EE2016 Micropprocessor Lab & Theory. Aug - Nov, 2024

Dept. of Electrical Engineering, IIT, Madras.

Experiment 2: Computations using Atmel Atmega8 AVR through Assembly Program Emulation

1 Aim

To implement basic arithmetic and logical manipulation programs using Atmel Atmega8 microcontroller in assembly program emulation, including addition, multiplication and comparison.

2 Equipments (Hardware/ Software) Required

A Windows PC with Microchip Studio IDE (Integrated Development Environment) for developing and debugging AVR & SAM microcontroller applications.

3 Basic Concepts / Familiarity Required

The basic concepts of microcontroller architecture, general purpose programming with the corresponding instruction sets should be very clear. Since the real-life/ hardware assembly language programming of Atmega microproccessor would be carried out in the next week's lab (Experiment 3), the hardware description will be covered before Experiment 3. Here, in Experiment 2, our focus is only on the emulation of assembly programming of Atmega8 microcontroller.

3.1 Approaches Available

In this lab session, you are asked to add given two 8 bit numbers and save the result in a register. There are two ways of doing this: (1) to implement in real-life on the atmega processor using machine language or assembly language and demonstrate and (2) to demonstrate by emulating, the execution of assembly language program on a PC loaded with Microchip simulator.

The first choice (choice (1)) would require (A) Atmega8 microcontroller along with other accessaries and (B) (Integrated Development Environment) IDE. In the case of Atmel's Atmega8 microcontroller, the IDE is the Microchip Studio software. This IDE primarily allows one to develop a project involving Atmega microcontroller and provides an user friendly environment of edit - run - debug - edit cycles till acceptable performance is achieved. (It is a common practice, that the IDE is used only at the development and testing phase, while it is a natural choice for educational purposes). Once the system is tested to satisfaction, the microcontroller, loaded with the developed code (the code burnt in its memory - either EEPROM or flash, which is non-volatile that can be electrically erased and reprogrammed) is ready for deployment. By and large the IDE is used for developmental purpose in engineering practice. Refer Mazidi, pages 82 - 83.

Thus, the second choice (2) of emulation requires only a Windows PC with the Microchip Studio (for AVR class of processors) emulator software loaded.

3.2 Emulation of Atmel AVR Assembly Programming

In this experiment, we adopt the second approach, i.e., emulation of Atmega8 microcontroller (leaving the real-life atmega8 microcontroller programming to the next week) in IDE.

3.3 Basic Concepts and Information Required

Following information is required (Refer the corresponding documents indicated in paranthesis)

A) Atmel AVR instruction set (Refer Atmel 8-bit AVR Instruction Set uploaded in moodle).

A) Atmel AVR assembly programming limited to

a) binary addition

a) binary multiplication

a) compare two binary words

(Refer Atmel AVR Assembler User Guide 1998)

A) Atmega8 microcontroller architecture (of ALU, registers and instruction cycles)

A) Microchip Studio 7 simulator (Refer Microchip studio user guide uploaded in moodle)

All of the above relevant documents are uploaded in Moodle.

3.4 Demo program

The following table provides a brief overview of some of the instructions that might be useful for this experiment.

Mnemonics	Operands	Description	Operation
ADD	Rd, Rr	Add without Carry	Rd ←Rd + Rr
ADC	Rd, Rr	Add with Carry	Rd ←Rd + Rr + C
SUB	Rd, Rr	Subtract without Carry	Rd ←Rd - Rr
SBC	Rd, Rr	Subtract with Carry	Rd ←Rd - Rr - C
AND	Rd, Rr	Logical AND	Rd ←Rd •Rr
OR	Rd, Rr	Logical OR	Rd ←Rd ⋁ Rr
INC	Rd	Increment	Rd ←Rd + 1
DEC	Rd	Decrement	Rd ←Rd - 1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ←Rd * Rr
CP	Rd, Rr	Compare	Rd - Rr
BREQ	k	Branch if Equal	If (Z = 1) then PC ←PC + k + 1
MOV	Rd, Rr	Copy Register	Rd ←Rr
LDI	Rd, K	Load Immediate	Rd ←K
ST	X, Rr	Store Indirect	(X) ←Rr
LSL	Rd	Logical Shift Left	Rd(n+1) ←Rd(n), Rd(0) ←0, C ←Rd(7)

The above can be categorized as arithmetic and logic instructions, branch instructions, data transfer instructions, and bit and bit-test instructions (Identify the category under which each instruction falls, as an exercise). Please note that there are several variants of the above instructions, notably the immediate, with carry, indirect, and the if X-condition variants for branching. To learn more about these, view the AVR Assembler User Guide on Moodle. [Or refer the backside in Mazidi et al].

3.4.1 Demo Program

The following is the demo program which performs the addition of two numbers in Atmel Atmega 8 processor. With this, the student is expected to write programs which does subtraction and multiplication.

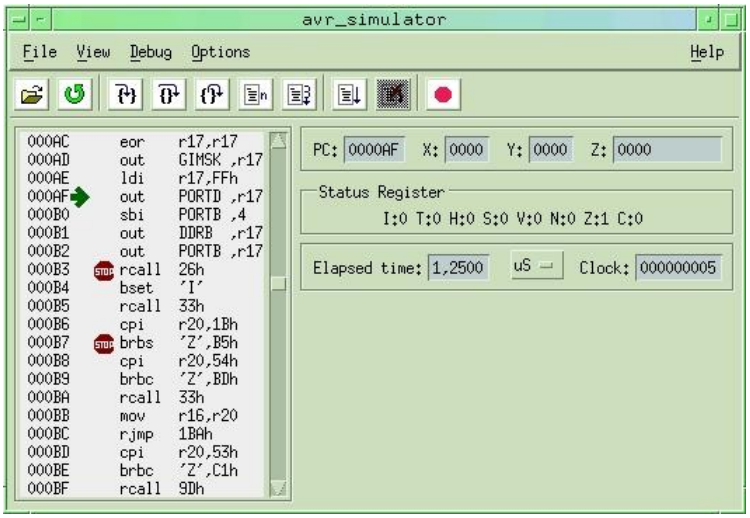
```
/*
* add.asm
*
*Created: 8/3/2018 11:43:15 AM
*Author:Students
*Program to add two numbers in memory and store the result in a given memory location say with address
1  .CSEG; define memory space to hold program - code segment
2  LDI ZL,LOW(NUM<<1); load byte addrss of LSB of word addrss
3  LDI ZH,HIGH(NUM<<1);load byte addrss of MSB of word addrss
4  LDI XL,0x60; load SRAM LSB of 16-bit address in X-register
5  LDI XH,0x00; above's MSB|word address can be line number here
6  LDI R16,00; clear R16, used to hold carry
7  LPM R0,Z+;Z now follows byte addrssng. points MSB of NUM addr
8  LPM R1,Z; Get second number (LSB of NUM addr) into R1,
9  ADD R0,R1; Add R0 and R1,result in R0,carry flag affected
10  BRCC abc; jump if no carry,
11  LDI R16,0x01 ; else make carry 1
12 abc:   ST X+,R0 ; store result in given address in SRAM ie 0x60
13  ST X,R16 ; store carry in next location 0x61
```

```
14  NOP ; End of program, No operation
15 NUM: .db 0xD3,0x5F; bytes to be added
; .db define data byte directive, inserts one or more
; constant bytes in the code segment (The number of
; inserted bytes must be even, otherwise an
; additional zero byte will be inserted by the
; assembler.)
..................... Actual convention used by the assembler
in terms of line number or address is as given below. Also, note
the convention used by assembler / Studio debugger for the
byte number towards byte addressing scheme ............
  .CSEG ; not counted for the purpose of addressing
0  LDI ZL,LOW(NUM<<1); line number is '0'. Byte # 0 & 1.
1  LDI ZH,HIGH(NUM<<1); Bytes 2, 3 [All instructions in
2  LDI XL,0x60; AVR are two bytes]. Bytes 4, 5
3  LDI XH,0x00; Bytes 6, 7 [How these bytes map back to
4  LDI R16,00; components of instruction is unclear] 8, 9
5  LPM R0,Z+; Bytes 10, 11
6  LPM R1,Z; Bytes 12, 13
7  ADD R0,R1; Bytes 14, 15
8  BRCC abc; Bytes 16, 17
9  LDI R16,0x01 ; Bytes 18, 19
10 abc:   ST X+,R0; Bytes 20, 21
11  ST X,R16; Bytes 22, 23
12  NOP; Bytes 24, 25
13 NUM: .db 0xD3,0x5F; Bytes 26, 27
; with the above convention, whatever I taught you today morning
; in class makes sense. Assembler ignores empty lines after NOP
; (if at all there is). We checked in the Studio also.
```

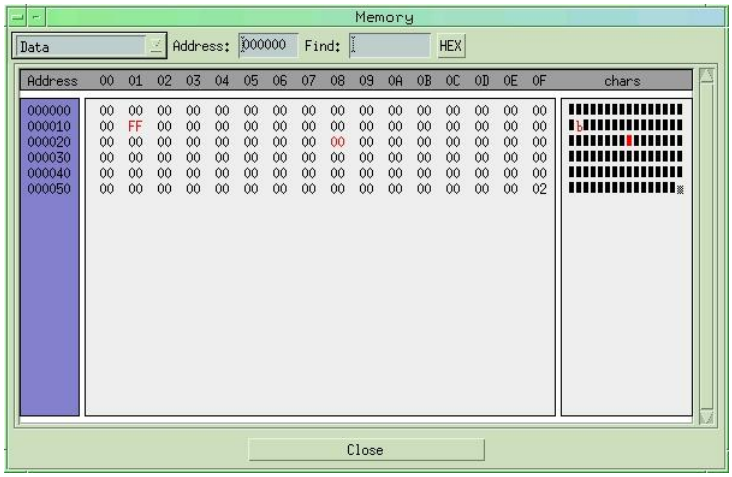
4 Running the Experiment in Atmel Studio 6.2 (Older version. Now Microchip Studio)

A screen shot of Atmel AVR Simulator (From an older Version):

(a) Main Window

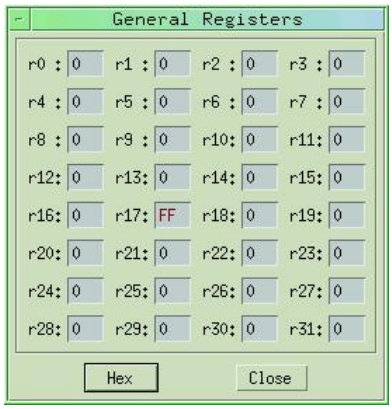


(b) Memory Window



.

.(c) Content of General Registers



5 Experiment

5.1 Problems to Do

Problem 1 (Common 8-bit mathematical operation): Given two 8-bit binary words (byte), compute the sum and store it in a register.
Problem 2 (16-bit addition using a 8-bit processor): Given two 16-bit binary words (byte), compute the sum and store it in a register.
Problem 3 (Multiplication of two 8-bit numbers): Given two 8-bit binary words (byte), compute the product and store it in a register.
Problem 4 (Largest of number given): Given a finite set of binary words, identify the largest in the given set.

5.2 General Procedure

Step 1 Draw the flow chart for the above problem.

Step 2 Convert the flow chart into the assembly language program, identifying the corresponding instructions from the Atmega8 instruction set [Till now, get the procedure corrected by the TA].

Step 3 Get it compiled by the AVR compiler.

Step 4 Run the program. Keep an eye on the values of the register for each cycle [Ask the TA, how to debug using AVR simulator].

Step 5 Verify the result with the manually calculated answer. If not, debug [Take help of TA, if necessary]. Demonstrate it to TA once your code does what it is supposed to do.

6 Results and Inference

Addition, substraction, multiplication, and comparison of 8-bit / 16-bit binary numbers are demonstrated through emulation of Atmega8 assembly programming.