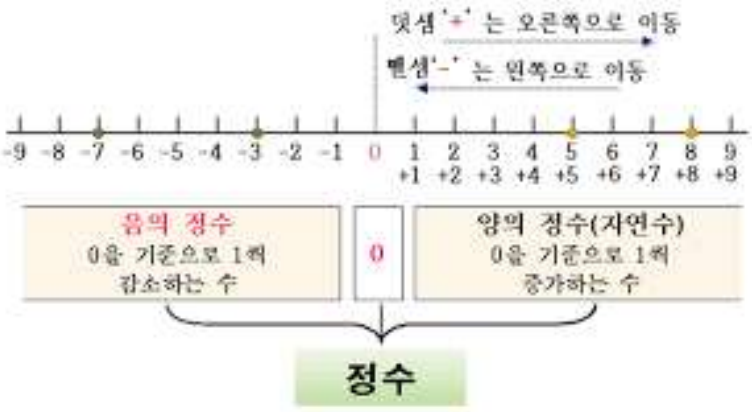
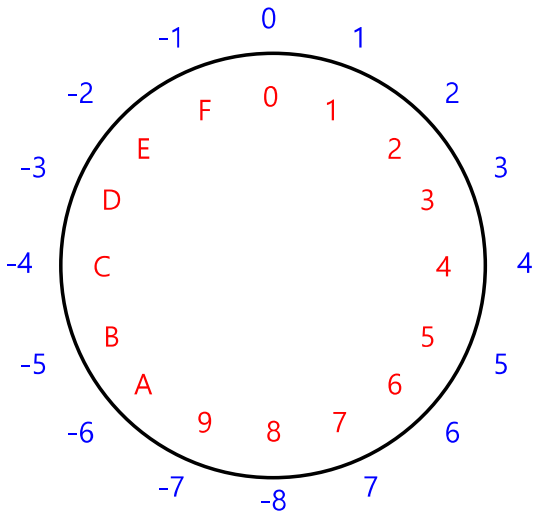
보수; 보충하는 수; Complement Number System
컴퓨터에서 음수를 표현하기 위해서 보수라는 개념을 사용
ALU에 뺄셈 회로가 없었음. 보수를 더하는 것이 뺄셈과 같은 효과

수학에서의 수체계와 컴퓨터에서의 수체계



수학에서는 무한대의
수가 존재



컴퓨터에서는
유한개의 수가 존재
(순환 버퍼의 개념)

02 정수 표현

1 보수의 개념과 음수

❖ 최상위비트(MSB)를 부호비트로 사용
- 양수(+) : 0 음수(-) : 1

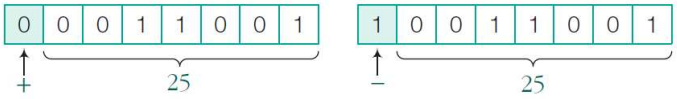
❖ 2진수 음수를 표시하는 방법
- 부호와 절댓값(sign-magnitude)
- 1의 보수(1's complement)
- 2의 보수(2's complement)

그림 2-1 비트, 니블, 바이트의 관계

MSB Most Significant Bit: 최상위 비트
LSB Least Significant Bit: 최하위 비트

컴퓨터에서는 음수를 나타내기 위해 2의 보수 방식을 사용함

❖ **부호와 절댓값**
- 부호비트만 양수와 음수를 나타내고 나머지 비트들은 같다.



❖ **1의 보수 방식**
- 0 → 1, 1 → 0으로 변환

0 0 0 0 0 0 1 1
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 1 1 1 1 1 0 0 ←1의 보수

❖ **2의 보수 방식**
- 1의 보수 + 1 = 2의 보수

0 0 0 0 0 0 1 1
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 1 1 1 1 1 0 0 ←1의 보수
\+ 1
1 1 1 1 1 1 0 1 ←2의 보수

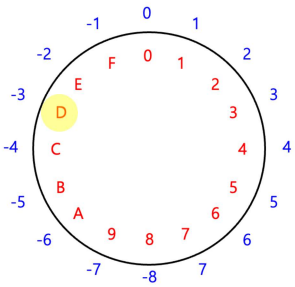
02 정수 표현

❖ r진수에는 r의 보수와 r-1의 보수가 있다.
- 10진수에는 9의 보수와 10의 보수가 있고
  8진수에는 7의 보수와 8의 보수가 있으며
  2진수에는 1의 보수와 2의 보수가 있다.
- 567의 9의 보수 : 999-567=432
- 567의 10의 보수 : 1000-567=433 (=432+1; 567의 9의 보수에 +1)
- 0011의 1의 보수 : 1111-0011=1100
- 0011의 2의 보수 : 10000-0011=1101 (=1100+1; 0011의 1의 보수에 +1)

컴퓨터 유한개의 수 체계에서는
❖ **양수를 보수로 바꾸면 음수: 0011(+3) -> 1101(-3)**
❖ **음수를 보수로 바꾸면 양수: 1101(-3) -> 0011 (+3)**

D

양수 ⟷ (보수) 음수 (보수)



컴퓨터에서는
유한개의 수가 존재
(순환 버퍼의 개념)

❖ 2진수와 그 수의 1의 보수와의 합은 모든 bit가 1이 된다. (0011 + 1100 = 1111)
❖ 2진수와 그 수의 2 보수와의 합은 모든 bit가 0이 된다. (0011 + 1101 = 10000)
(자릿수를 벗어나는 비트는 제외)

뺄셈 == 보수를 이용한 덧셈

- 10진수 한자리 숫자의 경우 숫자 3에 10의 보수는 7
- 5 – 3 = 2
- 5 + 7 = 2 (12는 두자리 숫자)
- 이때, 올림(carry) 발생하여 다음 자리로 숫자가 넘어 가는 것임

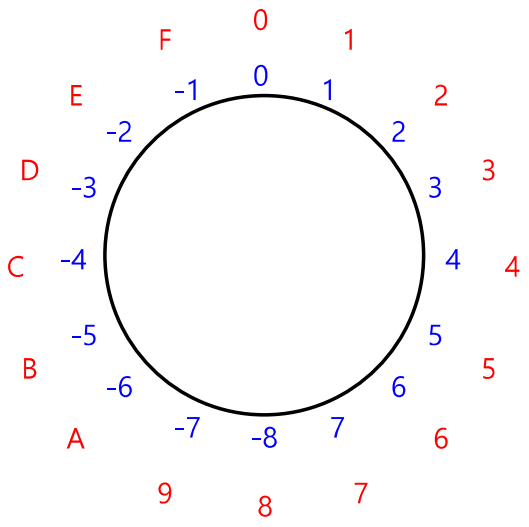
- 16진수 한자리 숫자의 경우 숫자 9에 16의 보수는 7
- A – 9 = 1
- A + 7 = 1

- 2진수 네자리 숫자의 경우 숫자 1001에 2의 보수는 0111
- 1100 – 1001 = 0011
- 1100 + 0111 = 0011

02 정수 표현

❖ 수의 표현 방법에 따른 10진수 대응 값

4비트 2진수	부호 없는 수	부호와 절댓값	1의 보수	2의 보수
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1



데이터형(Data Type) 나온 이유?

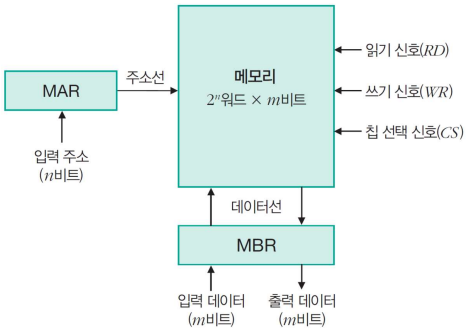


그림 6-2 주기억 장치의 동작 블록도

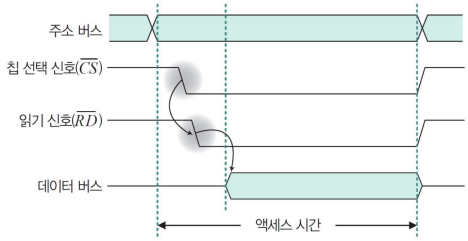


그림 6-3 메모리 읽기 동작의 타이밍도

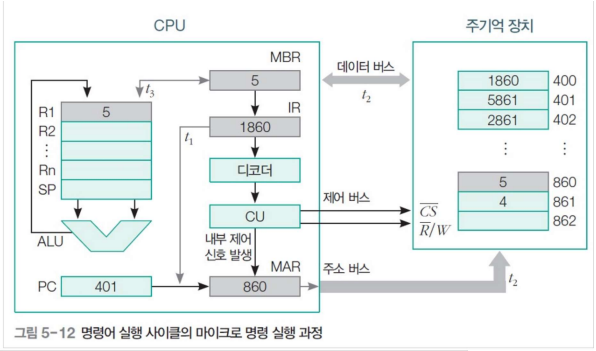


그림 5-12 명령어 실행 사이클의 마이크로 명령 실행 과정

t_1: MAR ← (IR:operand) ; 명령 레지스터의 오퍼랜드(860)를 MAR로
t_2: MBR ← M[MAR] ; 메모리에서 피연산자를 읽어 MBR로
t_3: R1 ← (MBR) ; MBR에서 R1(누산기 역할)로

컴퓨터 자원을 효율적으로 사용하기 위해 컴파일러가 적절한 명령으로 변환할 수 있도록 사용자로 하여금 데이터형을 먼저 정의하도록 함

다음 중 인원수에 따른 가장 효율적인 버스 배차는?

02 정수 표현

❖ 2의 보수에 대한 10진수의 표현 범위

표 2-4 n비트 2의 보수에 대한 10진수의 표현 범위

비트 수	2의 보수를 사용한 2진 정수의 표현 범위
n비트	$-2^{n-1} \sim +2^{n-1}-1$
4비트	$-2^{4-1}(-8) \sim +2^{4-1}-1(+7)$
8비트	$-2^{8-1}(-128) \sim +2^{8-1}-1(+127)$
16비트	$-2^{16-1}(-32,768) \sim +2^{16-1}-1(+32767)$
32비트	$-2^{32-1}(-2,147,483,648) \sim +2^{32-1}-1(+2,147,483,647)$
64비트	$-2^{64-1}(-9,223,372,036,854,775,808) \sim +2^{64-1}-1(+9,223,372,036,854,775,807)$

2 부호 확장

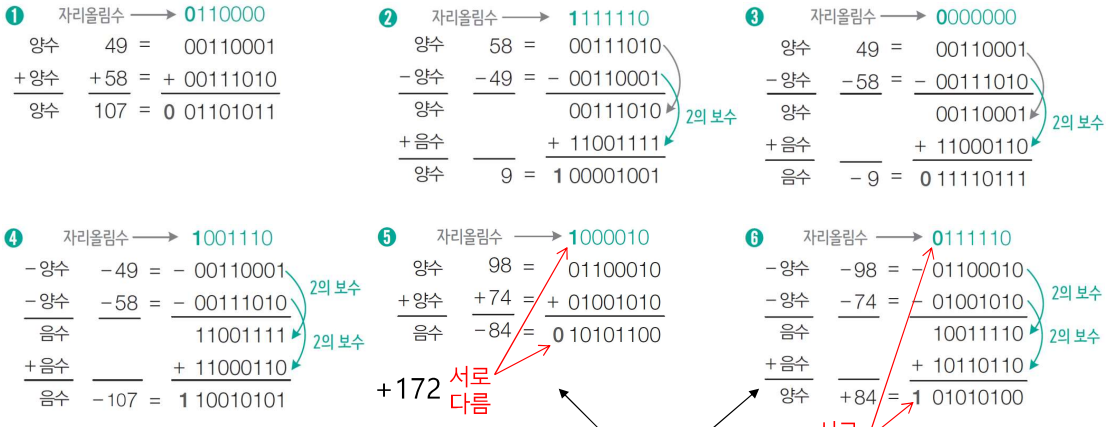
- 부호 확장이란 늘어난 비트 수 만큼 부호를 늘려주는 방법

표 2-5 2진수 표현 방법에 따른 부호 확장

2진수 표현 방식	부호 확장 방법	구분	8비트	16비트 확장
부호와 절댓값	부호만 MSB에 복사하고, 나머지는 0으로 채운다.	양수	00101010	**00000000** 00101010
		음수	10010111	**10000000** 00010111
1의 보수	늘어난 길이만큼 부호와 같은 값으로 모두 채운다.	양수	00101010	**00000000** 00101010
		음수	10010111	**11111111** 10010111
2의 보수	늘어난 길이만큼 부호와 같은 값으로 모두 채운다.	양수	00101010	**00000000** 00101010
		음수	10010111	**11111111** 10010111

3 2진 정수 연산

- 뺄셈의 원리를 보면, A-B 대신에 A+(B의 2의 보수)를 계산하면 된다.
- 뺄셈에서 2의 보수 방식을 사용하는 이유는 뺄셈을 가산기를 사용하여 수행할 수 있기 때문



signed byte(8비트)
표현할 수 있는 수의 범위는
-128 ~ 127

32비트	$-2^{32-1}(-2,147,483,648) \sim +2^{32-1}-1(+2,147,483,647)$

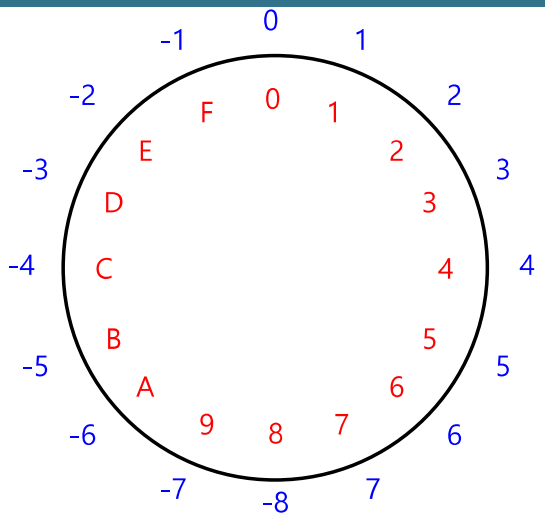
signed int의 범위를 벗어나면
(개발자에게는 등골이 오싹한) overflow;underflow 발생

02 정수 표현

❖ 수의 표현 방법에 따른 10진수 대응 값

4비트 2진수	부호 없는 수	2의 보수
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

Underflow ↑ ↓ Overflow



Underflow;언더플로우
최소값보다 낮아지지 못하고 최대값으로 가는 경우

Overflow;오버플로우
최대값보다 높아지지 못하고 최소값으로 가는 경우

overflow, underflow

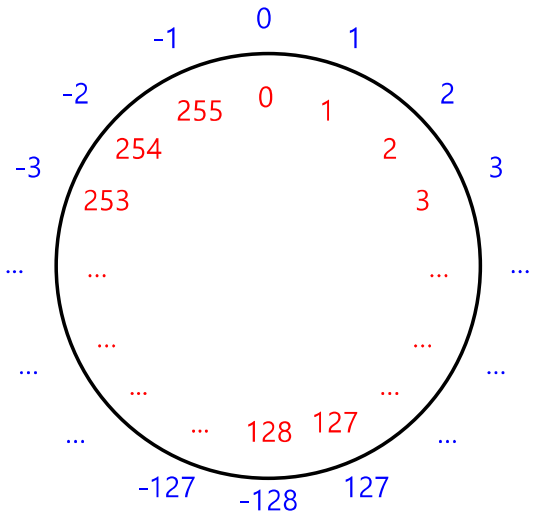
```c
unsigned char ubVar=0;
signed char bVar=0;

for(int i=0; i<257; i++)
{
    printf("%d ", ubVar);
    ubVar++;
}
printf("\n");

for(int i=0; i<257; i++)
{
    printf("%d ", bVar);
    bVar++;
}
printf("\n");
```

```c
ubVar = 255;
for(int i=0; i<257; i++)
{
    printf("%d ", ubVar);
    ubVar--;
}
printf("\n");

bVar =255;
for(int i=0; i<257; i++)
{
    printf("%d ", bVar);
    bVar--;
}
```



Underflow;언더플로우

Overflow;오버플로우

overflow, underflow