

AVR Binary Reversing

mongii@grayhash

개요

- AVR의 소스코드가 없을 때, 바이너리를 리버싱 하는 방법 설명
- AVR 어셈블리어 학습 방법 설명
- AVR 바이너리의 구조 이해

AVR용 gdb의 위치

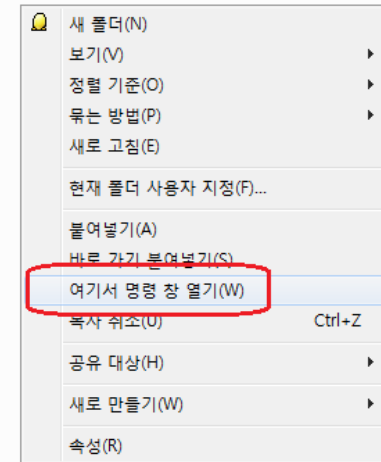
구성 라이브러리에 포함 공유 대상 새 폴더

이름	수정한 날짜	유형	크기
avr-addr2line.exe	2014-04-30 오후...	응용 프로그램	435KB
avr-ar.exe	2014-04-30 오후...	응용 프로그램	454KB
avr-as.exe	2014-04-30 오후...	응용 프로그램	600KB
avr-c++.exe	2014-04-30 오후...	응용 프로그램	587KB
avr-c++filt.exe	2014-04-30 오후...	응용 프로그램	434KB
avr-cpp.exe	2014-04-30 오후...	응용 프로그램	586KB
avr-elfedit.exe	2014-04-30 오후...	응용 프로그램	55KB
avr-g++.exe	2014-04-30 오후...	응용 프로그램	587KB
avr-gcc.exe	2014-04-30 오후...	응용 프로그램	585KB
avr-gcc-4.8.1.exe	2014-04-30 오후...	응용 프로그램	585KB
avr-gcc-ar.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcc-nm.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcc-ranlib.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcov.exe	2014-04-30 오후...	응용 프로그램	231KB
avr-gdb.exe	2014-04-30 오후...	응용 프로그램	3,695KB
avr-gprof.exe	2014-04-30 오후...	응용 프로그램	487KB
avr-ld.bfd.exe	2014-04-30 오후...	응용 프로그램	864KB
avr-ld.exe	2014-04-30 오후...	응용 프로그램	864KB
avr-man	2014-04-30 오후...	파일	2KB
avr-nm.exe	2014-04-30 오후...	응용 프로그램	443KB
avr-objcopy.exe	2014-04-30 오후...	응용 프로그램	564KB
avr-objdump.exe	2014-04-30 오후...	응용 프로그램	654KB
avr-ranlib.exe	2014-04-30 오후...	응용 프로그램	454KB
avr-readelf.exe	2014-04-30 오후...	응용 프로그램	340KB
avr-size.exe	2014-04-30 오후...	응용 프로그램	444KB
avr-strings.exe	2014-04-30 오후...	응용 프로그램	436KB
avr-strip.exe	2014-04-30 오후...	응용 프로그램	564KB
libtermcap-0.dll	2014-04-30 오후...	응용 프로그램 확장	9KB
mingwm10.dll	2014-04-30 오후...	응용 프로그램 확장	9KB

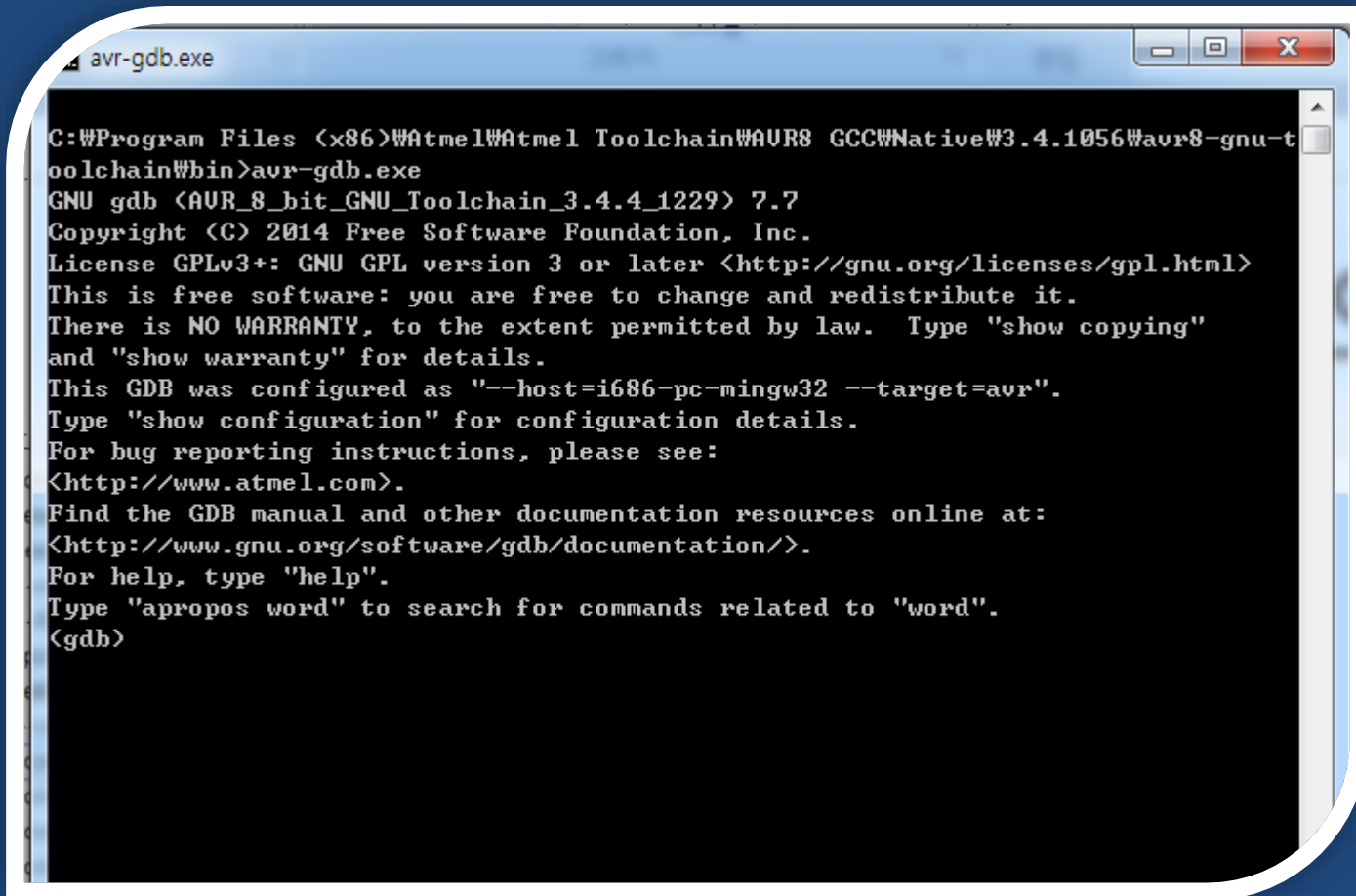
cmd 실행

avr-addr2line.exe	2014-04-30 오후...	응용 프로그램	435KB
avr-ar.exe	2014-04-30 오후...	응용 프로그램	454KB
avr-as.exe	2014-04-30 오후...	응용 프로그램	600KB
avr-c++.exe	2014-04-30 오후...	응용 프로그램	587KB
avr-c++filt.exe	2014-04-30 오후...	응용 프로그램	434KB
avr-cpp.exe	2014-04-30 오후...	응용 프로그램	586KB
avr-elfedit.exe	2014-04-30 오후...	응용 프로그램	55KB
avr-g++.exe	2014-04-30 오후...	응용 프로그램	587KB
avr-gcc.exe	2014-04-30 오후...	응용 프로그램	585KB
avr-gcc-4.8.1.exe	2014-04-30 오후...	응용 프로그램	585KB
avr-gcc-ar.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcc-nm.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcc-ranlib.exe	2014-04-30 오후...	응용 프로그램	50KB
avr-gcov.exe	2014-04-30 오후...	응용 프로그램	231KB
avr-gdb.exe	2014-04-30 오후...	응용 프로그램	3,695KB
avr-gprof.exe	2014-04-30 오후...	응용 프로그램	487KB
avr-ld.bfd.exe	2014-04-30 오후...	응용 프로그램	864KB
avr-ld.exe	2014-04-30 오후...	응용 프로그램	864KB
avr-man	2014-04-30 오후...	파일	2KB
avr-nm.exe	2014-04-30 오후...	응용 프로그램	443KB
avr-objcopy.exe	2014-04-30 오후...	응용 프로그램	564KB
avr-objdump.exe	2014-04-30 오후...	응용 프로그램	654KB
avr-ranlib.exe	2014-04-30 오후...	응용 프로그램	454KB
avr-readelf.exe	2014-04-30 오후...	응용 프로그램	340KB
avr-size.exe	2014-04-30 오후...	응용 프로그램	444KB
avr-strings.exe	2014-04-30 오후...	응용 프로그램	436KB
avr-strip.exe	2014-04-30 오후...	응용 프로그램	564KB
LED_BLANK.elf	2015-02-21 오후...	ELF 파일	5KB
libtermcap-0.dll	2014-04-30 오후...	응용 프로그램 확장	9KB
mingwm10.dll	2014-04-30 오후...	응용 프로그램 확장	9KB

Shift+마우스 우클릭



gdb 실행



```
C:\Program Files (x86)\Atmel\Atmel Toolchain\AVR8 GCC\Native\3.4.1056\avr8-gnu-toolchain\bin>avr-gdb.exe
GNU gdb (AVR_8_bit_GNU_Toolchain_3.4.4_1229) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-mingw32 --target=avr".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.atmel.com>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

디버깅 대상 코드

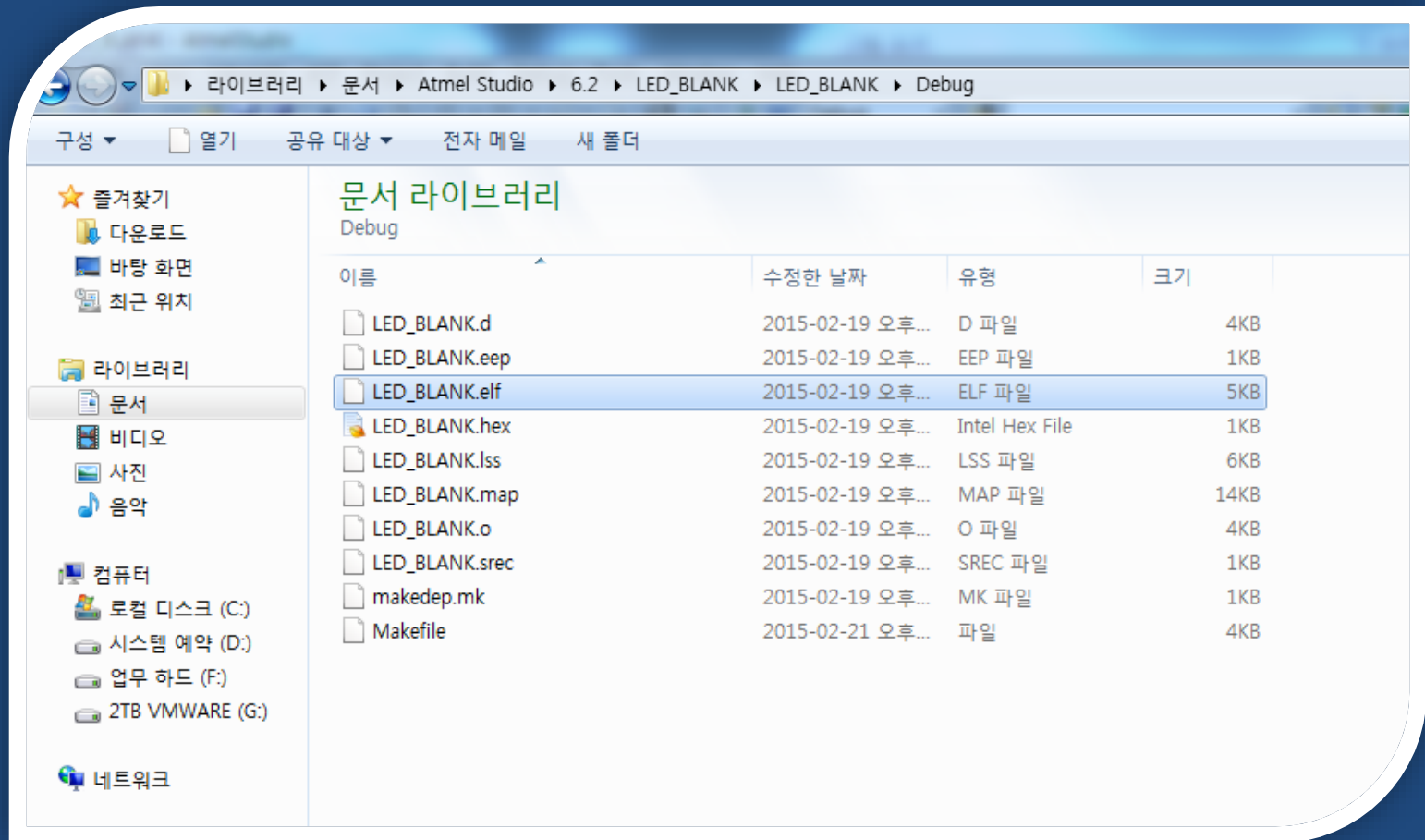
```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRA = 0xff;

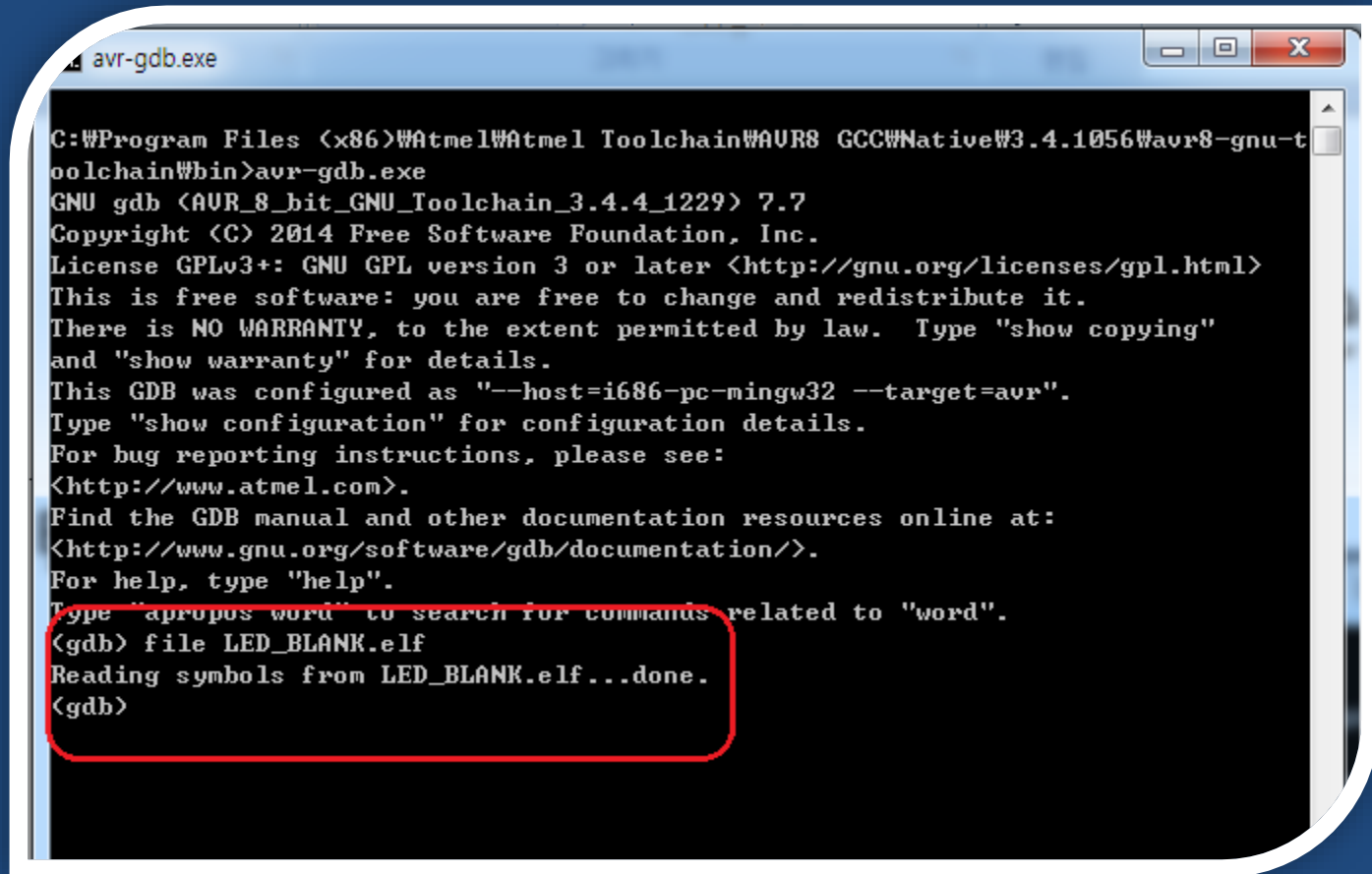
    while(1)
    {
        PORTA = 0xff;
        _delay_ms(1000);
        PORTA = 0x00;
        _delay_ms(1000);
    }
}
```

컴파일 된 바이너리의 위치

- gdb가 있는 폴더로 복사



대상 바이너리 Open



```
C:\Program Files (x86)\Atmel\Atmel Toolchain\AVR8 GCC\Native\3.4.1056\avr8-gnu-toolchain\bin>avr-gdb.exe
GNU gdb (AVR_8_bit_GNU_Toolchain_3.4.4_1229) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-mingw32 --target=avr".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.atmel.com>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file LED_BLANK.elf
Reading symbols from LED_BLANK.elf...done.
(gdb)
```


주요 명령어

- file : 디버깅 대상 파일 열기
- **disassemble** : 디스어셈블링
- x/10i [주소] : 10개의 명령어 출력

- run : 실행
- break : 브레이크 포인트 설정
- continue : 실행 재개

disass main

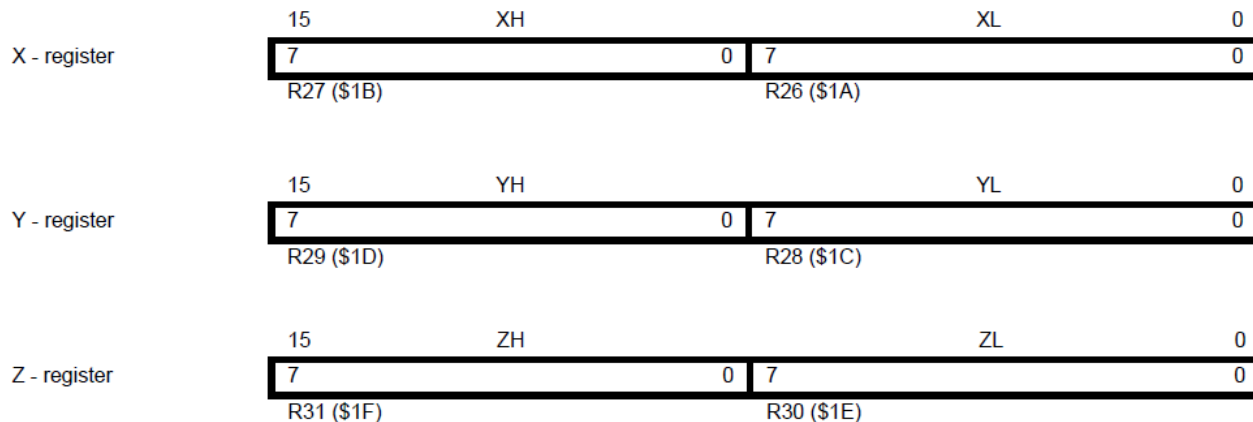
```
...ng symbols from LED_BLANK.elf...done.
(gdb) disass main
Dump of assembler code for function main:
0x0000009e <+0>:      ldi    r24, 0xFF      ; 255
0x000000a0 <+2>:      out    0x1a, r24      ; 26
0x000000a2 <+4>:      out    0x1b, r24      ; 27
0x000000a4 <+6>:      ldi    r18, 0xFF      ; 255
0x000000a6 <+8>:      ldi    r19, 0x69      ; 105
0x000000a8 <+10>:     ldi    r25, 0x18      ; 24
0x000000aa <+12>:     subi   r18, 0x01      ; 1
0x000000ac <+14>:     sbci   r19, 0x00      ; 0
0x000000ae <+16>:     sbci   r25, 0x00      ; 0
0x000000b0 <+18>:     brne   .-8            ; 0xaa <main+12>
0x000000b2 <+20>:     rjmp   .+0            ; 0xb4 <main+22>
0x000000b4 <+22>:     nop
0x000000b6 <+24>:     out    0x1b, r1       ; 27
0x000000b8 <+26>:     ldi    r18, 0xFF      ; 255
0x000000ba <+28>:     ldi    r19, 0x69      ; 105
0x000000bc <+30>:     ldi    r25, 0x18      ; 24
0x000000be <+32>:     subi   r18, 0x01      ; 1
0x000000c0 <+34>:     sbci   r19, 0x00      ; 0
0x000000c2 <+36>:     sbci   r25, 0x00      ; 0
0x000000c4 <+38>:     brne   .-8            ; 0xbe <main+32>
0x000000c6 <+40>:     rjmp   .+0            ; 0xc8 <main+42>
0x000000c8 <+42>:     nop
0x000000ca <+44>:     rjmp   .-42           ; 0xa2 <main+4>

End of assembler dump.
(gdb) _
```

AVR의 레지스터

- 32개의 8비트 범용 레지스터
 - R0~R31
- X, Y, Z 레지스터
 - 사실상 R26~R31 레지스터
 - 16비트 크기로 사용 가능 ex> Y 레지스터 == 스택 포인터

Figure 6-3. The X-, Y-, and Z-registers



disass main

```
(gdb) disass main
```

```
Dump of assembler code for function main:
```

```
0x0000009e <+0>:   ldi    r24, 0xFF      ; 255
0x000000a0 <+2>:   out   0x1a, r24      ; 26
0x000000a2 <+4>:   out   0x1b, r24      ; 27
0x000000a4 <+6>:   ldi    r18, 0xFF      ; 255
0x000000a6 <+8>:   ldi    r19, 0x69      ; 105
0x000000a8 <+10>:  ldi    r25, 0x18      ; 24
0x000000aa <+12>:  subi   r18, 0x01      ; 1
0x000000ac <+14>:  sbci   r19, 0x00      ; 0
0x000000ae <+16>:  sbci   r25, 0x00      ; 0
0x000000b0 <+18>:  brne   .-8           ; 0xaa <main+12>
0x000000b2 <+20>:  rjmp   .+0           ; 0xb4 <main+22>
0x000000b4 <+22>:  nop
0x000000b6 <+24>:  out   0x1b, r1       ; 27
0x000000b8 <+26>:  ldi    r18, 0xFF      ; 255
0x000000ba <+28>:  ldi    r19, 0x69      ; 105
0x000000bc <+30>:  ldi    r25, 0x18      ; 24
0x000000be <+32>:  subi   r18, 0x01      ; 1
0x000000c0 <+34>:  sbci   r19, 0x00      ; 0
0x000000c2 <+36>:  sbci   r25, 0x00      ; 0
0x000000c4 <+38>:  brne   .-8           ; 0xbe <main+32>
0x000000c6 <+40>:  rjmp   .+0           ; 0xc8 <main+42>
0x000000c8 <+42>:  nop
0x000000ca <+44>:  rjmp   .-42          ; 0xa2 <main+4>
```

```
End of assembler dump.
```

```
(gdb)
```

Instruction set summary

- Datasheet 352p~355p

10. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \cdot Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \cdot K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \cdot (\$FF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \cdot Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow \$FF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
JMP	k	Direct Jump	$PC \leftarrow k$	None	3
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	None	3
CALL	k	Direct Subroutine Call	$PC \leftarrow k$	None	4
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip If Equal	if $(Rd = Rr) PC \leftarrow PC + 2$ or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	$Rd - Rr$	Z, N,V,C,H	

Main 함수 내의 명령어들

- LDI : Load Immediate value
- OUT : Out Port
- SUBI : Subtract Constant from Register
- SBCI : Subtract with Carry Constant from Reg
- BRNE : Branch if Not Equal
- RJMP : Relative Jump
- NOP : No Operation

Subtract with Carry Constant from Reg

- Carry가 발생하는 경우 1

0b11111111 + 0b00000001

=>

0b00000000 with carry flag on

- Carry가 발생하는 경우 2

0b00000000 - 0b00000001

=>

0b11111111 with carry flag on

Subtract with Carry Constant from Reg

- `SBCI r0, 10`
 - `r0 = 20 / carry flag = 1`
- Subtract with carry
 - `r0(20) - 10 - 1 => 9`

disass main

(gdb) disass main

Dump of assembler code for function main:

```
0x0000009e <+0>:   ldi    r24, 0xFF      ; r24에 0xFF를 저장한다.
0x000000a0 <+2>:   out   0x1a, r24      ; 0x1a 포트에 r24를 출력한다.
0x000000a2 <+4>:   out   0x1b, r24      ; 0x1b 포트에 r24를 출력한다.
0x000000a4 <+6>:   ldi    r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000a6 <+8>:   ldi    r19, 0x69      ; r19에 0x69를 저장한다.
0x000000a8 <+10>:  ldi    r25, 0x18      ; r25에 0x18를 저장한다.
0x000000aa <+12>:  subi   r18, 0x01      ; r18에서 1을 뺀다.
0x000000ac <+14>:  sbci   r19, 0x00      ; r19에서 0-carry를 뺀다.
0x000000ae <+16>:  sbci   r25, 0x00      ; r25에서 0-carry를 뺀다.
0x000000b0 <+18>:  brne   .-8           ; main+12로 branch
0x000000b2 <+20>:  rjmp   .+0           ; main+22로 jump
0x000000b4 <+22>:  nop
0x000000b6 <+24>:  out   0x1b, r1       ; 0x1b 포트에 r1을 출력한다.
0x000000b8 <+26>:  ldi    r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000ba <+28>:  ldi    r19, 0x69      ; r19에 0x69를 저장한다.
0x000000bc <+30>:  ldi    r25, 0x18      ; r25에 0x18를 저장한다.
0x000000be <+32>:  subi   r18, 0x01      ; r18에서 1을 뺀다.
0x000000c0 <+34>:  sbci   r19, 0x00      ; r19에서 0-carry를 뺀다.
0x000000c2 <+36>:  sbci   r25, 0x00      ; r25에서 0-carry를 뺀다.
0x000000c4 <+38>:  brne   .-8           ; main+32로 branch
0x000000c6 <+40>:  rjmp   .+0           ; main+42로 jump
0x000000c8 <+42>:  nop
0x000000ca <+44>:  rjmp   .-42          ; main+4로 jump
```

End of assembler dump.

(gdb)

disass main

(gdb) disass main

Dump of assembler code for function main:

```
0x0000009e <+0>: ldi r24, 0xFF ; r24에 0xFF를 저장한다.
0x000000a0 <+2>: out 0x1a, r24 ; 0x1a 포트에 r24를 출력한다.
0x000000a2 <+4>: out 0x1b, r24 ; 0x1b 포트에 r24를 출력한다.
0x000000a4 <+6>: ldi r18, 0xFF ; r18에 0xFF를 저장한다.
0x000000a6 <+8>: ldi r19, 0x69 ; r19에 0x69를 저장한다.
0x000000a8 <+10>: ldi r25, 0x18 ; r25에 0x18를 저장한다.
0x000000aa <+12>: subi r18, 0x01 ; r18에서 1을 뺀다.
0x000000ac <+14>: sbci r19, 0x00 ; r19에서 0-carry를 뺀다.
0x000000ae <+16>: sbci r25, 0x00 ; r25에서 0-carry를 뺀다.
0x000000b0 <+18>: brne .-8 ; main+12로 branch
0x000000b2 <+20>: rjmp .+0 ; main+22로 jump
0x000000b4 <+22>: nop
0x000000b6 <+24>: out 0x1b, r1 ; 0x1b 포트에 r1을 출력한다.
0x000000b8 <+26>: ldi r18, 0xFF ; r18에 0xFF를 저장한다.
0x000000ba <+28>: ldi r19, 0x69 ; r19에 0x69를 저장한다.
0x000000bc <+30>: ldi r25, 0x18 ; r25에 0x18를 저장한다.
0x000000be <+32>: subi r18, 0x01 ; r18에서 1을 뺀다.
0x000000c0 <+34>: sbci r19, 0x00 ; r19에서 0-carry를 뺀다.
0x000000c2 <+36>: sbci r25, 0x00 ; r25에서 0-carry를 뺀다.
0x000000c4 <+38>: brne .-8 ; main+32로 branch
0x000000c6 <+40>: rjmp .+0 ; main+42로 jump
0x000000c8 <+42>: nop
0x000000ca <+44>: rjmp .-42 ; main+4로 jump
```

DDRA=0xFF

while{}

End of assembler dump.

(gdb)

Register Summary

- Datasheet 349p~350p

\$24 (\$44)	TCNT2	Timer/Counter2 (8 Bit)								155
\$23 (\$43)	OCR2	Timer/Counter2 Output Compare Register								155
\$22 (\$42)	OCDR	IDRD/OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0	260
\$21 (\$41)	WDTCSR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	57
\$20 (\$40)	SFIOR	TSM	–	–	–	ACME	PUD	PSR0	PSR321	84, 108, 141, 218
\$1F (\$3F)	EEARH	–	–	–	–	EEPROM Address Register High				31
\$1E (\$3E)	EEARL	EEPROM Address Register Low Byte								31
\$1D (\$3D)	EEDR	EEPROM Data Register								31
\$1C (\$3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	31
\$1B (\$3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	84
\$1A (\$3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	84
\$19 (\$39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	84
\$18 (\$38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	85
\$17 (\$37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	85
\$16 (\$36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	85
\$15 (\$35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	85
\$14 (\$34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	85
\$13 (\$33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	85
\$12 (\$32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	85
\$11 (\$31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	86
\$10 (\$30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	86

disass main

(gdb) disass main

Dump of assembler code for function main:

```
0x0000009e <+0>:   ldi    r24, 0xFF      ; r24에 0xFF를 저장한다.
0x000000a0 <+2>:   out   0x1a, r24      ; 0x1a 포트에 r24를 출력한다.
0x000000a2 <+4>:   out   0x1b, r24      ; 0x1b 포트에 r24를 출력한다.
0x000000a4 <+6>:   ldi    r18, 0xFF     ; r18에 0xFF를 저장한다.
0x000000a6 <+8>:   ldi    r19, 0x69     ; r19에 0x69를 저장한다.
0x000000a8 <+10>:  ldi    r25, 0x18     ; r25에 0x18를 저장한다.
0x000000aa <+12>:  subi  r18, 0x01     ; r18에서 1을 뺀다.
0x000000ac <+14>:  sbci  r19, 0x00     ; r19에서 0-carry를 뺀다.
0x000000ae <+16>:  sbci  r25, 0x00     ; r25에서 0-carry를 뺀다.
0x000000b0 <+18>:  brne  .-8           ; main+12로 branch
0x000000b2 <+20>:  rjmp  .+0           ; main+22로 jump
0x000000b4 <+22>:  nop
0x000000b6 <+24>:  out   0x1b, r1      ; 0x1b 포트에 r1을 출력한다.
0x000000b8 <+26>:  ldi    r18, 0xFF     ; r18에 0xFF를 저장한다.
0x000000ba <+28>:  ldi    r19, 0x69     ; r19에 0x69를 저장한다.
0x000000bc <+30>:  ldi    r25, 0x18     ; r25에 0x18를 저장한다.
0x000000be <+32>:  subi  r18, 0x01     ; r18에서 1을 뺀다.
0x000000c0 <+34>:  sbci  r19, 0x00     ; r19에서 0-carry를 뺀다.
0x000000c2 <+36>:  sbci  r25, 0x00     ; r25에서 0-carry를 뺀다.
0x000000c4 <+38>:  brne  .-8           ; main+32로 branch
0x000000c6 <+40>:  rjmp  .+0           ; main+42로 jump
0x000000c8 <+42>:  nop
0x000000ca <+44>:  rjmp  .-42          ; main+4로 jump
```

<- PORTA = 0xFF;

<- PORTA = 0x00;

End of assembler dump.

(gdb)

disass main

(gdb) disass main

Dump of assembler code for function main:

```
0x0000009e <+0>:    ldi    r24, 0xFF      ; r24에 0xFF를 저장한다.
0x000000a0 <+2>:    out    0x1a, r24      ; 0x1a 포트에 r24를 출력한다.
0x000000a2 <+4>:    out    0x1b, r24      ; 0x1b 포트에 r24를 출력한다.
0x000000a4 <+6>:    ldi    r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000a6 <+8>:    ldi    r19, 0x69      ; r19에 0x69를 저장한다.
0x000000a8 <+10>:   ldi    r25, 0x18      ; r25에 0x18를 저장한다.
0x000000aa <+12>:   subi   r18, 0x01      ; r18에서 1을 뺀다.
0x000000ac <+14>:   sbci   r19, 0x00      ; r19에서 0-carry를 뺀다.
0x000000ae <+16>:   sbci   r25, 0x00      ; r25에서 0-carry를 뺀다.
0x000000b0 <+18>:   brne   .-8            ; main+12로 branch
0x000000b2 <+20>:   rjmp   .+0            ; main+22로 jump
0x000000b4 <+22>:   nop
0x000000b6 <+24>:   out    0x1b, r1       ; 0x1b 포트에 r1을 출력한다.
0x000000b8 <+26>:   ldi    r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000ba <+28>:   ldi    r19, 0x69      ; r19에 0x69를 저장한다.
0x000000bc <+30>:   ldi    r25, 0x18      ; r25에 0x18를 저장한다.
0x000000be <+32>:   subi   r18, 0x01      ; r18에서 1을 뺀다.
0x000000c0 <+34>:   sbci   r19, 0x00      ; r19에서 0-carry를 뺀다.
0x000000c2 <+36>:   sbci   r25, 0x00      ; r25에서 0-carry를 뺀다.
0x000000c4 <+38>:   brne   .-8            ; main+32로 branch
0x000000c6 <+40>:   rjmp   .+0            ; main+42로 jump
0x000000c8 <+42>:   nop
0x000000ca <+44>:   rjmp   .-42           ; main+4로 jump
```

} _delay_ms(1000);

} _delay_ms(1000);

End of assembler dump.

(gdb)

1초를 만드는 방법

MCU의 속도 : 8,000,000hz = 1초에 8백만 clock 소모
⇒ 즉, 8백만 clock을 소모해야 1초가 흐른다.

```
0x000000a4 <+6>:      ldi      r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000a6 <+8>:      ldi      r19, 0x69      ; r19에 0x69를 저장한다.
0x000000a8 <+10>:     ldi      r25, 0x18     ; r25에 0x18를 저장한다.

0x000000aa <+12>:     subi     r18, 0x01      ; r18에서 1을 뺀다.
0x000000ac <+14>:     sbci     r19, 0x00      ; r19에서 0-carry를 뺀다.
0x000000ae <+16>:     sbci     r25, 0x00      ; r25에서 0-carry를 뺀다.
0x000000b0 <+18>:     brne     .-8          ; main+12로 branch
```

1초를 만드는 방법

MCU의 속도 : 8,000,000hz = 1초에 8백만 clock 소모
⇒ 즉, 8백만 clock을 소모해야 1초가 흐른다.

```
0x000000a4 <+6>:      ldi      r18, 0xFF      ; r18에 0xFF를 저장한다.
0x000000a6 <+8>:      ldi      r19, 0x69      ; r19에 0x69를 저장한다.
0x000000a8 <+10>:     ldi      r25, 0x18      ; r25에 0x18를 저장한다.

0x000000aa <+12>:     subi     r18, 0x01      ; 1 clock
0x000000ac <+14>:     sbci     r19, 0x00      ; 1 clock
0x000000ae <+16>:     sbci     r25, 0x00      ; 1 clock
0x000000b0 <+18>:     brne    .-8           ; 2 clock
```

$$(0xff(255)+1 * 0x69(105)+1 * 0x1(1)) * 5 = 135,680$$

$$(0xff(255)+1 * 0xff(255)+1 * 0x17(23)+1) * 5 = 7,864,320$$

$$135,680 + 7,864,320 = 8,000,000$$

0x00 주소엔 무엇이?

```
0x00 <__vectors>: rjmp .+138 ; 0x8c <__trampolines_start>
0x02 <__vectors+2>: nop
0x04 <__vectors+4>: rjmp .+150 ; 0x9c <__vector_9>
0x06 <__vectors+6>: nop
0x08 <__vectors+8>: rjmp .+146 ; 0x9c <__vector_9>
0x0a <__vectors+10>: nop
0x0c <__vectors+12>: rjmp .+142 ; 0x9c <__vector_9>
0x0e <__vectors+14>: nop
0x10 <__vectors+16>: rjmp .+138 ; 0x9c <__vector_9>
0x12 <__vectors+18>: nop
0x14 <__vectors+20>: rjmp .+134 ; 0x9c <__vector_9>
0x16 <__vectors+22>: nop
0x18 <__vectors+24>: rjmp .+130 ; 0x9c <__vector_9>
0x1a <__vectors+26>: nop
0x1c <__vectors+28>: rjmp .+126 ; 0x9c <__vector_9>
0x1e <__vectors+30>: nop
0x20 <__vectors+32>: rjmp .+122 ; 0x9c <__vector_9>
0x22 <__vectors+34>: nop
0x24 <__vectors+36>: rjmp .+118 ; 0x9c <__vector_9>
0x26 <__vectors+38>: nop
0x28 <__vectors+40>: rjmp .+114 ; 0x9c <__vector_9>
0x2a <__vectors+42>: nop
0x2c <__vectors+44>: rjmp .+110 ; 0x9c <__vector_9>
0x2e <__vectors+46>: nop
0x30 <__vectors+48>: rjmp .+106 ; 0x9c <__vector_9>
0x32 <__vectors+50>: nop
0x34 <__vectors+52>: rjmp .+102 ; 0x9c <__vector_9>
0x36 <__vectors+54>: nop
0x38 <__vectors+56>: rjmp .+98 ; 0x9c <__vector_9>
0x3a <__vectors+58>: nop
0x3c <__vectors+60>: rjmp .+94 ; 0x9c <__vector_9>
0x3e <__vectors+62>: nop
0x40 <__vectors+64>: rjmp .+90 ; 0x9c <__vector_9>
0x42 <__vectors+66>: nop
0x44 <__vectors+68>: rjmp .+86 ; 0x9c <__vector_9>
0x46 <__vectors+70>: nop
0x48 <__vectors+72>: rjmp .+82 ; 0x9c <__vector_9>
0x4a <__vectors+74>: nop
0x4c <__vectors+76>: rjmp .+78 ; 0x9c <__vector_9>
0x4e <__vectors+78>: nop
```


Interrupt Vectors

- Datasheet 59p~60p
- interrupt vector : 인터럽트가 발생했을 때, 그 인터럽트를 처리할 수 있는 서비스 루틴들의 주소를 가지고 있는 공간

Table 11-1. Reset and Interrupt Vectors

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	\$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	\$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	\$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	\$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B

Reset(trampolines_start) 분석

```
0x8c <__trampolines_start>:      eor    r1, r1    ; xor r1, r1 => 0
0x8e <__trampolines_start+2>:    out    0x3f, r1  ; SREG = 0
0x90 <__trampolines_start+4>:    ldi    r28, 0xFF ; r28 = 0xFF (Y-REG)
0x92 <__trampolines_start+6>:    ldi    r29, 0x10 ; r29 = 0x10 (Y-REG)
0x94 <__trampolines_start+8>:    out    0x3e, r29 ; SPH = 0x10
0x96 <__trampolines_start+10>:   out    0x3d, r28 ; SPL = 0xFF (SP=0x10FF)
0x98 <__trampolines_start+12>:   rcall  .+4      ; call <main>
0x9a <__trampolines_start+14>:   rjmp  .+48     ; jmp <exit>
```

- SREG 상태 레지스터를 0으로 초기화
- 스택 주소를 0x10FF로 초기화
- Main 함수 호출
- Main 종료 후 Exit 함수 호출

Exit 함수 분석

```
0xcc <exit>: cli  
0xce <__stop_program>: rjmp .-2 ; 0xcc <__stop_program>
```

- 인터럽트 비활성화
- 무한 루프

감사합니다.