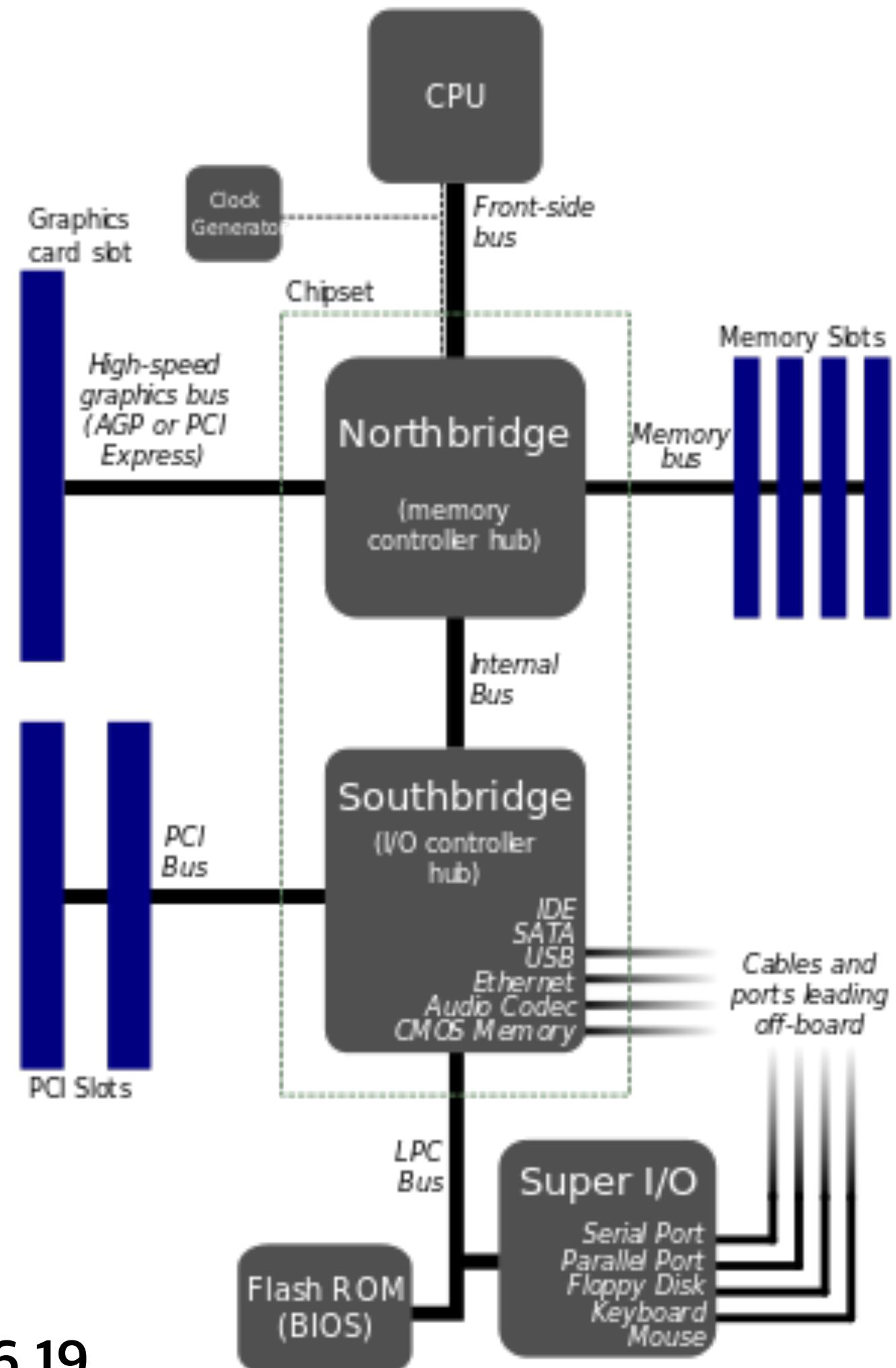
- ISA - EISA - VESA - PCI -AGP - PCI Express
- 각 시기별 한계를 극복하기 위해 새로운 버스 규격 등장

2017.06.19  
최동훈

# 컨트롤러

- I/O 기기에는 컨트롤러가 전부 내장 되어있다.
- 컨트롤러에는 다른 기기와 통신을 하기 위한 프로토콜이 내장 되어 있다.
- 컨트롤러에는 자체 I/O를 위한 자체 CPU가 내장되어 있다.

# 인텔 메인보드 아키텍쳐



## 노우스 브릿지

- 인텔 칩셋에서는 메모리 컨트롤 허브 (Memory Controller Hub, MCH)라고 부름
- CPU와 가깝다.
- CPU / RAM / VIDEO
- 고속용
- FSB와 관련

## 사우스 브릿지

- 인텔 칩셋에서는 입출력 컨트롤 허브 (I/O Controller Hub, ICH)라고 부름
- CPU와 멀다
- I/O용
- 상대적으로 저속

당신의 커리어 전환점 패스트캠퍼스

# THANK YOU :-)

FRONTEND PROGRAMMING SCHOOL

2017.06.19  
최동훈

---

FAST CAMPUS SCHOOL 2017  
Copyright FAST CAMPUS Corp. All Rights Reserved