

LAB 1

AVR ASSEMBLER & SIMULATOR

OBJECTIVES:

To examine and use an AVR assembler.

To examine and use an AVR simulator.

To examine the flag bits of the SREG

MATERIAL:

Atmel Studio or the assembler of your choice.

WEB SITES:

www.microchip.com for Atmel Studio Software

ACTIVITY 1

Write and assemble the following program. Use the simulator to single-step to examine the value that is contained in the registers.

```
LDI R16,0xFF  
OUT DDRB,R16  
  
L1: OUT PORTB,R16  
LDI R20,0  
OUT PORTB,R20  
RJMP L1
```

1) What is the value of DDRB after executing OUT DDRB, R16?

DDRB = 0xFF and all pins as output

2) What happens to PORTB immediately after:

- OUT PORTB, R16

Register load with 0xFF, all 8 pins go high(on)

- OUT PORTB, R20

Register load with 0x00, all 8 pins go low(off)

LAB 1

AVR ASSEMBLER & SIMULATOR

ACTIVITY 2

Write and assemble a program to add all the single digits of your ID number using R0 and R24 and save the result in R16. Pick 7 random numbers (all single digits) if you do not want to use your ID number. Then use the simulator to single-step the program and examine the registers of each step.

- 1) Indicate the size (8- or 16-bit) of each of the following registers for each step.

R0 = R24 = R16 = 8 bit

Data in the memory pointed by current PC = 16 bit

My id is 6581166

LDI R16, 0 ; R16 = 0 (sum)

LDI R0, 6

ADD R16, R0 ; R16 = 6

LDI R24, 5

ADD R16, R24 ; R16 = 6 + 5 = 11

LDI R0, 8

ADD R16, R0 ; R16 = 11 + 8 = 19

LDI R24, 1

ADD R16, R24 ; R16 = 19 + 1 = 20

LDI R0, 1

ADD R16, R0 ; R16 = 20 + 1 = 21

LDI R24, 6

ADD R16, R24 ; R16 = 21 + 6 = 27

LDI R0, 6

ADD R16, R0 ; R16 = 27 + 6 = 33 (0x21)

ACTIVITY 3

Write and assemble a program to load register R20 with value \$99. Then from register

R20 move it to R0, R12, and R31. Use the simulator to single-step the program and examine the registers.

1) Can LDI load an immediate value directly into below registers? If it is not, please give the reason.

- R0, No because it is in the lower half. Meaning LDI cannot load it immediately.
- R12, No because it is in the lower half. Meaning LDI cannot load it immediately.
- R20, Yes because it is in the upper half.
- R31, Yes because it is in the upper half.

```
LDI R20, 0x99  
MOV R0, R20  
MOV R12, R20  
MOV R31, R20
```

ACTIVITY 4

Write and assemble a program to add the following data and then use the simulator to examine the C, H and Z flags after the execution of each addition. \$92, \$23, \$66, \$87, \$F5

```
LDI R16, 0x00  
; Add $92  
LDI R17, 0x92  
ADD R16, R17      ; 0x00 + 0x92 = 0x92, Check flags: C=0, H=1, Z=0  
  
; Add $23  
LDI R17, 0x23  
ADD R16, R17      ; 0x92 + 0x23 = 0xB5, Check flags: C=0, H=0, Z=0  
  
; Add $66  
LDI R17, 0x66  
ADD R16, R17      ; 0xB5 + 0x66 = 0x1B with carry, Check flags: C=1, H=1, Z=0  
  
; Add $87  
LDI R17, 0x87  
ADD R16, R17      ; 0x1B + 0x87 = 0xA2, Check flags: C=0, H=0, Z=0
```

```

; Add $F5
LDI R17, 0xF5
ADD R16, R17      ; 0xA2 + 0xF5 = 0x197 (keeps 0x97 in R16), Check flags: C=1,
H=0, Z=0

```

ACTIVITY 5

Write and assemble the following program. Use the simulator to single-step and examine the flags and register content after the execution of each instruction.

```

LDI R20,$27 ; R20 = 0x27
LDI R21,$15 ; R21 = 0x15
SUB R20, R21 ; R20 = 0x27 - 0x15 = 0x12

LDI R20,$20 ; R20 = 0x20
LDI R21,$15 ; R21 = 0x15
SUB R20, R21 ; R20 = 0x20 - 0x15 = 0x0B

LDI R24,95 ; R24 = 95 = 0x5F

LDI R25,95 ; R25 = 95 = 0x5F
SUB R24, R25 ; R24 = 0x5F - 0x5F = 0x00

LDI R22,50 ; R22 = 50 = 0x32
LDI R23,70 ; R23 = 70 = 0x46
SUB R22, R23 ; R22 = 0x32 - 0x46 = 0xEC (with borrow)

L1: RJMP L1

```

ACTIVITY 6

- Write and assemble a program to load a value into each location of R20 – R23. Use the COM instruction to complement the value in each register. Use the simulator to single-step and examine the flags and register content after the execution of each instruction.

```

LDI R20,0      ;R20=$00
LDI R21,0xFF   ;R21=$FF
LDI R22,0x11   ;R22=$11
LDI R23,0x22   ;R23=$22

COM R20 ;R20=$FF, Flags: C=1 Z=0
COM R21 ;R21=$00, Flags: C=1 Z=1
COM R22 ;R22=$EE, Flags: C=1 Z=0
COM R23 ;R23=$DD, Flags: C=1 Z=0

L1: RJMP L1

```

b) Find the value of the C flag after the execution of the following codes.

(1) LDI R20, \$85

LDI R21, \$92

ADD R20, R21 :C=1

(2) LDI R16, \$15

LDI R17, \$72

ADD R16, R17 :C=0

(3) LDI R25, \$F5

LDI R26, \$52

ADD R25, R26 :C=1

(4) LDI R25, \$FE

INC R25

INC R25 :C = 0

ACTIVITY 6

Write and assemble the following program. Use the simulator to single-step to examine how the program works. Give explanations line by line as well as the data in status register

```

.EQU SUM = 0x300
.ORG 00

LDI R16, 0x25
LDI R17, $34
LDI R18, 0b000110001
ADD R16, R17
ADD R16, R18
LDI R17, 11
ADD R16, R17
STS SUM, R16

```

HERE: JMP HERE

- a) After building the program, check the “Memory” windows with “0x00,prog”. Find the last address of the program segment.
0x000A for word address or 0x0014 in byte address
- b) Change .ORG 0x00 to .ORG0x100. Find the start and last address of program.
Start address at 0x0000 and last address at 0x0107

```

.EQU SUM = 0x300
.ORG 0x00

LDI R16, 0x25
LDI R17, $34
LDI R18, 0b000110001

.ORG 0x100

ADD R16, R17
ADD R16, R18
LDI R17, 11
ADD R16, R17
STS SUM, R16

```

HERE: JMP HERE

c) What happens to our compiled program when we add .ORG 0x100 as above code.

Adding .ORG 0x100 makes the assembler skip to 0x100. Forcing assembler to place subsequent instructions starting only at 0x0100. This creates a gap of unused memory. This wastes time and memory, and is inefficient without instruction to bridge the gap.

d) Why is the address not 0x100?

Appear as 0x200 instead, due to the difference between word and byte addressing.

```
.SET RAM_START = 0x0100
.CSEG
.ORG 0x0000
hello_data:
    .DB "HELLO WORLD!"

main:
    LDI YL, LOW(RAM_START)
    LDI YH, HIGH(RAM_START)
    LDI ZL, LOW(2*hello_data)
    LDI ZH, HIGH(2*hello_data)
    LDI R18, 12

copy_loop:
    LPM R16, Z+
    ST Y+, R16
    DEC R18
    BRNE copy_loop

END:
    rjmp END
```

LAB 1

AVR ASSEMBLER & SIMULATOR

- e) What does this program do? Explain line-by-line of the code by debugging and observing the changes in the SRAM.

Line 1: Set RAM_START to 0x0100 where copy will start

Line 2: code goes to program memory

Line 3: start assembling at 0x0000

Line 4: Label for data location

Line 5: Stores 12 ASCII characters

Line 6: main

Line 7-8: Load address 0x0100 into Y pointer, Y point to SRAM destination

Line 9-10: Load address of string to Z pointer, 2* is need because LPM use byte address but program memory is word addressable in AVR

Line 11: Counter, which = 12

Line 12: Loop

Line 13: Load program memory, read byte from Z pointer into R16, increment Z

Line 14: Copy R16 to SRAM at Y pointer, increment Y

Line 15: Decrement counter

Line 16: go back to copy_loop if counter is not 0

Line 17-18: Infinite loop

Debugging in SRAM: if we debug it in simulator and view memory window we can see that:

Before execution , SRAM starting at 0x0100 will be empty

After execution, address 0x0100 will have ASCII hex values for “HELLO WORLD!” sequentially.

0x0100: 'H' (0x48)

0x0101: 'E' (0x45)

0x0102: 'L' (0x4C)

0x0103: 'L' (0x4C)

0x0104: 'O' (0x4F)

0x0105: ' ' (0x20)

0x0106: 'W' (0x57)

0x0107: 'O' (0x4F)

0x0108: 'R' (0x52)

0x0109: 'L' (0x4C)

0x010A: 'D' (0x44)

0x010B: '!' (0x21)