

Machine Language Instructions, warming up with SPIM simulator, high-to-low level language conversion

Submitted By
Fahd Humayun
168000889 (fh186)

Assignment 1

Instruction	op	rs	rt	constant
	6 bits	5 bits	5 bits	16 bits
ori \$9, \$0, 1	001101	00000	01001	0000 0000 0000 0001

Assignment 2

```
.globl main                                #by making main function global it will be
                                           #visible to code stored in other files

main:                                     #the beginning of main function
    lw $t0, Number1($0)                 #load word to register t0 ($t2) i.e. 18+0=18
    ori $t1, $0, 1                      #or immediate register t1 ($t3) i.e. $t3 = 0+1 = 1 (initialize $t3
    #to 1)
    ori $9, $0, 1                      #initialize $t1 to 1

#compute the factorial of Number in register t0 ($t0 or $t2)
factloop:                               #beginning of the loop/iterations to compute the factorial
    bge $t1, $t0, factexit             #jump to factexit if $t1 is greater than or equal to $t0
    mul $9, $t0, $9                    #multiply $t0 with $9 and store it in $9
    sub $t0, $t0, 1                    #subtract the value in $t0 by 1
    j factloop                         #jump to factloop

factexit:                               #when the above condition is satisfied it will jump to this line of
the code

#the computation of the factorial is over

#the result in $9 is half of what the original result is, because, the factorial of 18 results
in
#6402373705728000 (in decimal) which converted to hex results in 16BEECCA730000. So, the result
is
#clearly more than 32 bits which the register can not store, so the register stores the
last/lower 32 bits
#which the $9 represents that is CA730000, while the higher 32 bits 16BEECCA can not be
observed in the
#result by $9

    li $2, 10                          #load immediate $2 or $v0 with 10
    syscall                             #run the operation depending upon the code (value) loaded in $2

    .data 0x10000000                    #used for the data to be contained in the code
    .align 2                           #means next item should be on a word boundary

Number1: .word 18                      #Number1 is a word initialized to 18
```

Assignment 3

1. Missing instructions filled out according to the comments

```
.text 0x00400000
.align 2
.globl main

main:
    lw $a0, size           #load the size of array into $a0, using lw
    li $a1, 0              #initialize index i to 0
    li $t2, 4              #t2 contains constant 4, initialize t2
    ori $t3, $0, 1         #initialize result to one

loop:
    mul $t1, $a1, $t2       #t1 gets i*4
    lw $a3, Array($t1)      #a3 = Array[i]
    mul $t3, $t3, $a3       #result = result * Array[i]
    sw $t3, Array2($t1)     #store result in the Array2 in location i
    addi $a1, $a1, 1        #i = i + 1
    blt $a0, $a1, END      #go to END if finished
    j loop

END:

    li $v0, 10              #load 10 to v0 for exit
    syscall

.data 0x10000000
.align 2

#data to be used
Array: .word 2 5 6 7 12 16
size: .word 6
Array2: .space 24
```

2. The program was run step by step and the change in each register observed is summarized in the table below the values are represented in decimal

Registers						
\$a0	\$a1	\$a3	\$t1	\$t2	\$t3	\$v0
6	0	0	0	4	1	0
6	0	2	0	4	2	0
6	1	5	4	4	10	0
6	2	6	8	4	60	0
6	3	7	12	4	420	0
6	4	12	16	4	5040	0
6	5	16	20	4	80640	0
6	6	6	24	4	483840	0
6	7	6	24	4	483840	10

3.

```
User data segment [10000000]..[10040000]
[10000000] 00000002 00000005 00000006 00000007 . . . . .
[10000010] 0000000c 00000010 00000006 00000002 . . . . .
[10000020] 0000000a 0000003c 000001a4 000013b0 . . . . < . . . . .
[10000030] 00013b00 00076200 00000000 00000000 . ; . . . b . . . . .
[10000040]..[1003ffff] 00000000
```

4. The *index* i reaches $size + 1$ and not $size$ because the instruction to end the loop i.e. *blt* $\$a0, \$a1, END$ is at the end of the loop and has condition for less than, therefore, when the *index* reaches the *size* the loop will run one more time because the condition is still not true to end the loop as according to the condition *size* should be less than the *index*, while the *size* is equal to the *index* and not less than. When the loop runs again the *index* is incremented again and that is why it reaches $size + 1$, now the condition is satisfied as *size* is less than the *index*.

Additional: The extra iteration in the loop makes the output/result wrong because it multiplies a number to the result which is not part of the array elements. This can be fixed by either placing the instruction for terminating the loop at the beginning of the loop or by changing the condition to *ble* that is to check if the *size* is less than or equal to the *index*, which ends in giving the correct output/result as well.

Assignment 4

1. When the code was typed and executed, the only registers used in the code for first part are $\$a0, \$t1$, and $\$v0$, the value of each register recorded is in the table below

Registers		
$\$v0$	$\$t0$	$\$t1$
5	0	0
7 (user input number)	0	0
7	7	0
5	7	0
1 (user input number)	7	1

2. Register $\$v0$ is used for system's input and output operations. The register is loaded with values from 1 – 10 which is used for the operations of printing integers, float, double, and string to reading integers, float, double, and string. Each value that is loaded to $\$v0$ has particular operation specified. Other than printing or reading it is also used to exit the code by loading it with 10. *syscall* is used whenever a value is loaded into $\$v0$ for reading, printing, or terminating, without *syscall* the operation intended would not be performed and the numbers from 1 – 10 would just be considered as some values stored in $\$v0$.

3.

```
.text
main:

#prompt user for first number, load message from memory to $a0 for printing
li $v0, 4          #load syscall print_string into $v0
la $a0, Message1   #load address of data Message1 into $a0
syscall            #make the syscall

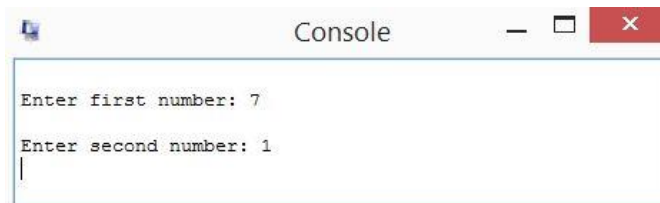
#get first number from user, put into $t0
li $v0, 5          #load syscall read_int into $v0
syscall            #make the syscall
move $t0, $v0      #move the number read into $t0

#prompt user for second number, load message from memory to $a0 for printing
li $v0, 4          #load syscall print_string into $v0
la $a0, Message2   #load address of data Message1 into $a0
syscall            #make the syscall

#get second number from user, put into $t1
li $v0, 5          #load syscall read_int into $v0
syscall            #make the syscall
move $t1, $v0      #move the number read into $t1

.data
Message1: .asciiz "\nEnter first number: "    #declare and initialize Message1 as string
Message2: .asciiz "\nEnter second number: "   #declare and initialize Message2 as string
```

The output/console window of the code executed with the changes to the code to print a message for user to enter the number



```
Console

Enter first number: 7

Enter second number: 1
|
```

```

.text
main:

#prompt user for first number, load message from memory to $a0 for printing
    li $v0, 4          #load syscall print_string into $v0
    la $a0, Message1   #load address of data Message1 into $a0
    syscall            #make the syscall

#get first number from user, put into $t0
    li $v0, 5          #load syscall read_int into $v0
    syscall            #make the syscall
    move $t0, $v0      #move the number read into $t0

#prompt user for second number
    li $v0, 4          #load syscall print_string into $v0
    la $a0, Message2   #load address of data Message2 into $a0
    syscall            #make the syscall

#get second number from user, put into $t1
    li $v0, 5          #load syscall read_int into $v0
    syscall            #make the syscall
    move $t1, $v0      #move the number read into $t1

#prompt user to input a message
    li $v0, 4          #load syscall print_string into $v0
    la $a0, Message3   #load address of data Message3 into $a0
    syscall            #make the syscall

#get message from user, put into memory location
    li $v0, 8          #load syscall read_string into $v0
    la $a0, UserMessage #load address of data UserMessage into $a0
    li $a1, 100        #load immediate $a1 with 100
    move $t3, $a0      #move $a0 to $t3
    syscall            #make the syscall

    sub $t2, $t0, $t1   #subtract $t1 from $t0, store result in $t2

#print message
    li $v0, 4          #load syscall print_string into $v0
    la $a0, Message4   #load address of data Message4 into $a0

```

4.


```

        syscall          #make the syscall

#print the message that user input
        la $a0, UserMessage #load address of data UserMessage into $a0
        move $a0, $t3       #move $t3 to $a0
        li $v0, 4           #load syscall print_string into $v0
        syscall            #make the syscall

#print the message for result
        li $v0, 4           #load syscall print_string into $v0
        la $a0, Message5    #load address of data Message5 into $a0
        syscall            #make the syscall

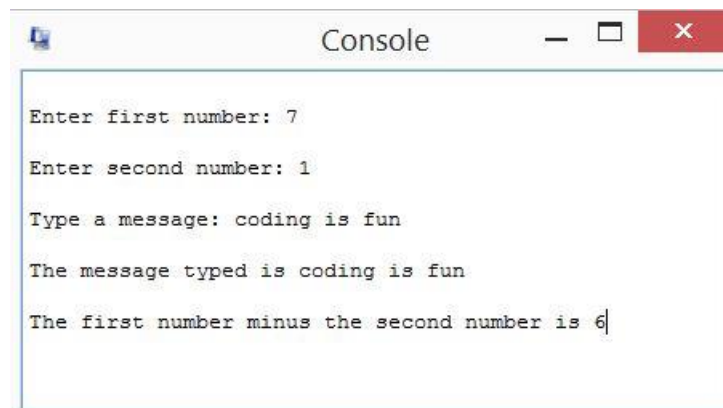
#print the result of difference stored in $t2
        li $v0, 1           #load syscall print_int into $v0
        move $a0, $t2       #move $t2 to $a0
        syscall            #make the syscall

#terminate the program
        li $v0, 10          #load syscall exit into $v0
        syscall            #make the syscall

#declare and initialize strings for printing the messages on the screen and
#also an empty string with 100 bytes of space which can store 100 characters
.data
Message1: .asciiz "\nEnter first number: "
Message2: .asciiz "\nEnter second number: "
Message3: .asciiz "\nType a message: "
Message4: .asciiz "\nThe message typed is "
Message5: .asciiz "\nThe first number minus the second number is "
UserMessage: .space 100

```

Console Window:



```

Console
Enter first number: 7
Enter second number: 1
Type a message: coding is fun
The message typed is coding is fun
The first number minus the second number is 6

```

Assignment-5

.text

#beginning of main function

main:

#display msg to **enter** first pair

```
li $v0, 4          #load syscall print_string into $v0
la $a0, msg1        #load address of data msg1 into $a0
syscall            #make the syscall
```

#prompt user **for** x1

```
li $v0, 4          #load syscall print_string into $v0
la $a0, x1          #load address of data x1 into $a0
syscall            #make the syscall
```

#get x1 from user, put **into** \$t0

```
li $v0, 5          #load syscall read_int into $v0
syscall            #make the syscall
```

```
move $t0, $v0      #move $v0 to $t0
```

#prompt user **for** y1

```
li $v0, 4          #load syscall print_string into $v0
la $a0, y1          #load address of data y1 into $a0
syscall            #make the syscall
```

#get y1 from user, put **into** \$t1

```
li $v0, 5          #load syscall read_int into $v0
syscall            #make the syscall
```

```
move $t1, $v0      #move $v0 to $t1
```

#display msg to **enter** second pair

```
li $v0, 4          #load syscall print_string into $v0
la $a0, msg2        #load address of data msg1 into $a0
syscall            #make the syscall
```

#prompt user **for** x2

```
li $v0, 4          #load syscall print_string into $v0
la $a0, x2          #load address of data x2 into $a0
```

```

        syscall                #make the syscall

#get x2 from user, put into $t2
    li $v0, 5                  #load syscall read_int into $v0
    syscall                    #make the syscall

    move $t2, $v0              #move $v0 to $t2

#prompt user for y2
    li $v0, 4                  #load syscall print_string into $v0
    la $a0, y2                 #load address of data y2 into $a0
    syscall                    #make the syscall

#get y2 from user, put into $t3
    li $v0, 5                  #load syscall read_int into $v0
    syscall                    #make the syscall

    move $t3, $v0              #move $v0 to $t3

#calculation for manhattan distance
    sub $t4, $t0, $t2          #subtract $t2 from $t0, store result in $t4
    sub $t5, $t1, $t3          #subtract $t3 from $t1, store result in $t5
    add $t6, $t4, $t5          #add $t4 and $t5, store result in $t6

#print message for the result
    li $v0, 4                  #load syscall print_string into $v0
    la $a0, msg3               #load address of data msg3 into $a0
    syscall                    #make the syscall

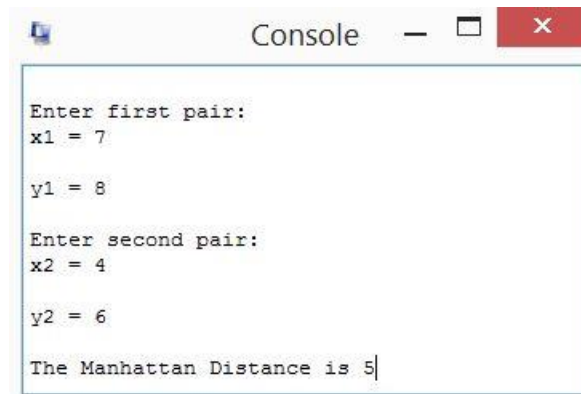
#print the result
    li $v0, 1                  #load syscall print_int into $v0
    move $a0, $t6              #move $t6 to $a0
    syscall                    #make the syscall

#terminate the program
    li $v0, 10                 #load syscall exit into $v0
    syscall                    #make the syscall

#declare and initialize strings for printing the messages
.data
msg1: .asciiz "\nEnter first pair:"
x1: .asciiz "\nx1 = "
y1: .asciiz "\ny1 = "
msg2: .asciiz "\nEnter second pair:"
x2: .asciiz "\nx2 = "
y2: .asciiz "\ny2 = "
msg3: .asciiz "\nThe Manhattan Distance is "

```


Console Window



```
Enter first pair:
x1 = 7
y1 = 8
Enter second pair:
x2 = 4
y2 = 6
The Manhattan Distance is 5|
```

Assignment 6

```
#need to initialize the count to 0
    addi $v0, $0, 0           #initialize $v0 to 0

loop:
    lw $v1, 0($a0)            #read the next word
    sw $v1, 0($a1)            #store the word to the destination

    addi $v0, $v0, 1          #increment the counter

#to point to the next word the pointer must be added 4 and not 1
    addi $a0, $a0, 4           #move the pointer to the next word
    addi $a1, $a1, 4           #move the pointer to the next word

    bne $v1, $zero, loop

#the zero word should not be counted in the counter so at the
#end of the loop decrement the counter by 1 because it will
#have counted the word 0 in the count also as part of the loop
    sub $v0, $v0, 1           #decrement the counter
```

Assignment 7

The main function of the program prompts the user for a number – runs a loop that divides the number by 2 – checks the remainder (when a number is divided the remainder gets stored into HI register while the quotient gets stored into the LO register) – if the remainder is zero jump to ADD function, add the number to the result, and decrement the original number by 1 – repeat the loop again – if the remainder is not zero then decrement the original number and repeat the loop again till the number reaches 0 – at the end print answer and terminate the program.

If the user inputs a much larger number then an error would occur, there would be an overflow, because, the register to store the number is 32 bits. A register can store $2^{32} - 1 = 4,294,967,295$ combinations (numbers), but, because of the signed bits half of those combination or numbers will be positive and the other half will be negative numbers. So, a register can store a maximum number which in particular is 2,147,483,647.

In the given program where it adds the even numbers from 0 to the number entered by the user – the overflow would occur after the number 92,681, because, if the even numbers are added from 0 to 92,681 it will result in 2,147,441,940 which is less than the maximum number a register can store. So, any number above that will result in an overflow.

Formula to find the sum of even numbers is given by,

$$N(N + 1)$$

Where N is found by,

$$N = \frac{\text{first even} + \text{last even}}{2} - 1$$

$$N = \frac{2 + 92680}{2} - 1 = 46340$$

Hence,

$$N(N + 1) = 46340 * 46341 = 2,147,441,940$$

Assignment-8

```
.data 0x10000000
    ask: .asciiz "\nEnter a number between 0 and 50000: "
    ans1: .asciiz "\nThe sum of number entered: "
    ans2: .asciiz "\nThe sum of all the odd numbers in the number entered: "

.text 0x00400000
.globl main
main:

#prompt user for number
    li $v0, 4          #load syscall print_string into $v0
    la $a0, ask         #load address of data ask into $a0
    syscall            #make the syscall

#read number from user
    li $v0, 5          #load syscall print_int into $v0
    syscall            #make the syscall

#store the number entered by user in $t0 and $t6
    move $t0, $v0
    move $t6, $v0

#initializing registers to 0 for increment and storing results
    addi $t1, $0, 0     #initialize $t1 to 0
    addi $t2, $0, 0     #initialize $t2 to 0
    addi $t4, $0, 0     #initialize $t4 to 0
    addi $t5, $0, 0     #initialize $t5 to 0

    li $t3, 2          #initialize $t3 to 2

#beginning of loop
loop:
    beq $t0, 0, SUM     #if $t0 equal 0 jump to SUM
    div $t0, $t3         #divide $t0 by $t3
    mfhi $t1            #move remainder from HI to $t1
    bne $t1, $0, ADD     #if remainder not equal to zero jump to ADD
    sub $t0, $t0, 1     #decrement $t0
    j loop              #jump to loop

#calculate the sum of odd number from 0 to the number entered
```

```

ADD:
    add $t2, $t2, $t0    #add $t2 with $t0 and store result in $t2
    sub $t0, $t0, 1      #decrement $t0
    j loop               #jump to loop

#calculate the sum of numbers from 0 to the number entered
SUM:
    blt $t6, 0, END      #if $t6 = 0, jump to END
    add $t5, $t5, $t4     #add $t5 with $t4 and store result in $t5
    add $t4, $t4, 1       #increment $t4
    sub $t6, $t6, 1       #decrement $t6
    j SUM                 #jump to SUM

#print the results and terminate the program
END:
    li $v0, 4             #load syscall print_string into $v0
    la $a0, ans1           #load address of data ans1 into $a0
    syscall               #make the syscall

    move $a0, $t5          #move $t5 to $a0

    li $v0, 1             #load syscall print_int into $v0
    syscall               #make the syscall

    li $v0, 4             #load syscall print_string into $v0
    la $a0, ans2           #load address of data ans2 into $a0
    syscall               #make the syscall

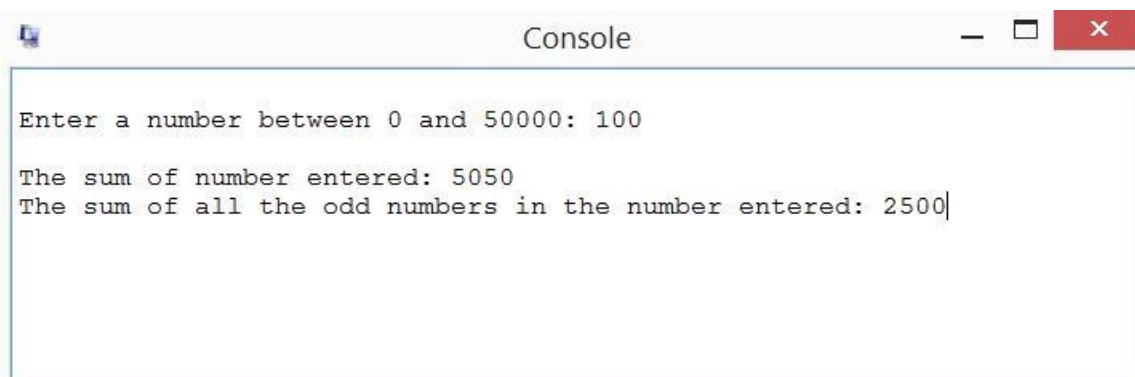
    move $a0, $t2          #move $t2 to $a0

    li $v0, 1             #load syscall print_int into $v0
    syscall               #make the syscall

    li $v0, 10            #load syscall exit into $v0
    syscall               #make the syscall

```

Console Window



```

Enter a number between 0 and 50000: 100

The sum of number entered: 5050
The sum of all the odd numbers in the number entered: 2500|

```


Conclusion

The basic operations of assembly language were practiced. Simple assembly programs were written in a text editor (notepad++) and the program was loaded and executed by using QtSpim – the simulation tool. Step by step execution of code was used to observe and understand how a computer executes assembly and machine language instructions – to understand the use and behavior of different registers in a computer e.g. how to load or store words, to perform arithmetic operations, to read and print integers and strings, to use conditional instructions, and to use iterations.