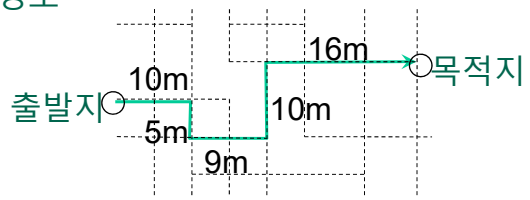
제1장
마이크로프로세서의 프로그램 실행원리

Youtube 주소
[1-1] http://youtu.be/yMFCQSAh6Ag
[1-2] http://youtu.be/QYknWA8dAL8

단순한 기계를 위한 프로그램 1

희망 이동 경로



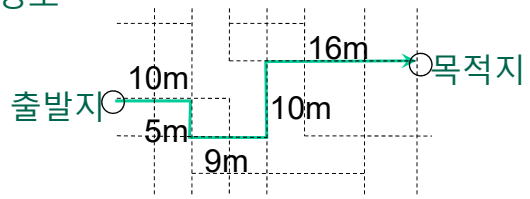
약간의 지능을 가진 주행로봇의 프로그램

```
전방으로 두 블록을 직진한 후에; 우회전을 하고;
한 블록을 직진한 후에; 좌회전을 하고;
두 블록을 직진한 후에; 좌회전을 하고;
한 블록을 직진한 후에; 우회전을 하고
전방으로 세 블록을 직진한다.
```

한 블록이란 ???

단순한 기계를 위한 프로그램 2

희망 이동 경로



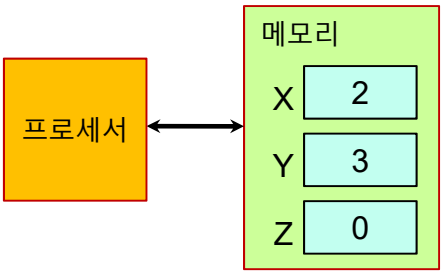
단순한 주행로봇의 프로그램

1미터 전진; 직전동작 9회 반복; 우회전;
1미터 전진; 직전동작 4회 반복; 우회전; 직전동작 2회 반복
1미터 전진; 직전동작 8회 반복; 우회전; 직전동작 2회 반복
1미터 전진; 직전동작 9회 반복; 우회전;
1미터 전진; 직전동작 15회 반복;

3개의 단순한 명령으로 모든 경우 표현 가능
반복 명령어가 없다면???

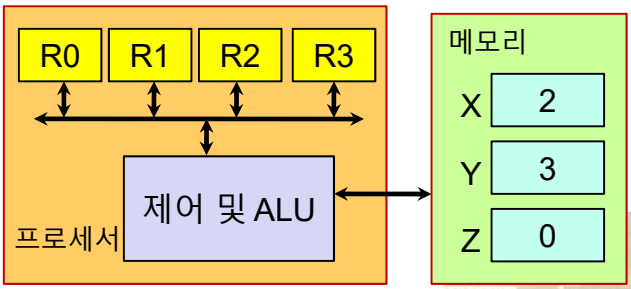
프로세서와 메모리

- 간단한 덧셈 프로그램
  Z = X + Y
- 가정
  프로세서는 메모리에 대해서
  읽기와 쓰기만 할 수 있다.
- 덧셈을 위한 실행절차
  1. 메모리의 X 번지의 내용 읽기
  2. 메모리의 Y 번지의 내용 읽기
  3. 읽어온 2개의 값을 더하기
  4. 더한 결과를 메모리의 Z 번지에 쓰기



프로세서와 내부의 레지스터

- 레지스터 (Register)
  - ✓작업 중간 결과를 임시로 저장하기 위한 장치
- 덧셈을 위한 실행 절차
  1. R1 ← M[X] *메모리의 X 번지 내용을 읽어와서 내부의 R1 레지스터에 저장한다*
  2. R2 ← M[Y]
  3. R3 ← R1 + R2
  4. M[Z] ← R3



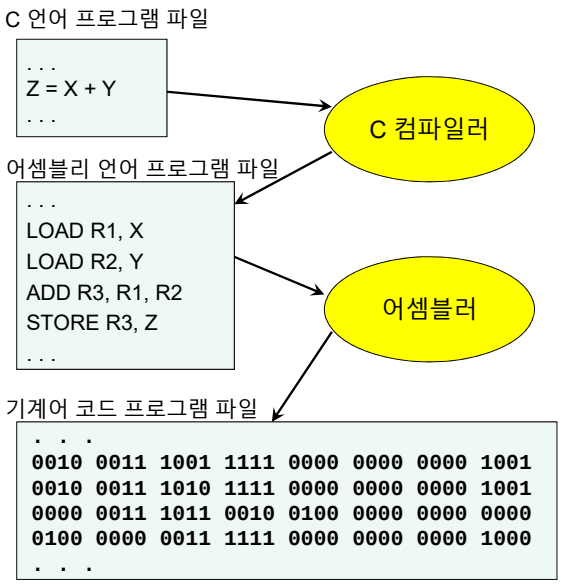
어셈블리 언어 프로그램

```
LOAD R1, X
LOAD R2, Y
ADD R3, R1, R2
STORE R3, Z
```

```
R1 ← M[X]
R2 ← M[Y]
R3 ← R1 + R2
M[Z] ← R3
```

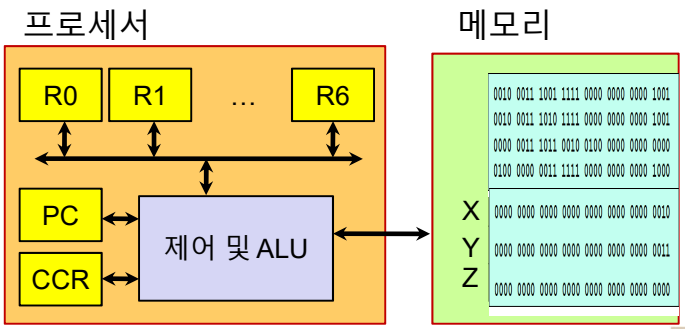
- 명령어 (Instruction)
  - ✓ LOAD : 메모리 특정주소의 내용을 레지스터로 읽어 들이기
  - ✓ STORE : 레지스터 내용을 메모리 특정주소에 기록하기
  - ✓ ADD : 프로세서 내부에서 덧셈 작업 수행
- R1 대신 R0를 사용하면?
- **컴파일러** (compiler) : 고급수준의 프로그램을 대상 프로세서의 어셈블리 언어 프로그램으로 번역
- **어셈블러** (assembler) :어셈블리언어 프로그램을 기계어 코드 프로그램으로 변환

컴파일러와 어셈블러의 역할

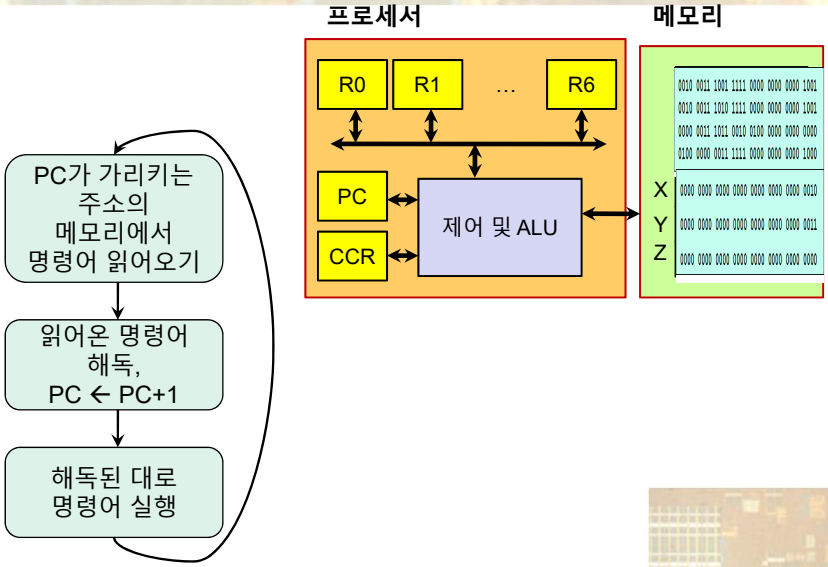


프로세서의 명령어 실행

- PC (Program Counter)
- CCR (Condition Code Register)



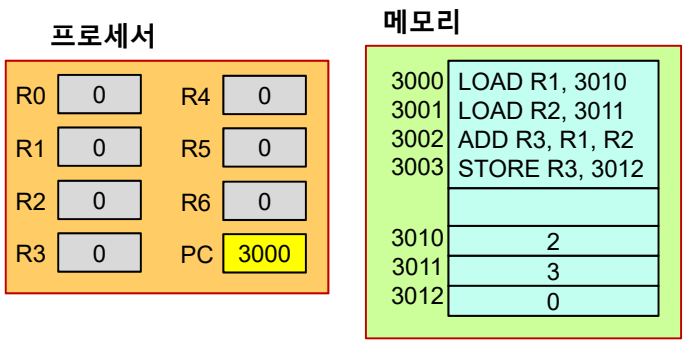
명령어 실행과정



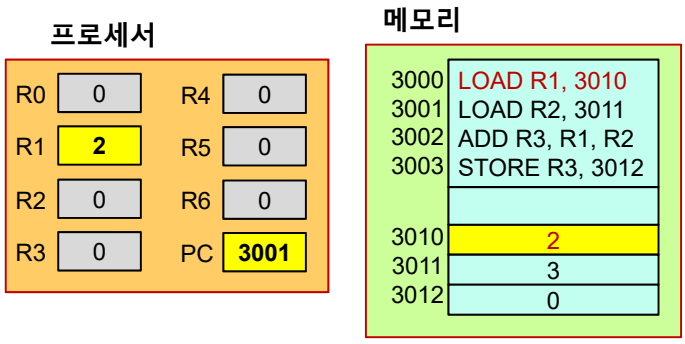
메모리 상의 프로그램 내용

메모리 주소	메모리 내용	의미
3000	LOAD R1, 3010	R1 ← M[X]
3001	LOAD R2, 3011	R2 ← M[Y]
3002	ADD R3, R1, R2	R3 ← R1 + R2
3003	STORE R3, 3012	M[Z] ← R3
3004	…	
…	…	
3010	2	변수 X의 메모리 내용
3011	3	변수 Y의 메모리 내용
3012	0	변수 Z의 메모리 내용

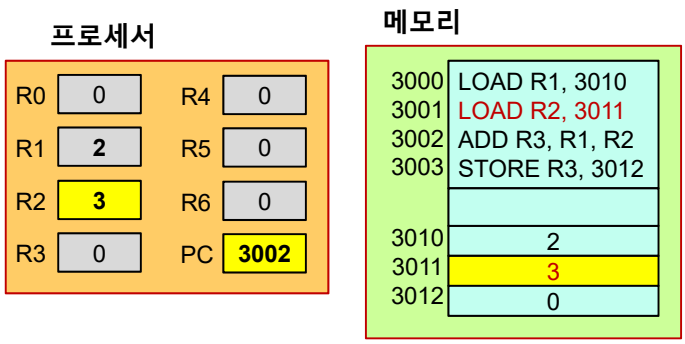
프로그램 실행 전



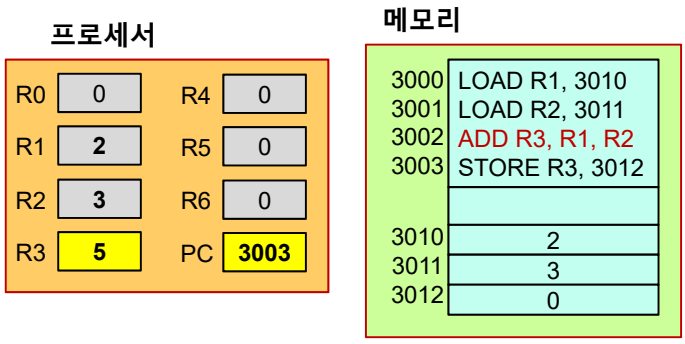
‘LOAD R1, 3010’ 실행 후



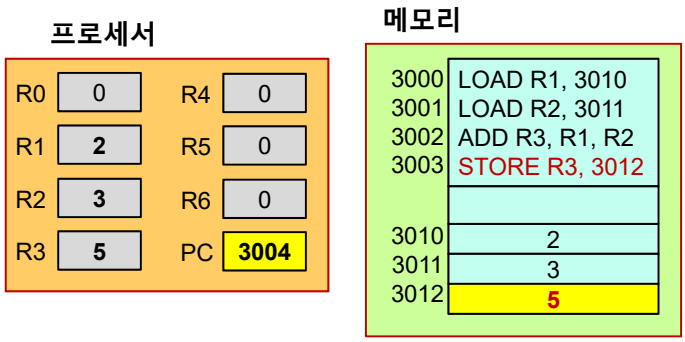
‘LOAD R2, 3011’ 실행 후



‘ADD R3, R1, R2’ 실행 후



‘STORE R3, 3012’ 실행 후



프로그램 실행과정에서 메모리 읽기와 쓰기

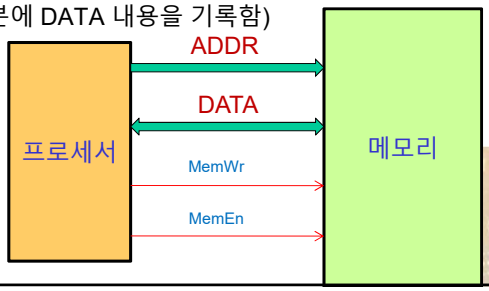
- 3000번지 읽기 (명령어 읽기)
- 3010번지 읽기 (LOAD 실행)
- 3001번지 읽기 (명령어 읽기)
- 3011번지 읽기 (LOAD 실행)
- 3002번지 읽기 (명령어 읽기)
- (ADD 실행)
- 3003번지 읽기 (명령어 읽기)
- 3012번지 쓰기 (STORE 실행)

➔ 읽기 6번, 쓰기 1번

메모리 주소	메모리 내용
3000	LOAD R1, 3010
3001	LOAD R2, 3011
3002	ADD R3, R1, R2
3003	STORE R3, 3012
3004	...
...	...
3010	2
3011	3
3012	0

(참고) 프로세서의 메모리 읽기와 쓰기

- 메모리 읽기 처리 절차
  - t1.1: ADDR 설정, MemWr 은 READ (0)로 설정
  - t1.2: MemEn 활성화 (1)
    (메모리에서 ADDR 주소의 내용이 DATA 부분으로 전달됨)
  - t2.1: DATA 의 값을 프로세서 내부의 지정된 레지스터에 기록
  - t2.2: MemEn 비활성화 (0)
- 메모리 쓰기 처리 절차
  - t1.1: ADDR 설정, DATA 설정, MemWr은 WRITE (1)로 설정
  - t1.2: MemEn 활성화 (1)
    (메모리는 ADDR 주소 부분에 DATA 내용을 기록함)
  - t2.1: MemEn 비활성화 (0)



메모리 상의 기계어 코드와 데이터

주소	기계어 코드	의미
3000	001000 111 001 111 1 0000 0000 0000 1001	LOAD R1, 3010
3001	001000 111 010 111 1 0000 0000 0000 1001	LOAD R2, 3011
3002	000000 111 011 001 0 0100 0000 0000 0000	ADD R3, R1, R2
3003	010000 000 011 111 1 0000 0000 0000 1000	STORE R3, 3012
3004	...	...
...	...	...
3010	0000 0000 0000 0000 0000 0000 0000 0010	2
3011	0000 0000 0000 0000 0000 0000 0000 0011	3
3012	0000 0000 0000 0000 0000 0000 0000 0000	0

비부호 정수의 표현 범위

비트수	10진수 표현	2진수 표현	16진수 표현
4	0 ~ 15	0000B ~ 1111B	0x0 ~ 0xF
8	0 ~ 255	00000000B ~ 11111111B	0x00 ~ 0xFF
16	0 ~ 65,535	0000000000000000B ~ 1111111111111111B	0x0000 ~ 0xFFFF

8비트 정수의 표현 범위

수의 범위	2진수 표현	16진수 표현
1 ~ 127	00000001B ~ 01111111B	0x01 ~ 0x7F
0	00000000B	0x00
-128 ~ -1	10000000B ~ 11111111B	0x80 ~ 0xFF

8비트 정수의 2의 보수 표현

정수 값	2진수 표현
127	0111 1111
126	0111 1110
125	0111 1101
124	0111 1100
...	...
3	0000 0011
2	0000 0010
1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
-3	**1111 1101**
...	...
-126	1000 0010
-127	1000 0001
-128	1000 0000

(참고)
8 비트 x의 2의보수 (2's complement)
→ *x와 더해서 2^8이 되는 수*
→ *2^8 - x*
→ *$(2^8 - 1) - x + 1$*
→ *11111111 – x + 1*
→ *x의 1의 보수 + 1*
→ *각 비트를 반대로 한 값 + 1*

n 자리 r진수 x의 r의 보수
→ *x 와 더해서 r^n이 되는 수*
→ *r^n - x*

8비트 수의 정수와 비부호 정수 값

2진수 표현	비부호 정수 값	정수 값
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
...	...	...
0111 1110	126	126
0111 1111	127	127
1000 0000	**128**	**-128**
1000 0001	**129**	**-127**
...	...	...
1111 1110	**254**	**-2**
1111 1111	**255**	**-1**

데이터의 유형의구별 방법???
- 2진수 자체만으로는 구별할 수 없고
-특정 메모리 주소별로 어떤 의미로 쓰이는지 정한 대로

비트 수 확장1 - 비부호 정수를 위한 제로 확장

- 연산 시에 자리 수를 통일해야 함
- 비부호 정수의 비트 확장 (제로확장)

```
  0001 0001 (17)
+      1001  (9)
-------------------
       (26)
```

⇒

```
  0001 0001 (17)
+ 0000 1001  (9)
-------------------
  0001 1010 (26)
```

- 정수의 제로확장 오류

```
  0001 0001 (17)
+      1001 (-7)
-------------------
       (10)
```

⇒

```
  0001 0001 (17)
+ 0000 1001  (9)
-------------------
  0001 1010 (26)
```


비트 수 확장2 - 정수를 위한 부호 확장

- 양수의 경우

```
  0001 0001 (17)
+      0111  (7)
-------------------
```

⇒

```
  0001 0001 (17)
+ 0000 0111  (7)
-------------------
  0001 1000 (24)
```

- 음수의 경우

```
  0001 0001 (17)
+      1001 (-7)
-------------------
```

⇒

```
  0001 0001 (17)
+ 1111 1001 (-7)
-------------------
  0000 1010 (10)
```

(참고) C 언어에서 비트 수 확장

- 변수 선언
  - ✓short x; long y;
  - ✓unsigned short ux; unsigned long uy;
  - ✓ ㈜ *32비트 CPU에서는 보통 short는 16비트, int/long은 32비트임*
  - ✓ ㈜ *64비트 CPU에서는 보통 short 16 bit, int 32 bit, long 64 bit*
  - ✓ ㈜ *Java에서는 unsigned가 없고, byte(8bit), short, int, long(64bit)*
- y = y + x;
  - ✓x와 y는 (부호가 있는)정수로 선언되었으므로
  - ✓x 의 값을 **부호확장**한 후에 y의 값과 더해서 y에 저장하도록
- uy = uy + ux
  - ✓ux와 uy는 비부호 정수로 선언되었으므로
  - ✓ux 의 값을 **제로확장**한 후에 uy의 값과 더해서 uy에 저장하도록

덧셈과 캐리의 발생 (비부호 정수 연산)

- 비부호 정수의 표현 가능한 범위 위반 표시
  →2진 덧셈으로 최상위 비트에서 발생하는 Carry 비트를 별도로 저장
- 8비트 비부호 정수 덧셈의 예

```
      0100 0001 (65)          0100 0001 (65)
+     0100 0010 (66)    +     1100 0010(194)
-------------------     -------------------
C=0   1000 0011(131)    C=1   0000 0011  (3)
```

정상 / 오류

(참고) 덧셈과 캐리의 활용 (큰 수의 덧셈)

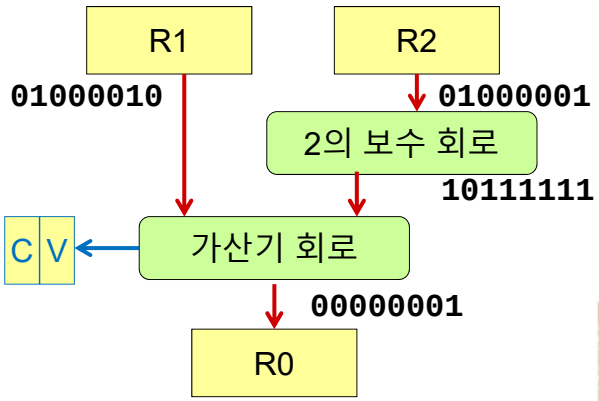
- 예: 32비트 프로세서에서 128비트로 표현해야 하는 정수의 덧셈

→ 32비트 정수 4개 사용

- 최하위 32비트 정수 간에 덧셈을 한 후에
- 차상위 32비트 정수간에 덧셈을 할 때 직전의 캐리 값도 더함

(참고) ADD와 SUB 명령어를 위한 회로

- 감산회로는 2의 보수를 더하는 것으로 처리
- 정수와 비부호 정수 계산을 동일한 회로로 처리
- 'R0 ← R1 - R2' 연산의 처리



뺄셈과 캐리의 발생 (비부호 정수 연산)

- 감산회로는 2의 보수를 더하는 것으로 설계
- 캐리가 있으면 정상적인 값
- 캐리가 없으면 표현범위 초과 (borrow 발생을 의미)

```
  0100 0010 (66)             0100 0010   .
- 0100 0001 (65)     →   +   1011 1111   .
----------------         ------------------
            (1)        C=1   0000 0001  (1)   정상

  0100 0001 (65)             0100 0001   .
- 0100 0010 (66)     →   +   1011 1110   .
----------------         ------------------
       (표현불가)        C=0   1111 1111(255)  오류
```


(참고) 뺄셈과 캐리의 활용

- 1. 천문학적으로 큰 수의 뺄셈
- 2. 비부호 정수에 대한 조건문의 처리

✓비부호 정수 x에 대하여 99보다 크거나 같으면

→ if (x >= 99)

→ x - 99 연산과정에서 Carry 가 1이면 (borrow가 0 이면)

㈜ 뺄셈 명령에서 C 비트 값이 CCR에 저장되고, 다음 명령에서 이 C 비트 값에 따라 분기 (Branch) 여부를 결정함

오버플로우의 발생 (정수 연산)

- 뺄셈은 2의 보수를 더하는 것으로 처리
- Overflow: 정수의 표현 가능한 범위 위반 표시
  →2진 덧셈으로 발생하는 Overflow 비트도 별도로 저장
- 8비트 정수 덧셈의 예

```
      0001 0100 (20)          0010 0001 (33)
    +  1111 1100 (-4)       +  0010 0000 (32)
V=0   0001 0000 (16)  정상  V=0   0100 0001 (65)  정상

      0100 0001  (65)         1011 1111 (-65)
    +  0100 0000  (64)      +  1100 0000 (-64)
V=1   1000 0001(-127)       V=1  0111 1111 (127)
                  오류                          오류
```


(참고) 뺄셈과 오버플로우의 활용

- 정수에 대한 조건문의 처리
  ✓정수 x에 대하여 99보다 크거나 같으면
  → if (x >= 99)
  → x - 99 연산과정에서 결과가 Overflow가 0이면서 결과가 음이 아니거나, Overflow가 1이면서 결과가 음이면
  ㈜ 뺄셈 명령에서 V 비트 값이 CCR에 저장되고, 다음 명령에서 이 V 비트 값에 따라 분기 (Branch) 여부를 결정함

- (참고) 비부호 정수에 대한 조건문의 처리
  ✓x 가 비부호 정수라면
  → if (x >= 99)
  → x - 99 연산과정에서 Carry 가 1이면 (borrow가 0 이면)

8비트 연산결과의 의미 비교

		정수	비부호정수
	0001 0100	20	20
+	1111 1100	-4	252
	0001 0000	16	16
	V=0, C=1	V=0	C=1

		정수	비부호정수
	0010 0001	33	33
+	0010 0000	32	32
	0100 0001	65	65
	V=0, C=0	V=0	C=0

		정수	비부호정수
	0100 0001	65	65
+	0100 0000	64	64
	1000 0001	-127	129
	V=1, C=0	V=1	C=0

		정수	비부호정수
	1011 1111	-65	191
+	1100 0000	-64	192
	0111 1111	127	127
	V=1, C=1	V=1	C=1

프로세서는 데이터 종류에 상관없이 C와 V 비트를 결정하여 CCR에 저장

논리 연산 진리표

입력		출력			
A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

논리 연산의 예

	1100 0011			1100 0011	
AND	0101 0101		**OR**	0101 0101	
	0100 0001			1101 0111	
	1100 0011				
XOR	0101 0101		**NOT**	1100 0101	
	1001 0110			0011 1010	

ASCII 문자 표현

- ✓ nul : null
- ✓ bel : bell
- ✓ sp : space
- ✓ bs : back space
- ✓ ht : horizontal tab
- ✓ nl : new line
- ✓ cr : carriage return
- ✓ esc : escape
- ✓ del : delete

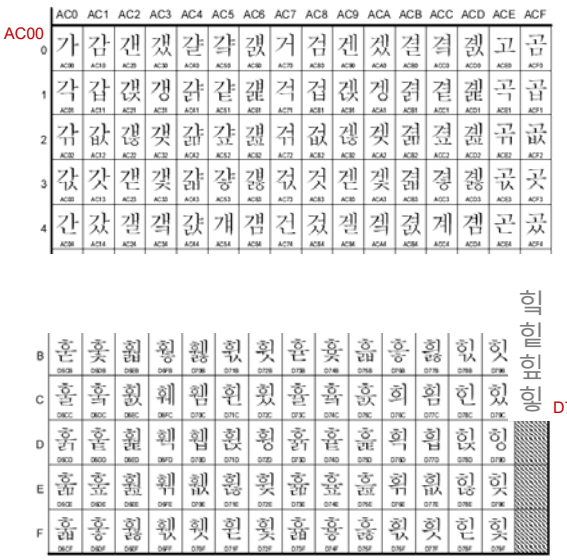
문자	16진수	10진수	문자	16진수	10진수	문자	16진수	10진수	문자	16진수	10진수
nul	00	0	sp	20	32	@	40	64	`	60	96
soh	01	1	!	21	33	A	41	65	a	61	97
stx	02	2	"	22	34	B	42	66	b	62	98
etx	03	3	#	23	35	C	43	67	c	63	99
eot	04	4	$	24	36	D	44	68	d	64	100
enq	05	5	%	25	37	E	45	69	e	65	101
ack	06	6	&	26	38	F	46	70	f	66	102
bel	07	7	'	27	39	G	47	71	g	67	103
bs	08	8	(	28	40	H	48	72	h	68	104
ht	09	9	)	29	41	I	49	73	i	69	105
nl	0a	10	*	2a	42	J	4a	74	j	6a	106
vt	0b	11	+	2b	43	K	4b	75	k	6b	107
np	0c	12	,	2c	44	L	4c	76	l	6c	108
cr	0d	13	-	2d	45	M	4d	77	m	6d	109
so	0e	14	.	2e	46	N	4e	78	n	6e	110
si	0f	15	/	2f	47	O	4f	79	o	6f	111
dle	10	16	0	30	48	P	50	80	p	70	112
dc1	11	17	1	31	49	Q	51	81	q	71	113
dc2	12	18	2	32	50	R	52	82	r	72	114
dc3	13	19	3	33	51	S	53	83	s	73	115
dc4	14	20	4	34	52	T	54	84	t	74	116
nak	15	21	5	35	53	U	55	85	u	75	117
syn	16	22	6	36	54	V	56	86	v	76	118
etb	17	23	7	37	55	W	57	87	w	77	119
can	18	24	8	38	56	X	58	88	x	78	120
em	19	25	9	39	57	Y	59	89	y	79	121
sub	1a	26	:	3a	58	Z	5a	90	z	7a	122
esc	1b	27	;	3b	59	[	5b	91	{	7b	123
fs	1c	28	<	3c	60	₩	5c	92	|	7c	124
gs	1d	29	=	3d	61	]	5d	93	}	7d	125
rs	1e	30	>	3e	62	^	5e	94	~	7e	126
us	1f	31	?	3f	63	_	5f	95	del	7f	127

ASCII 문자 표현

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(	38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29	)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

한글 코드 (16비트)

AC00 Hangul Syllables ACFF



Unicode
ISO/IEC 10646-1
KS X 10025-1

(참고) 텍스트 파일과 이진 파일

- 0과 1의 기록: 하드디스크는 N극의 방향, RAM/Flash는 전압
- 텍스트(Text) 파일: 모든 내용을 문자 코드로만 기록
  - ✓일반 텍스트파일 편집기 (notepad/메모장, vi, emacs, …
  - ✓프로그램 통합 개발환경의 편집기 (eclipse, visual studio, …)
  - ✓…
- 이진(Binary) 파일: 텍스트 파일이 아닌 것 전부
  - ✓한글, MS Word, 엑셀, 파워포인트, …
  - ✓오디오 (MPEG3, …)
  - ✓영상 (…)
  - ✓어셈블러 출력 (기계어 코드 파일)
  - ✓…

기타 정보의 표현

- 부동소수점 (Wikipedia floating point 항목 참조)
  - ✓ 32비트 표현: 부호 1비트, 지수 8비트, 유효숫자 23비트
  - ✓ 64비트 표현: 부호 1비트, 지수 11비트, 유효숫자 52비트
- 그림
  - ✓흑백 그림: 픽셀당 1비트
  - ✓컬러 그림: 픽셀당 red 8비트, green 8비트, blue 8비트, 기타
- 소리
  - ✓고정 소수점으로 표현

- *(참고) 영문 대문자를 소문자로 변환하는 방법*
  - ✓*0x20을 더해주거나, 끝에서 6번째 비트만 1로 변경한다*
  - ✓*소문자를 대문자로 변환하는 방법은 ???*