디지털 시스템 및 마이크로 컴퓨터 II

(2주차)

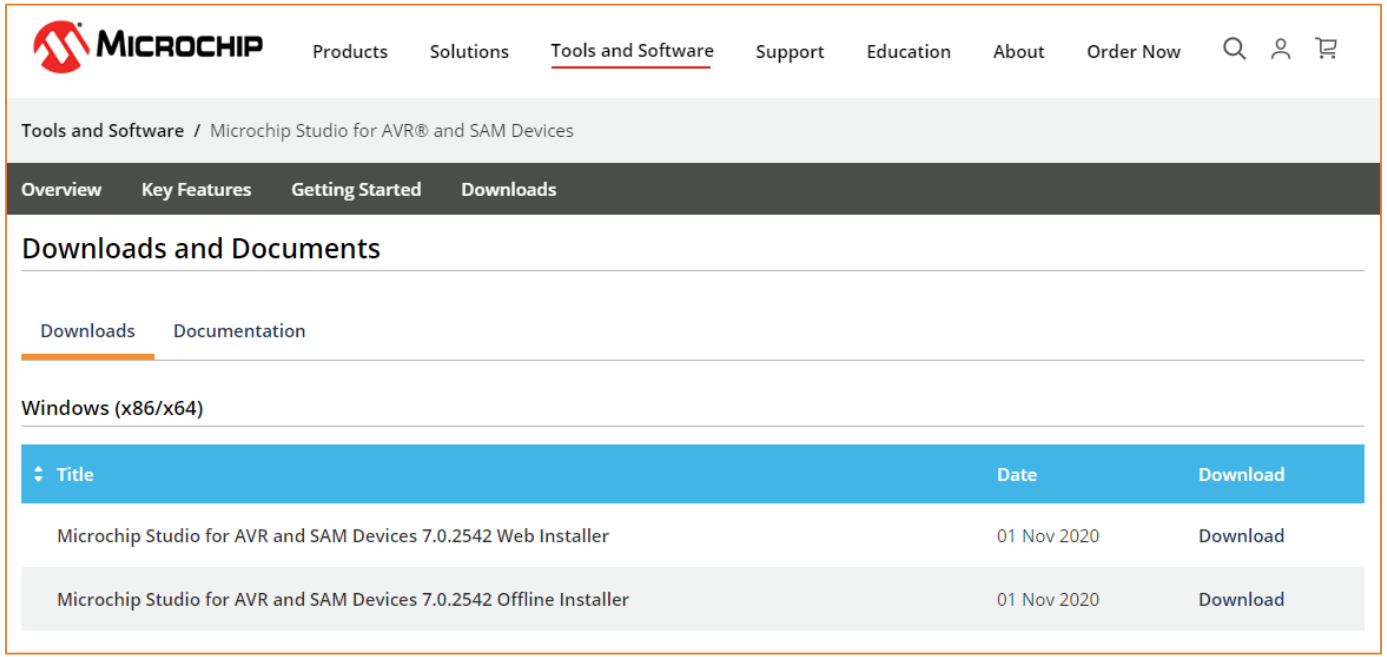
인제대학교 의용공학부

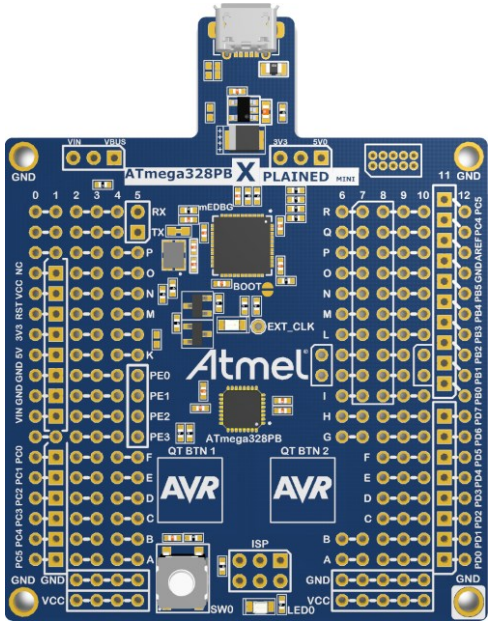
조 종만 교수

AVR Development Tools

AVR Development Tools

- Hardware Development Tool
  - Atmega328PB Xplained Mini (Xmini) Board
- Software Development Tool
  - Microchip Studio (Atmel Studio 7)





Atmega328PB Xplained Mini (Xmini) Board

ATmega328PB Instruction Set

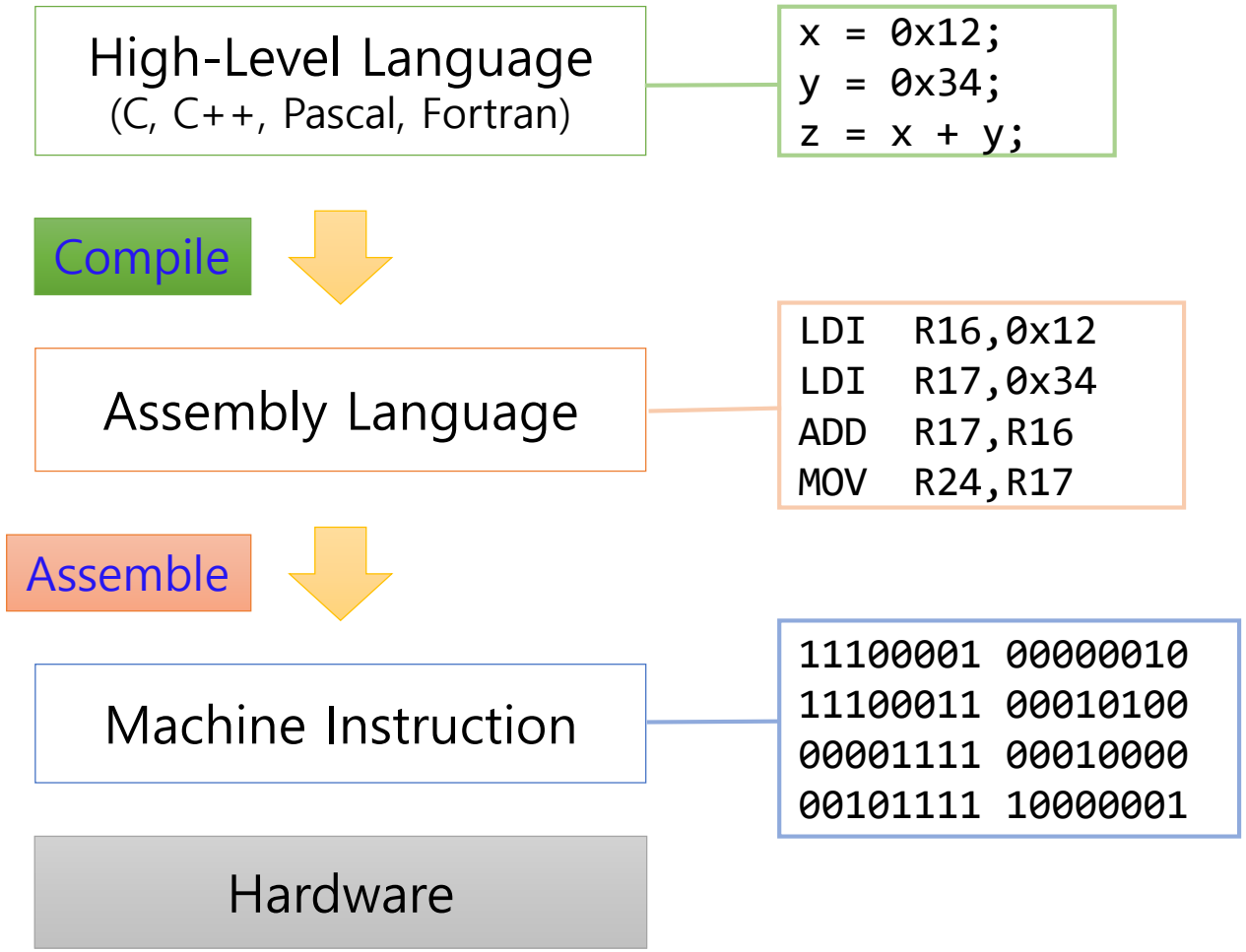
Objectives

- Understanding ATmega328PB Instruction Set
  - 각 명령어의 기능/동작
  - 명령어 실행에 따른 SREG(Status Register)의 변화
  - Opcode의 구성
  - 명령어 실행 시간
  - 데이터 메모리, 프로그램 메모리, 스택 메모리

수강에 필요한 자료

- Microchip Studio
- AVR Instruction Set Manual, DS40002198A, 2020
- AVR Assembler (AVRASM2), DS40001917A, 2017

Assembly Language



ATmega328PB Register File

Data Memory Address

R0	0x00
R1	0x01
R2	0x02
…	
R13	0x0D
R14	0x0E
R15	0x0F
R16	0x10
R17	0x11
…	
R26	0x1A
R27	0x1B
R28	0x1C
R29	0x1D
R30	0x1E
R31	0x1F

1 byte

ATmega328PB Registers are special storages with 8 bits capacity.
They are connected directly to the ALU → fast access time.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Assembly Language Statement Format

- An assembly language statement line may take one of the four following forms:
  - `[label:] instruction [operands] [Comment]`
  - Comment
  - Empty line
- A comment has the following form:

  `; [Text]`
- Items placed in braces are optional.
- The text between the comment-delimiter (`;`) and the end of line (EOL) is ignored by the Assembler.

이후에 사용될 표현(용어) 설명

- The assembler is not case sensitive.

- The operands have the following forms:

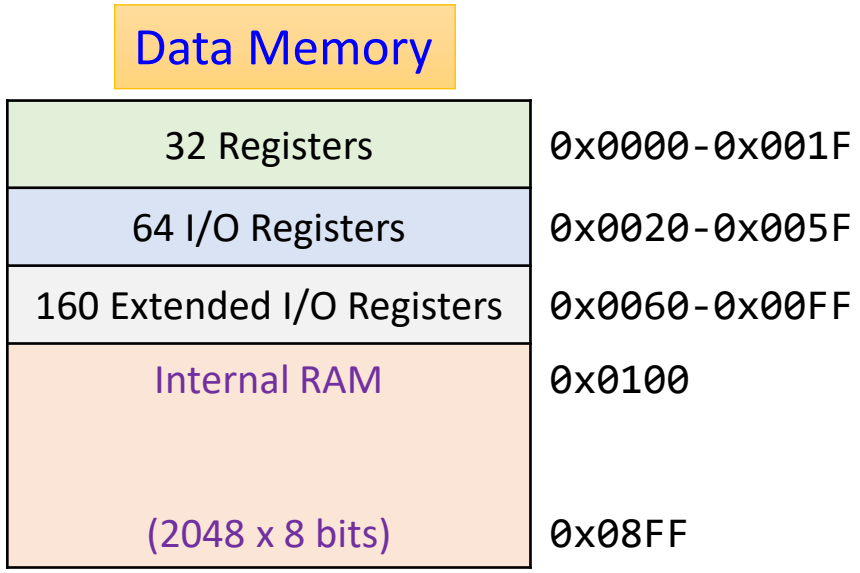
```
Rd: R0-R31 or R16-R31 (depending on instruction)
Rr: R0-R31
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction.
   Can be a constant expression.
q: Constant (0-63), can be a constant expression
```

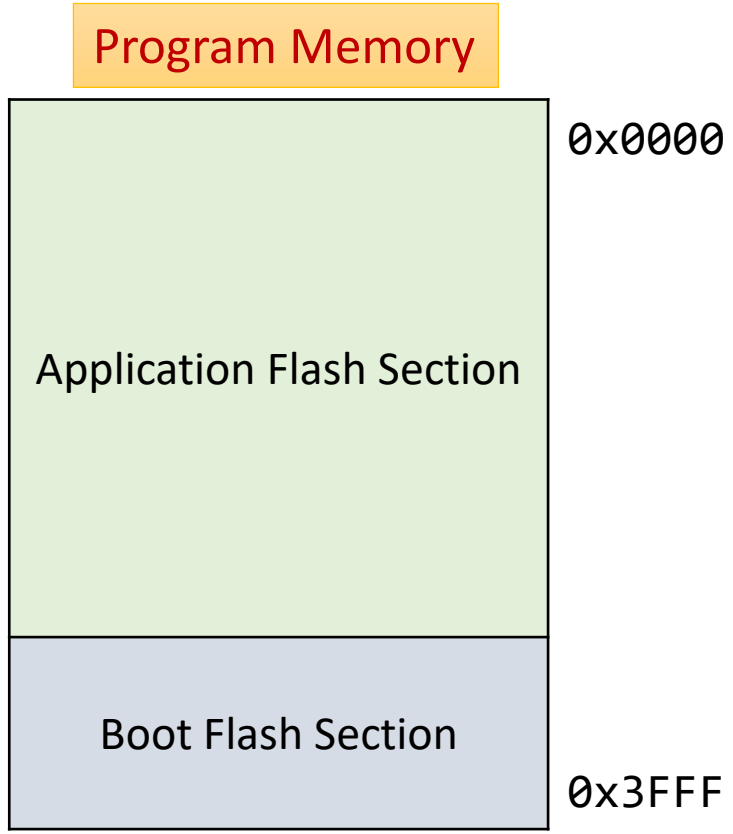
데이터 이동 명령어

(Data Transfer Instructions)

ATmega328PB Memories (1)

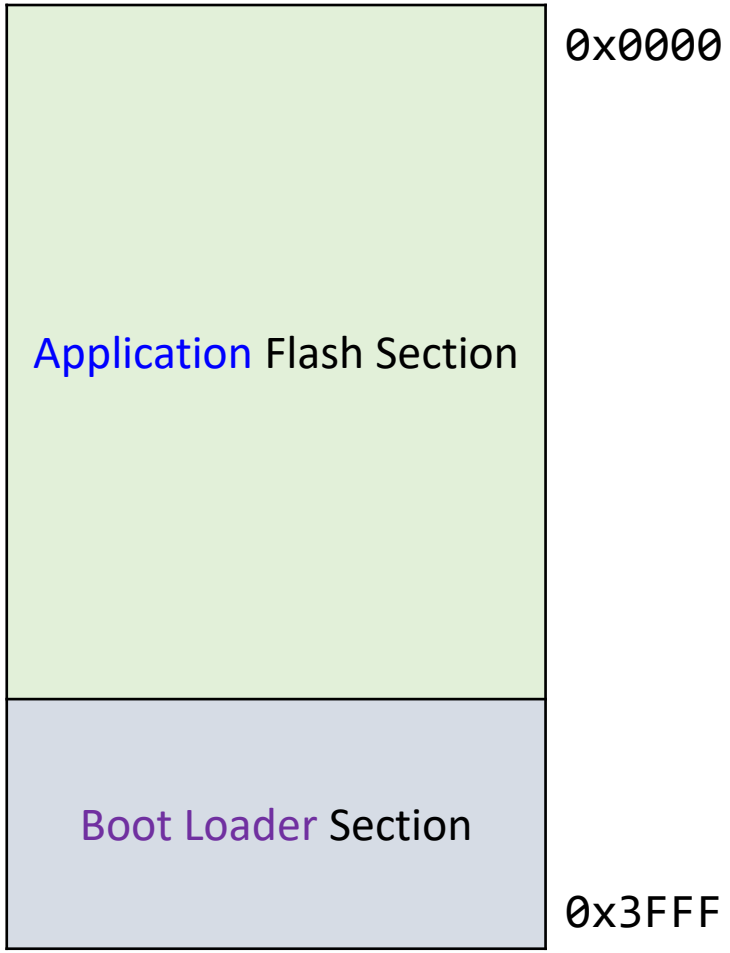
- Two memory spaces of ATmega328PB
  - Data memory
  - Program memory





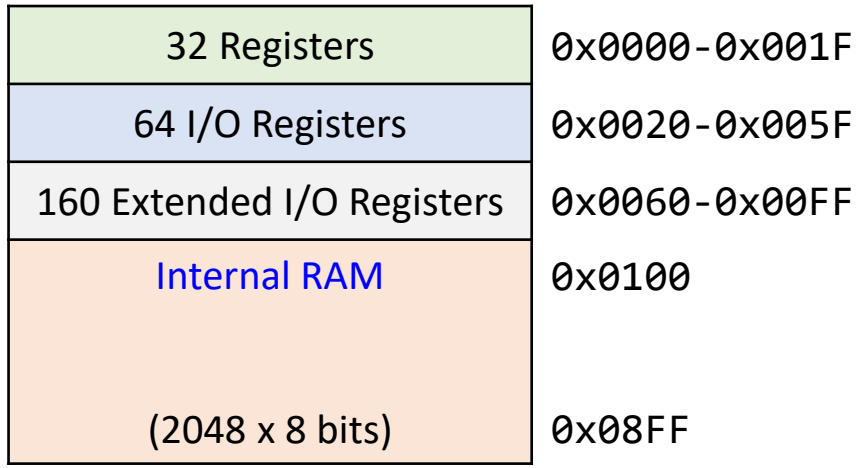
ATmega328PB Memories (2)

- In-System Reprogrammable Flash Program Memory
  - Boot Loader Section
  - Application Program Section



ATmega328PB Memories (3)

- SRAM Data Memory Space
  - Register File: 32
  - I/O Registers: 64
  - Extended I/O Registers: 160
  - Internal data SRAM: 2048



데이터 이동 명령어 (1)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	3
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	3
ST	X, Rr	Store Indirect	(X) ← Rr	None	2

데이터 이동 명령어 (2)

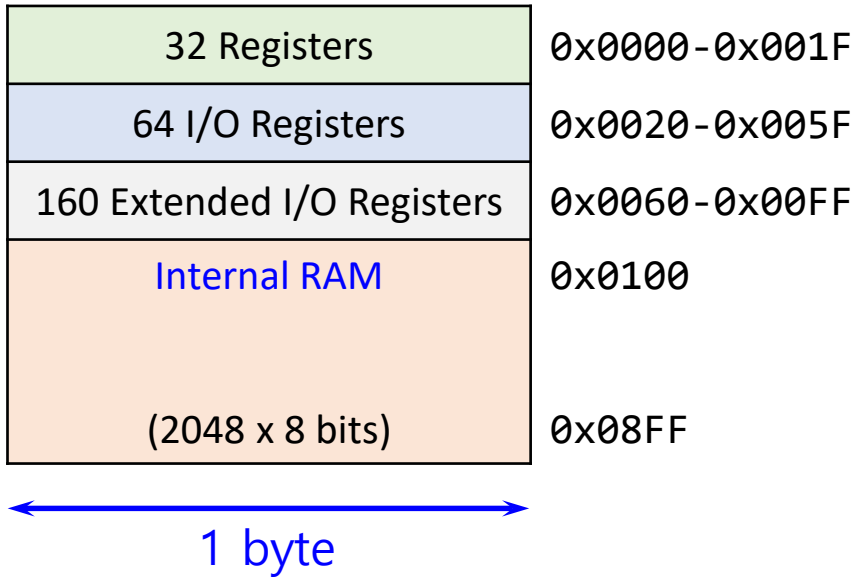
Mnemonics	Operands	Description	Operation	Flags	No. of Clock
DATA TRANSFER INSTRUCTIONS					
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

ATmega328PB Data Memory Access

ATmega328PB의 데이터 메모리 영역: 0x0000 - 0x08FF

0000 0000 0000 0000 - 0000 1000 1111 1111

따라서 데이터 메모리를 access하기 위해서는 16 bits의 번지 지정 레지스터가 필요하다.



데이터 메모리 access에 사용되는 16-bit 번지 지정 레지스터

- SP(Stack Pointer)
- X-Register
- Y-Register
- Z-Register

ATmega328PB Register File

	Address	
R0	0x00	
R1	0x01	
R2	0x02	
…		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
…		
R26	0x1A	XL: X-register Low Byte
R27	0x1B	XH: X-register High Byte
R28	0x1C	YL: Y-register Low Byte
R29	0x1D	YH: Y-register High Byte
R30	0x1E	ZL: Z-register Low Byte
R31	0x1F	ZH: Z-register High Byte

1 byte

ATmega328PB Registers are special storages with 8 bits capacity.
They are connected directly to the CPU → fast access time.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Additional Function:
These registers are 16-bit address pointers for indirect addressing of the memory space.
XH, XL, YH, YL, ZH, ZL: for Data Memory
ZH, ZL: for Program Memory

데이터 메모리 쓰기/읽기 (STS/LDS Instruction) (1)

직접 번지 지정

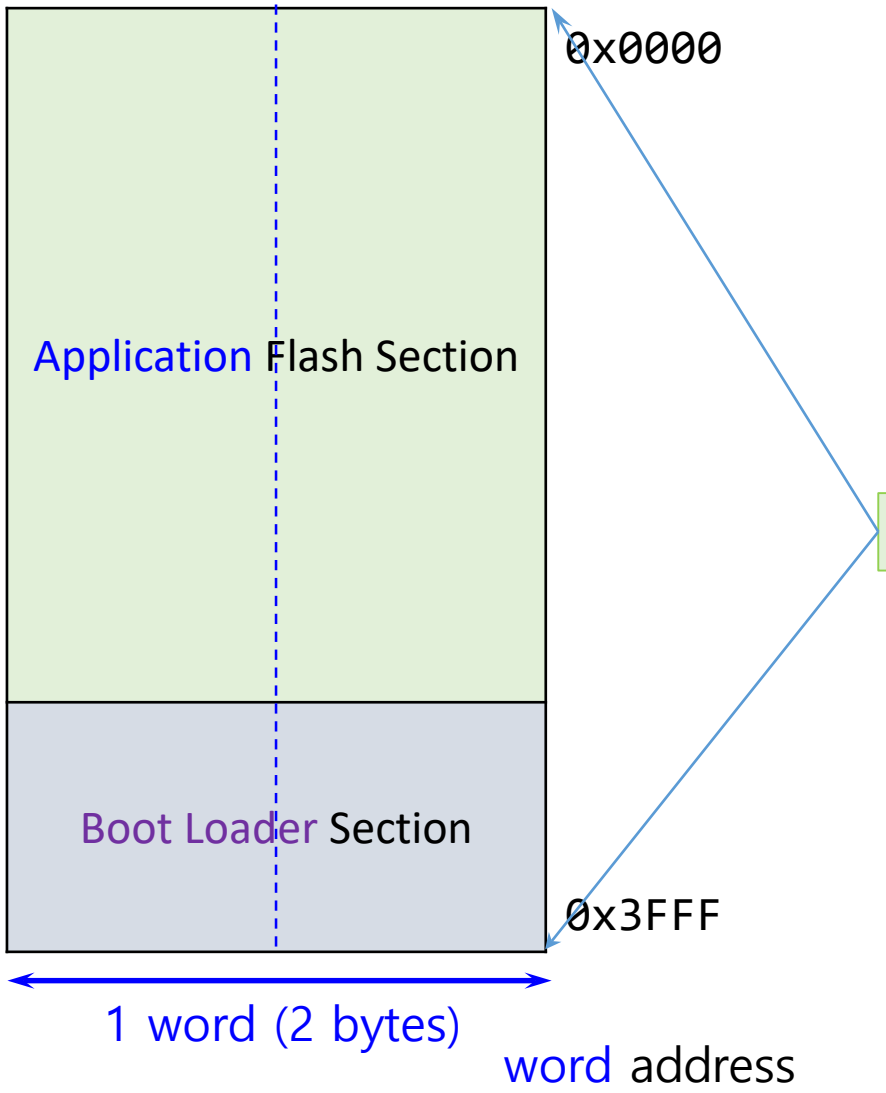
```
        ldi   r16,0x45  ; 상수 0x45를 R16 레지스터에 저장

        STS   0x102,R16 ; R16의 값(상수 0x45)을 데이터 메모리 0x0102번지에 저장

        LDS   R17,0x102 ; 데이터 메모리 0x0102번지의 내용을 R17 레지스터로 복사

here:   rjmp here
```

데이터 메모리 쓰기/읽기 (ST/LD Instruction) (2)

간접 번지 지정

```
; 데이터 메모리 0x0102번지에 0x45를 저장
        ldi   r16,0x45
        ldi   XL,0x02    ; R26, low byte of address
        ldi   XH,0x01    ; R27, high byte of address
        ST    X,R16      ; R16의 값(상수 0x45)을 데이터 메모리 0x0102번지에 저장
        LD    R17,X      ; 데이터 메모리 0x0102번지의 내용을 R17 레지스터로 복사
here:   rjmp here
```

ATmega328PB Program Memory Access



ATmega328PB의 프로그램 메모리의 영역:

0000 0000 0000 0000 - 0011 1111 1111 1111

따라서 프로그램 메모리를 access하기 위해서는 16 bits의 번지 지정 레지스터가 필요하다.

PC(Program Counter), Z-Register

Program Counter(PC), Z-Register : 16 bits

Program Memory 읽기 (LPM Instruction)

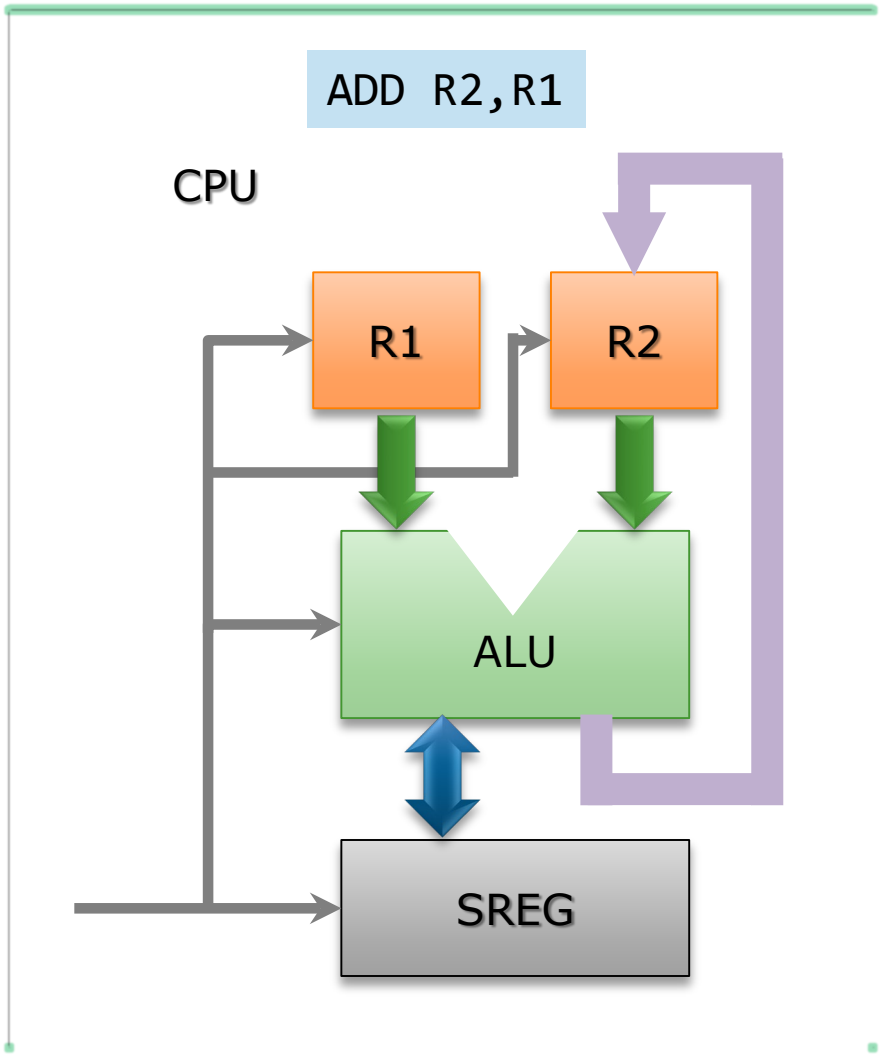
```
; Program Memory의 TABLE 번지에 있는 내용을 읽어서 R17에 저장
;
        ldi     ZL,LOW(TABLE << 1)    ; low byte of address
        ldi     ZH,HIGH(TABLE << 1)   ; high byte of address
        LPM     R17,Z
here:   rjmp    here
;
; Program Memory의 TABLE 번지에 네 바이트의 상수를 저장
TABLE:  .DB     0x41, 0x42, 0x43, 0x3A
```

List File (.lss) 읽는 방법

산술 논리 연산 명령어

(Arithmetic and Logic Instructions)

산술 논리 연산 명령어 (Arithmetic and Logic Instructions)



연산 결과에 따라 SREG가 영향을 받음

산술 논리 연산 명령어 (1)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V	2
AND	Rd, Rr	Logical AND	Rd ← Rd ∧ Rr	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd ∧ K	Z,N,V	1
OR	Rd, Rr	Logical OR	Rd ← Rd ∨ Rr	Z,N,V	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd ∨ K	Z,N,V	1

산술 논리 연산 명령어 (2)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
ARITHMETIC AND LOGIC INSTRUCTIONS					
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd ∨ K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd ∧ ($FFh - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd ∧ Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd,Rr	Multiply Unsigned	R1, R0 ← Rd × Rr	C	2

산술 연산 명령어

(Arithmetic Instructions)

수의 표현

unsigned 8-bit: 0부터 +255까지의 양수만을 표현
signed 8-bit: -128부터 +127까지의 수를 2의 보수 형태로 표현

8-비트의 2진수 표현								unsigned	signed 2's complement
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	+1	+1
0	1	1	1	1	1	1	1	+127	+127
1	0	0	0	0	0	0	0	+128	-128
1	1	1	1	1	1	1	1	+255	-1

ATmega328PB의 산술연산에서는 모든 8비트 수치는 signed 2's complement 형식으로 취급

AVR Status Register (SREG)

Bit No.	7	6	5	4	3	2	1	0
Name	I	T	H	S	V	N	Z	C
Reset	0	0	0	0	0	0	0	0

- Bit 7 – I: Global Interrupt Enable
- Bit 6 – T: Copy Storage
- Bit 5 – H: Half Carry Flag. Set if carry from bit 3 to bit 4 occurred.
- Bit 4 – S: Sign Flag. $S = N \oplus V$

  Can be used to determine if the result of the previous operation is positive or negative.
- Bit 3 – V: Two's Complement Overflow Flag.
- Bit 2 – N: Negative Flag. Same as bit 7 of the previous operation.
- Bit 1 – Z: Zero Flag. Set if the result of the previous operation is zero.
- Bit 0 – C: Carry Flag. Set if carry from bit 7 occurred.

Addition and Status Register (1)

8-비트 상수의 덧셈과 SREG

```
; 0x56+0x78=? (86+120=?)
        LDI     R16,0x56
        LDI     R17,0x78
        ADD     R16,R17
HERE:   RJMP    HERE
```

Half Carry

carry	0		1	1	1	0		0	0	0
0x56			0	1	0	1		0	1	1	0
+ 0x78	+		0	1	1	1		1	0	0	0
0xCE		0		1	1	0	0		1	1	1	0

Carry Negative Flag

- H (0): Half Carry Flag. Carry from bit 3 to bit 4 was 0.
- S (0): Sign Flag. $S = N \oplus V = 1 \oplus 1 = 0$
- V (1): Two's Complement Overflow Flag.

  양수 86과 양수 120의 합 206이 양수의 최대 범위(+127)를 벗어나 음수 -50이 되었으므로 overflow가 발생함.
- N (1): Negative Flag. Same as bit 7 of the previous operation.
- Z (0): Zero Flag. The result of the previous operation is not zero.
- C (0): Carry Flag. No Carry from bit 7.

Addition and Status Register (2)

8-비트 상수의 덧셈과 SREG

```
; 0x9A+0x78=? (-102+120=? )
        LDI     R16,0x9A
        LDI     R17,0x78
        ADD     R16,R17
HERE:   RJMP    HERE
```

Half Carry

carry		1		1	1	1	1	0	0	0
	0x9A			1	0	0	1	1	0	1	0
+	0x78	+		0	1	1	1	1	0	0	0
1	0x12		1	0	0	0	1	0	0	1	0
			Carry	Negative Flag							

- H (1): Half Carry Flag. Carry from bit 3 to bit 4 occurred.
- S (0): Sign Flag. $S = N \oplus V = 0 \oplus 0 = 0$
- V (0): Two's Complement Overflow Flag.

  음수 -102와 양수 120의 합이 양수 18이 되어 양수의 범위(+127)를 벗어나지 않아 overflow가 발생하지 않음.
- N (0): Negative Flag. Same as bit 7 of the previous operation.
- Z (0): Zero Flag. The result of the previous operation is not zero.
- C (1): Carry Flag. No Carry from bit 7.

SREG의 S(Sign) 비트에 관하여

- Sign 비트는 직전의 연산 결과가 양수인지 음수인지를 나타내는 비트임.
- 이 비트는 $S = N \oplus V$ 식으로 결정되고, 원래의 연산 결과가 양수(S=0)인지 음수(S=1)인지를 나타냄.

- N=0, V=0: 8-비트의 연산 결과가 양수이고 Overflow가 발생하지 않았으므로 연산 결과는 양수임. S=0
  (예1) 24 + 34 = 58 (예2) -102 + 120 = 18

- N=0, V=1: 8-비트의 연산 결과는 양수이지만 Overflow가 발생하였으므로 만일 16-비트 연산을 하였다면 결과는 음수임. S=1
  (예) 0xAA + 0x88 = 0x132. 그러나, 하위 8 비트만 취한 8 비트 연산 결과는 0x32 (10진수 +50).
  즉, (-86) + (-120) = -206이 되어야 하나 이 값은 부호를 고려한 8-비트 음수의 최소값인 -128을 초과하기 때문에 연산 결과는 +50으로 처리됨.
  8 비트 연산 결과는 양수이나 원래의 연산 결과는 음수라는 것을 의미하기 위해 Sign(S) 비트는 1로 됨.

- N=1, V=0: 8-비트의 연산 결과가 음수이고 Overflow가 발생하지 않았으므로 연산 결과는 음수임. S=1
  (예) 24 + (-34) = -10

- N=1, V=1: 8-비트의 연산 결과는 음수이지만 Overflow가 발생하였으므로 만일 16-비트 연산을 하였다면 결과는 양수임. S=0
  (예) 0x58 + 0x78 = 0xD0. 이 식을 10진수로 나타내면 86 + 120 = +206.
  그러나, 이 값은 부호를 고려한 8-비트 양수의 최대값인 +127을 초과하여 -58로 취급됨.
  8 비트 연산 결과는 음수이나 원래의 연산 결과는 양수라는 것을 의미하기 위해 Sign(S) 비트는 0으로 됨.

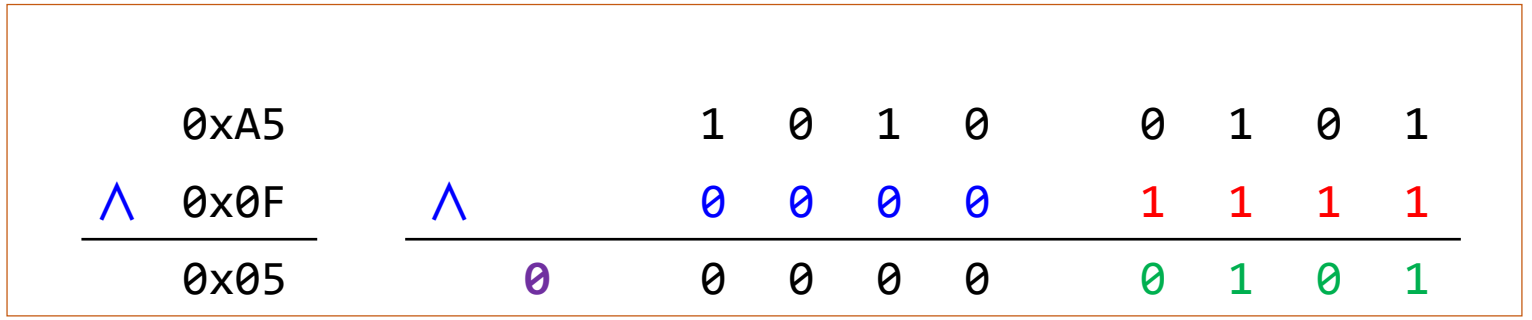
논리 연산 명령어

(Logical Instructions)

Logical Operation and Status Register

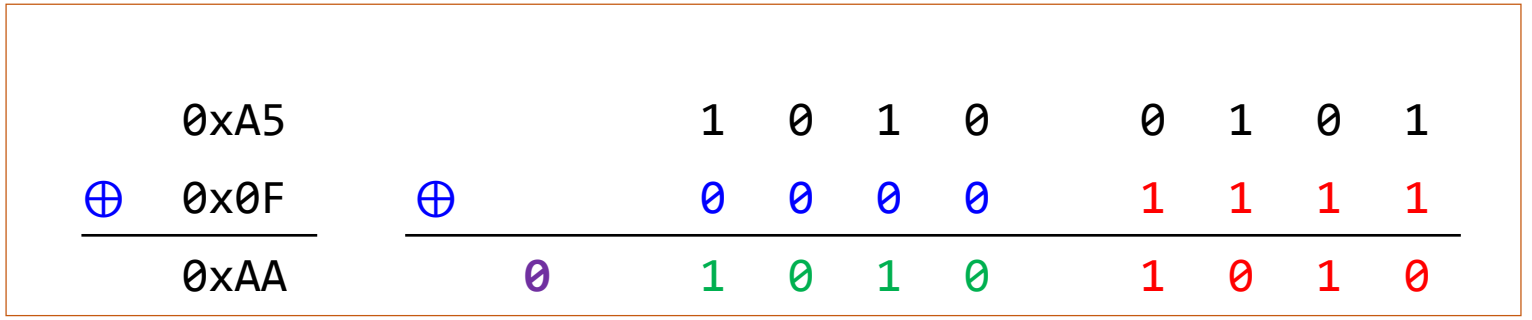
Logical AND (Bitwise AND)

```
; 0xA5 ∧ 0x0F = ?
        LDI     R16,0xA5
        ANDI    R16,0x0F

HERE:   RJMP    HERE
```



Logical XOR (Bitwise XOR)

```
; 0xA5 ⊕ 0x0F = ?
        LDI     R16,0xA5
        LDI     R17,0x0F
        EOR     R16,R17
HERE:   RJMP    HERE
```



ATmega328PB Opcode

Immediate Operand

ANDI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255

0111	KKKK	dddd	KKKK

ANDI R16,0x0F d=16-16=0=0000, K=0x0F=00001111 opcode=0x700F

0111	0000	0000	1111

Assemble 후 생성된 list 파일(.lss) 참조

분기 명령어

(Branch Instructions)

ATmega328PB Instruction Set (4)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Jump	PC ← k	None	3
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Call Subroutine	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if(I/O(P,b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register Set	if(I/O(P,b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC ← PC + k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC ← PC + k + 1	None	1 / 2

ATmega328PB Instruction Set (5)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BRANCH INSTRUCTIONS					
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC+ k + 1	None	1 / 2
BRLT	k	Branch if Less Than, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2

조건부 분기 명령어를 사용한 반복문 예제 (1)

1부터 10까지의 합 구하기

```
; c-language
unsigned char  sum=0;
unsigned char  n;

for (n=10; n>0; n--)
    sum += n;

while (1)
    ;
```

LDI, RJMP, ADC, DEC, BRNE

```
; assembly language
        sub     r17,r17   ; sum=0, C=0
        ldi     r16,10    ; n=10

loop:   adc     r17,r16   ; sum = sum + n
        dec     r16       ; n--;
        brne    loop      ; jump to loop if n != 0

here:   rjmp    here      ; wait here
```

조건부 분기 명령어를 사용한 반복문 예제 (2)

F_CPU가 1 MHz일 때 약 1 msec의 시간 지연

```
CPU의 클럭 주파수, F_CPU = 1,000,000 Hz

CPU의 클럭 주기, T_CPU = 1 / F_CPU = 1 us

1 ms (1,000 usec) 시간 지연에 필요한 클럭 수
    N_CLK = 1000 usec / 1 us = 1000 (clk)

loop 한 번을 실행하는데 소요되는 클럭 수
    LOOP_CLK = 2 + 2 = 4 (clk)

1 ms의 시간 지연에 필요한 loop 실행 회수
    N_LOOP = 1000 clk / 4 clk
           = 250 (회)
```

LDI, SBIW, RJMP, BRNE

```
start:  ldi     r24,LOW(250)    ; 1
        ldi     r25,HIGH(250)   ; 1

loop:   sbiw    r24,1           ; 2
        brne    loop            ; 2/1  (2+2)*250 = 1,000 clk
                                ; 1,000 clk * 1 us = 1 ms
here:   rjmp    here
```

- LOW and HIGH functions

```
HIGH(0x1234) == 0x12     ; HIGH byte of the expression
LOW(0x1234) == 0x34      ; LOW byte of the expression
```

- SBIW instruction

```
SBIW R24,k      R25:R24 ← R25:R24 – k
```

비트 조작 명령어

(Bit and Bit Test Instructions)

비트 조작 명령어 (1)

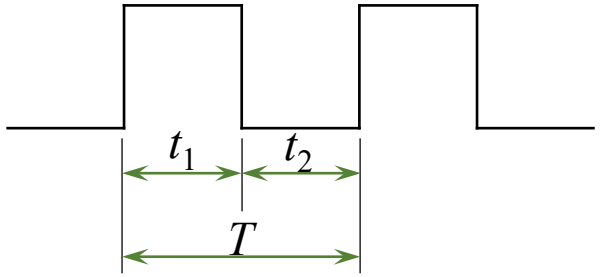
Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BIT AND BIT-TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0, C ← Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0, C ← Rd(0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register	I/O(P, b) ← 1	None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1

비트 조작 명령어 (2)

Mnemonics	Operands	Description	Operation	Flags	No. of Clock
BIT AND BIT-TEST INSTRUCTIONS					
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Two's Complement Overflow	V ← 1	V	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

비트 조작 명령어 예제

예제: PB5 핀에 그림과 같은 펄스 신호를 출력



I/O Register Control using SBI, CBI Instructions

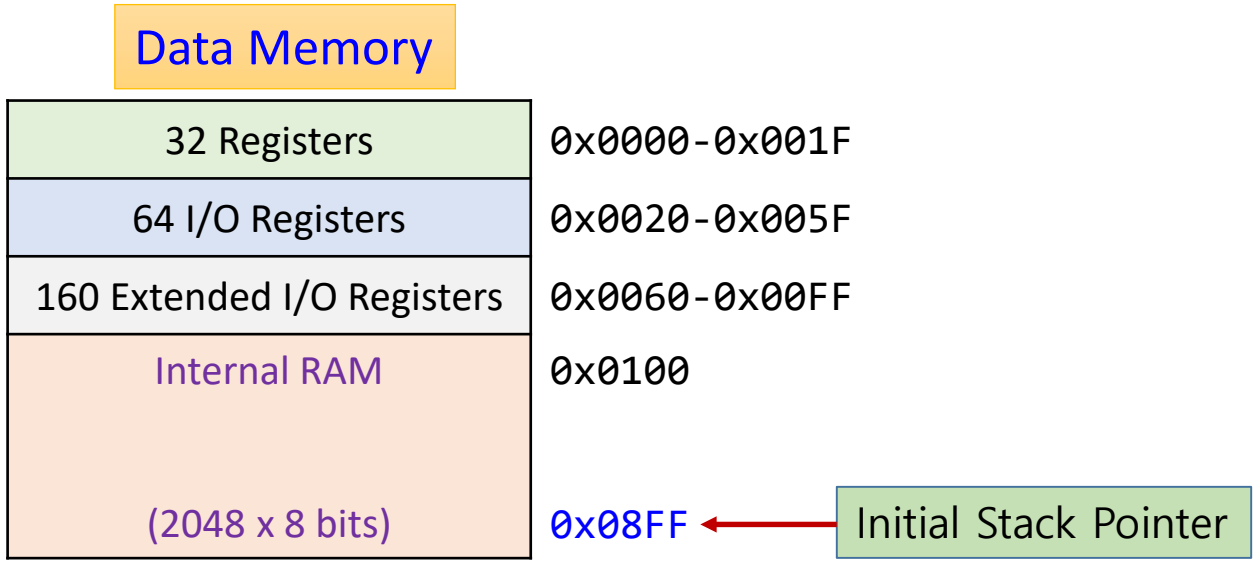
```
        SBI     DDRB,5      ; DDRB의 비트5를 1로 세트
LOOP:   SBI     PORTB,5     ; PORTB의 비트5를 1로 세트
        NOP
        CBI     PORTB,5     ; PORTB의 비트5를 0으로 클리어
        NOP
        RJMP    LOOP        ; 위의 동작을 무한 반복
```

이 예제에서 F_CPU가 1 MHz일 때, t_1, t_2 및 T의 값은 얼마인가?

(명령어 실행에 소요되는 Cycle 수 확인 방법)

Stack Memory and Stack Pointer for AVR

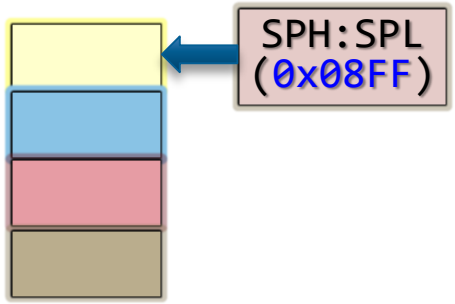
- Stack Pointer, SPL and SPH (0x3D and 0x3E)
  - A stack is a last-in first-out data structure
  - AVR stack is implemented as growing from higher to lower memory locations
  - 16-bit stack pointer(SPH:SPL) points to the top of stack (0x08FF)



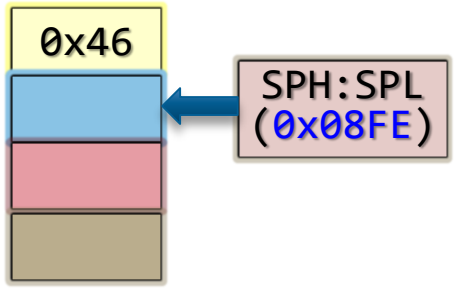
Stack Pointer 초기화 및 PUSH/POP Instruction

```
1.   LDI   R16,0x08
2.   OUT   SPH,R16
3.   LDI   R16,0xFF
4.   OUT   SPL,R16

5.   LDI   R16,0x46
6.   LDI   R17,0x47

7.   PUSH  R16
8.   PUSH  R17
     ...
; R16, R17 사용
; R16, R17 내용변경
       ...
9.   POP   R17
10.  POP   R16
```

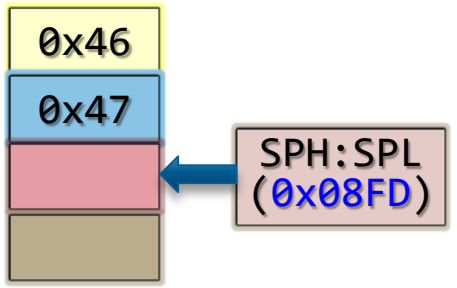
ATmega328PB RAM ADDRESS: 0x0100~0x08FF



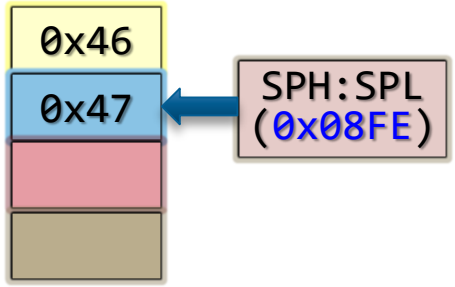
After Line #4



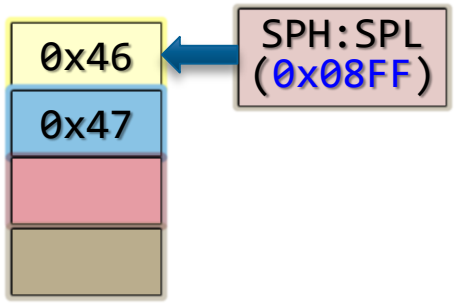
After Line #7



After Line #8



After Line #9



After Line #10

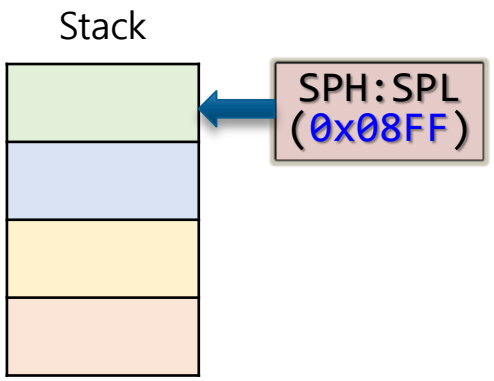
Subroutine 호출과 Stack Pointer (RCALL/RET Instruction)

```
; SP가 0x08FF로 초기화되어 있다고 가정
        ldi     r16,0x12
        rcall   mul2
here:   rjmp    here
;
mul2:   add     r16,r16
        ret
```

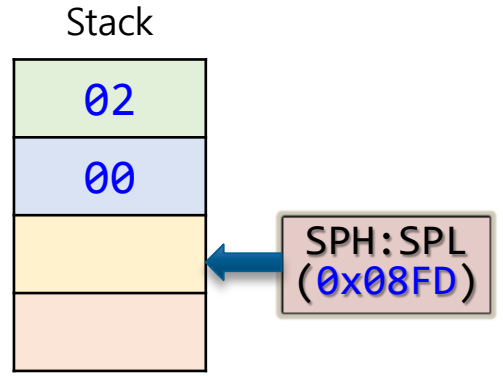
어셈블된 리스트 파일

```
1:  0000 e102           ldi     r16,0x12
2:  0001 d001           rcall   mul2
3:  0002 cfff   here:   rjmp    here
                ;
4:  0003 0f00   mul2:   add     r16,r16
5:  0004 9508           ret
```

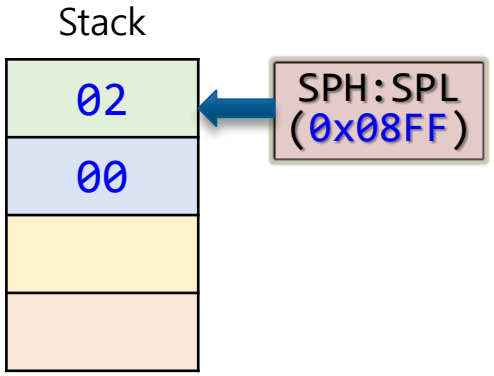
Program memory address



초기 상태



2번행 실행 직후/4번행 실행 직전



5번행 실행 직후/3번행 실행 직전

Summary

- ATmega328PB Instruction Set
  - 각 명령어의 기능/동작
  - 명령어 실행에 따른 SREG(Status Register)의 변화
  - Opcode의 구성
  - 명령어 실행 시간
  - 데이터 메모리, 프로그램 메모리, 스택 메모리
- Assembly Language의 장점?

What's next?


THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.