Lab #0

Familiarization with The Lab

Florida Polytechnic
University

EEL4746C: Microcomputers

Fall 2018

Student Name: Peter A. Dranishnikov

Student ID: U0000005258

Lab Partner(s): N/A

Section: 01

Experiment Date: September 10th, 2018

Table of Contents

Introduction

The purpose of this lab was to introduce and familiarize with the assembly instructions, GNU toolchain, compilation, linking, and debugging on an AVR microcomputer. This lab had a simulator program instead of a hardware microcomputer for execution of the compiled program.

Discussion

The AVR assembly code below executed a load of a 32-bit hexadecimal number to 4 registers, the sum of the least two significant bytes stored into the least significant byte register without carry, and branching conditions depending on the result of different status register flags. In the set number, the sum of the two least significant bytes is zero, so the second branching instruction would trigger to label "j02," which sets register 20 to decimal 2 (using or immediate). The program ends with an infinite loop of writing the results of register 20 to an address repeatedly.

```
.global start
.set number1, 0x01AA02FE
.text
start:
      ldi r20, 0
      ldi r16, lo8(number1)
      ldi r17, hi8(number1)
      ldi r18, hh8(number1)
      ldi r19, hhi8(number1)
      add r16, r17
      brcc j01
      breq j02
      brpl j03
      brge j04
j01:
      ldi r20, 1
      rjmp infiniteloop
j02:
      ori r20, 2
      rjmp infiniteloop
j03:
      ori r20, 4
      rjmp infiniteloop
j04:
      ori r20, 8
infiniteloop:
      sts result1, r20
      rjmp infiniteloop
.data
.org 0x00A0
result1:
      .skip 4, 0xaa
result2:
      .skip 4, 0x55
```

Experimental procedure

The code previously displayed was written in a text editor and saved to disk as a text file. Using the GNU toolchain for AVR assembly, the code was assembled, then linked to an executable. Using the simulator, the executable was loaded and debugged through gdb. The code below summarizes the steps of assembly and linking (lab0.asm is assumed to be the source file):

```
$ avr-as -mmcu=atmega328p -o lab0_out.o lab0.asm -ggdb
$ avr-ld -o lab0_exec.elf lab0_out.o
$ simulavr -d atmega328 -g -F 20000000 &
$ avr-gdb
(gdb) target remote localhost:1212
(gdb) file lab0_exec.elf
(gdb) load lab0_exec.elf
(gdb) continue
^C
(gdb) disassemble 0x00000000, 0x00000026
(gdb) info registers
```

Results/Measurements/Observations

Register values upon termination:

```
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            0x0      0
r17            0x2      2
r18            0xaa     170
r19            0x1      1
r20            0x2      2
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           0x21     33
SP             0x0      0x0 <start>
PC2            0x22     34
pc             0x22     0x22 <infiniteloop>
```

Disassembled executable (in simulator memory):

```
   0x00000000 <start+0>:     ldi   r20, 0x00   ; 0
   0x00000002 <start+2>:     ldi   r16, 0xFE   ; 254
   0x00000004 <start+4>:     ldi   r17, 0x02   ; 2
   0x00000006 <start+6>:     ldi   r18, 0xAA   ; 170
   0x00000008 <start+8>:     ldi   r19, 0x01   ; 1
   0x0000000a <start+10>:    add   r16, r17
   0x0000000c <start+12>:    brcc  .+6         ;  0x14 <j01>
   0x0000000e <start+14>:    breq  .+8         ;  0x18 <j02>
   0x00000010 <start+16>:    brpl  .+10        ;  0x1c <j03>
   0x00000012 <start+18>:    brge  .+12        ;  0x20 <j04>
   0x00000014 <j01+0>:  ldi   r20, 0x01   ; 1
   0x00000016 <j01+2>:  rjmp  .+10        ;  0x22 <infiniteloop>
   0x00000018 <j02+0>:  ori   r20, 0x02   ; 2
   0x0000001a <j02+2>:  rjmp  .+6         ;  0x22 <infiniteloop>
   0x0000001c <j03+0>:  ori   r20, 0x04   ; 4
   0x0000001e <j03+2>:  rjmp  .+2         ;  0x22 <infiniteloop>
   0x00000020 <j04+0>:  ori   r20, 0x08   ; 8
=> 0x00000022 <infiniteloop+0>:    sts   0x0100, r20
   0x00000026 <infiniteloop+4>:    rjmp  .-6         ;  0x22 <infiniteloop>
   0x00000028:    nop
   0x0000002a:    nop
   0x0000002c:    nop
```

Result Discussion

The disassembled assembly referred to offsets instead of labels (gdb adds the label values from the debugging symbols intentionally included in the assembly step). The “nop”s at the end indicated empty space beyond the executable.

Conclusion

The expected behavior and the actual behavior of the code is identical and the disassembled executable logically matched the original source code. Therefore, it is possible to assemble, link, debug, and disassemble AVR assembly code on a microcomputer simulator.

Answers to lab’s questions

There were no applicable lab questions.