Lab 2

Assembly Language 2

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 write assembly programs that perform variable assignment, data movements, and flow control. You will also learn how to use the GDB Text User Interface (TUI).

2 Procedure

2.1 Part 1 (Practice variable assignment)

Step 1. Type the listing in section 4.1.
Step 2. Assemble and link the program.
Step 3. Run the simulator and load the program in `avr-gdb`.
Step 4. Trace the program and observe the changes on the memory and the registers. Make sure to utilize the TUI commands from Section 3.
Step 5. What does this program do?

2.2 Part 2 (Loop and program control)

Step 1. Type the listing in section 4.2.
Step 2. Assemble and link the program.
Step 3. Run the simulator and load the program in `avr-gdb`.
Step 4. Trace the program and observe the changes on the memory and the registers. Make sure to utilize the TUI commands from Section 3.
Step 5. What does this program do?

2.3 Part 3 (Implementing a simple algorithm)

Step 1. Type the listing in section 4.3.
Step 2. Assemble and link the program.
Step 3. Run the simulator and load the program in `avr-gdb`.
Step 4. Trace the program and observe the changes on the memory and the registers. Make sure to utilize the TUI commands from Section 3.
Step 5. What does this program do?

2.4 Part 4 (Write your own code)

Step 1. Write an AVR assembly program that will find the average of 8 unsigned characters (bytes). The values are entered in the same way as the programs provided in the class. (Hint: use division by subtraction).
Step 2. Assemble and link the program.
Step 3. Run the simulator and load the program in `avr-gdb`.
Step 4. Trace the program and observe the changes on the memory and the registers. Make sure to utilize the TUI commands from Section 3.
Step 5. Did your program perform the required task?
Step 6. What are the limitations or drawbacks of your code?

3 GDB TUI commands

`tui enable`	Enable the TUI mode
`tui disable`	Disable the TUI mode
`layout <name>`	Display the selected layout of TUI. **`<name>`** can be one of the following: 1. **`src`**: display the source code 2. **`asm`**: display the assembly of the program memory content. 3. **`regs`**: display the registers content 4. **`split`**: display the source code and the assembly of the program memory content.
`[ctrl]+[x][1]`	One window
`[ctrl]+[x][2]`	Split windows
`[ctrl]+[x][o]`	Change window focus (in split window mode)

.

4 Appendix

4.1 Code 1

```
.text
.org 0x0000

.set op1, 100020
.set op2, 2053

reset_vector:
                                jmp start      ; skip
                                    interrupt vector table

.org 0x0100
start:
; get the bytes of operand 1
                                ldi r16, lo8(op1)
                                ldi r17, hi8(op1)
                                ldi r18, hlo8(op1)
                                ldi r19, hhi8(op1)

; store operand 1 to m
                                sts m, r16
                                sts m+1, r17
                                sts m+2, r18
                                sts m+3, r19

; get the bytes of operand 2
                                ldi r16, lo8(op2)
                                ldi r17, hi8(op2)
                                ldi r18, hlo8(op2)
                                ldi r19, hhi8(op2)

; store operand 2 to n
                                sts n, r16
                                sts n+1, r17
                                sts n+2, r18
                                sts n+3, r19

; get m
                                lds r16, m
                                lds r17, m+1
                                lds r18, m+2
                                lds r19, m+3

; get y
                                lds r20, n
                                lds r21, n+1
                                lds r22, n+2
                                lds r23, n+3
```

```
47
48  ; Add x+y
49                                  add r16, r20
50                                  adc r17, r21
51                                  adc r18, r22
52                                  adc r19, r23
53
54  ; store addition result into o
55                                  sts o, r16
56                                  sts o+1, r17
57                                  sts o+2, r18
58                                  sts o+3, r19
59
60  ; infinite loop
61  infiniteloop:
62                                  rjmp infiniteloop
63
64  .data
65  .org 0x00A0
66  m:
67                                  .skip 4, 0
68  n:
69                                  .skip 4, 20
70  o:
71                                  .skip 4, 40
```

4.2 Code 2

```
.global start
.text
.org 0x0000

.set op1, 15
.set op2, 16

reset_vector:
                                jmp start      ; skip
                                    interrupt vector table

.org 0x0100
start:
                                ldi   r16, lo8(op1)
                                ldi   r17, lo8(op2)
                                ldi   r18, 0
                                ldi   r19, 1
do:
                                add   r18, r16
                                sub   r17, r19
while:
                                brne do

; infinite loop
infiniteloop:
                                rjmp infiniteloop
.end

what does this code do?
```

4.3 Code 3

```
.global start
.text
.org 0x0000

.set op1, 150
.set op2, 12
.set op3, 230

reset_vector:
               jmp start     ; skip interrupt vector table

.org 0x0100
start:
               ldi r16, lo8(op1)
               ldi r17, lo8(op2)
               ldi r18, lo8(op3)
               mov r20, r16          ; minimum
               mov r21, r16          ; maximum
               mov r19, r16
               sub r19, r17
               breq  skip1
               brlo  less1
               mov r20, r17
               rjmp skip1
less1:
               mov r21, r17
skip1:
               mov r19, r18
               sub r19, r21
               brlo skip2
               mov r21, r18
skip2:
               mov r19, r18
               sub r19, r20
               brlo  less2
               rjmp skip3
less2:
               mov r20, r18
skip3:
               rjmp skip3

.end
```