Lab 1

Assembly Language 1

EEL 4746C / EEL 5756C: Microcomputers

Fall 2018

Student Name:	
Student ID:	
Section:	

Contents

1 Objective

The objective of this lab is to be able to trace an assembly program.

2 Introduction

In this experiment you will learn how to trace the execution of an assembly program in the memory and check the content of the memory. We will start by using Code 1 in section 4.1 to explain the experiment.

Step 1. Write code 1 and save it as file `lab1-1.asm`.

Step 2. Assemble the `lab1-1.asm` file and generate the object file `lab1-1.o` by using the following command:

```
>> avr-as -mmcu=atmega328p -ggdb -o lab1-1.o lab1-1.asm -a
```

Step 3. Check the output on the screen and make sure that there is no error in the code.

Step 4. Link the object file `lab1-1.o` to generate the executable file `lab1-1.x` by using the following command:

```
>> avr-ld -o lab1-1.x lab1-1.o
```

Step 5. In a different terminal, run the simulator using the following command:

```
>> simulavr -d atmega328 -g -F 20000000
```

Step 6. Now that the simulator is running, go back to the original terminal. In the terminal run the GNU debugger by typing the following command:

```
>> avr-gdb
```

Step 7. Inside the debug terminal, type the following command to connect to the server:

```
(gdb) target remote 127.0.0.1:1212
```

Step 8. Type the following commands to set the file to be loaded and loading the file (select yes when prompted to confirm changing file):

```
(gdb) file lab1-1.x
(gdb) load
```

Step 9. To see the machine code and its equivalent assembly code in the program memory, use the following command (note the command output is highlighted in blue color):

```
(gdb) disassemble /r 0x000000, +1
Dump of assembler code from 0x0 to 0x80000a:
=> 0x00000000 <start+0>:  f1 e0  ldi    r31, 0x01    ; 1
   0x00000002 <start+2>:  e0 e0  ldi    r30, 0x00    ; 0
   0x00000004 <start+4>:  0f e8  ldi    r16, 0x8F    ; 143
   0x00000006 <start+6>:  00 83  st     Z, r16
   0x00000008 <loop+0>:   ff cf  rjmp   .-2          ;  0x8 <loop>
   0x0000000a:      00 00  nop
```

Step 10. Use the following command to see the machine code with the original assembly source and the corresponding line numbers in the original assembly code (note the command output is highlighted in blue color):

```
(gdb) disassemble /m 0x000000, +1
Dump of assembler code from 0x0 to 0x800001:
4       ldi r31,hi8(a)
=> 0x00000000 <start+0>:  ldi    r31, 0x01     ; 1

5       ldi r30,lo8(a)
   0x00000002 <start+2>:  ldi    r30, 0x00     ; 0

6       ldi r16, 143
   0x00000004 <start+4>:  ldi    r16, 0x8F     ; 143

7       st  Z, r16
   0x00000006 <start+6>:  st     Z, r16

8     loop:
9       rjmp  loop
   0x00000008 <loop+0>:   rjmp   .-2           ;  0x8 <loop>

End of assembler dump.
```

Step 11. To look at the first 24 bytes content of the RAM (0x0100) in decimal as bytes, use the following command (note the command output is highlighted in blue color):

```
(gdb) x /24b 0x0100
0x800100:    -86    -86    -86    -86    -86    -86    -86    -86
0x800108:    -86    -86    -86    -86    -86    -86    -86    -86
0x800110:    -86    -86    -86    -86    -86    -86    -86    -86
```

Step 12. To look at the first 17 bytes content of the RAM (0x0100) in hexadecimal as bytes, use the following command (note the command output is highlighted in blue color):

```
(gdb) x /17xb 0x0100
0x800100:    0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa
0x800108:    0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa
0x800110:    0xaa
```

Step 13. To look at the first 32 bytes content of the RAM (0x0100) in hexadecimal as 16-bit words (halfword), use the following command (note the command output is highlighted in blue color):

```
(gdb) x /32x 0x0100
0x800100:    0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
0x800110:    0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
0x800120:    0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
0x800130:    0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
```

Step 14. To look at the first 32 bytes content of the RAM (0x0100) in hexadecimal as 32-bit words (word), use the following command (note the command output is highlighted in blue color):

```
(gdb) x /32xw 0x0100
 0x800100:    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800110:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800120:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800130:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800140:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800150:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800160:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800170:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
```

Note: The content of the RAM is not initialized after the load.

Step 15. To look at the content of the registers, run the following command (note the command output is highlighted in blue color):

```
(gdb) info registers
```

or

```
(gdb) i r
r0             0xaa 170
r1             0xaa 170
r2             0xaa 170
r3             0xaa 170
r4             0xaa 170
r5             0xaa 170
r6             0xaa 170
r7             0xaa 170
r8             0xaa 170
r9             0xaa 170
r10            0xaa 170
r11            0xaa 170
r12            0xaa 170
r13            0xaa 170
r14            0xaa 170
r15            0xaa 170
r16            0xaa 170
r17            0xaa 170
r18            0xaa 170
r19            0xaa 170
r20            0xaa 170
r21            0xaa 170
r22            0xaa 170
r23            0xaa 170
r24            0xaa 170
r25            0xaa 170
r26            0xaa 170
```

```
r27            0xaa 170
r28            0xaa 170
r29            0xaa 170
r30            0xaa 170
r31            0xaa 170
SREG           0x0  0
SP             0x0  0x0 <start>
PC2            0x0  0
pc             0x0  0x0 <start>
```

Step 16. To trace the program instruction by instruction, use the following command:

```
(gdb) si
5       ldi r30,lo8(a)
```

Note: the debugger will print the next instruction to be executed will the line number from the original source.

Step 17. Since the first instruction has a register (`r31`) as a destination, the change can be viewed by using the following command (note the command output is highlighted in blue color):

```
(gdb) i r
r0             0xaa 170
r1             0xaa 170
r2             0xaa 170
r3             0xaa 170
r4             0xaa 170
r5             0xaa 170
r6             0xaa 170
r7             0xaa 170
r8             0xaa 170
r9             0xaa 170
r10            0xaa 170
r11            0xaa 170
r12            0xaa 170
r13            0xaa 170
r14            0xaa 170
r15            0xaa 170
r16            0xaa 170
r17            0xaa 170
r18            0xaa 170
r19            0xaa 170
r20            0xaa 170
r21            0xaa 170
r22            0xaa 170
r23            0xaa 170
r24            0xaa 170
r25            0xaa 170
```

```
r26             0xaa 170
r27             0xaa 170
r28             0xaa 170
r29             0xaa 170
r30             0xaa 170
r31             0x1  1
SREG            0x0  0
SP              0x0  0x0 <start>
PC2             0x2  2
pc              0x2  0x2 <start+2>
```

Step 18. Repeat stepping into the code until you reach to the `st` instruction. The `st` instruction will store the content of register (`r16`) into the memory location that is pointed to by `Z(r31:r30)`. Check the content of the memory after stepping into the `st` instruction. Use the following command to examine the change to the RAM:

```
(gdb) x /16xb 0x0100
0x800100:     0x8f   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa
0x800108:     0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa   0xaa
(gdb) x /16xh 0x0100
0x800100:     0xaa8f 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
0x800110:     0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa 0xaaaa
(gdb) x /16xw 0x0100
0x800100:     0xaaaaaa8f    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800110:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800120:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
0x800130:     0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa    0xaaaaaaaa
```

Step 19. Look at the content of the registers file.

```
r0              0xaa 170
r1              0xaa 170
r2              0xaa 170
r3              0xaa 170
r4              0xaa 170
r5              0xaa 170
r6              0xaa 170
r7              0xaa 170
r8              0xaa 170
r9              0xaa 170
r10             0xaa 170
r11             0xaa 170
r12             0xaa 170
r13             0xaa 170
r14             0xaa 170
r15             0xaa 170
r16             0x8f 143
```

```
r17             0xaa 170
r18             0xaa 170
r19             0xaa 170
r20             0xaa 170
r21             0xaa 170
r22             0xaa 170
r23             0xaa 170
r24             0xaa 170
r25             0xaa 170
r26             0xaa 170
r27             0xaa 170
r28             0xaa 170
r29             0xaa 170
r30             0x0  0
r31             0x1  1
SREG            0x0  0
SP              0x0  0x0 <start>
PC2             0x8  8
pc              0x8  0x8 <loop>
```

Step 20. Explain the result in the previous step.

Step 21. Quit the debugger and stop the simulator. Repeat steps 5 to 15. Type the following command to run the whole program:

```
(gdb) continue
```

Step 22. Stop the running by pressing `[Ctrl]+[c]`.

Step 23. Look at the content of the registers and the RAM.

Step 24. Repeat steps 1 and 2 for Code 2 in section 4.2. Why there is an error?

Step 25. Fix the error by modifying line 10 to read:

```
            .byte 30, 31
```

Step 26. Repeat steps all the steps from 1 to 23. What is the difference from the previous code?

3 Procedure

Step 1. Repeat all the steps from 1-23 in the introduction on code 3 in section 4.3.

Step 2. Step into each instruction and indicate any modification to the memory or the registers.

4 Appendix

4.1 Code 1

```
.global start
.text
start:
                ldi r31,hi8(a)
                ldi r30,lo8(a)
                ldi r16, 143
                st  Z, r16
loop:
                rjmp  loop


.data
.org  0x00A0        ; Start of RAM for this assembler
a:
          .byte 100,0       ; values will not be
              intialized in RAM
          .hword   40       ; but address space will be
              reserved
```

4.2 Code 2

```
.global start
.text
start:
                ldi r31,hi8(a)
                ldi r30,lo8(a)
                ldi r16, 143
                st  Z, r16
loop:
                rjmp  loop
          .byte 30
          .hword 101
          .byte 20, 55

.data
.org  0x00A0        ; Start of RAM for this assembler
a:
          .byte 100,0       ; values will not be
              intialized in RAM
          .hword   40       ; but address space will be
              reserved
```

4.3 Code 3

```
.global start
.text

.set  a, 10
.set  b, 25
.set  c, 15
.set  d, -10
.set  e, 246
start:
          ldi   r16, a
          ldi   r17, b
          ldi   r18, 0
          cp    r16, r17    ; Compare content of r16, r17
          breq  if          ; Branch if equal
else:
          ldi   r18, c
          rjmp  endif
if:
          ldi   r18, d
endif:
          mov   r18, r18
loop:
          rjmp loop
.end

I can write whatever I want here ( Usually documentation
    ) .
```