# Our First Few Simple MIPS Assembly Programs

## Exercise 0

For this exercise, you will download the MARS assembler, and run a few simple MIPS assembly programs from the course website.

First, download MARS (link on course website). Launch it by double-clicking the .jar file. Familiarize yourself with the menu.

Configure it as follows: *Settings->Memory Configuration->Compact, Data at Address 0*. This setting tells the assembler where it can expect to place data and code in memory.

Download three assembly files from the course website (under Lecture 7): [Sum.asm](http://www.cs.unc.edu/~montek/teaching/Comp411-Fall12/Home/Home_files/Sum.asm), [SumArray.asm](http://www.cs.unc.edu/~montek/teaching/Comp411-Fall12/Home/Home_files/SumArray.asm) and [Fibonacci1.asm](http://www.cs.unc.edu/~montek/teaching/Comp411-Fall12/Home/Home_files/Fibonacci1.asm).

Use the *File->Open* menu to open one of these programs. Assemble (i.e., compile) the program by hitting *Run->Assemble*, or by hitting the screwdriver/wrench icon, or by pressing F3. Run the program by hitting *Run->Go*, or by hitting the icon with the play button. You may single-step through the program by hitting *Run->Step*, or hitting the icon with the '1' to the right of the play button. You can also set one or more breakpoints by checking the box to the left of the instructions where you want execution to break, and then hit *Run*.

The bottom pane has two tabs: *Mars Messages* shows errors or warnings during assembly; *Run I/O* is the input/output.

Run each of the three programs. Make sure you understand every single line of code. Here is what to expect for each:

1. **Sum.asm**: Single-step through the program, and observe that the result (i.e., sum of numbers 0..4) will be in register $8 at the end of execution.

Change to sum of numbers 5-10

# Add the first five integers

.text 0x3000

.globl main

main:

add $8,$0,$0 # sum = 0

add $9,$0,$0 # for (i = 0; ...

loop:

addu $8,$8,$9 # sum = sum + i;

addi $9,$9,1 # for (...; ...; i++

slti $10,$9,5 # for (...; i<5;

bne $10,$0,loop

end:

ori $v0, $0, 10 # system call 10 for exit

syscall # we are out of here.

1. **SumArray.asm**: Single-step through it, and observe that the result (sum of 7, 8, 9, 10 and 8) will be in the **sum** variable (at data address 0x0 in the memory). Change the variable **a** to half word, and modify the program. Compare the result, observe memory.

# Add the numbers in an array

.data 0x0

sum: .space 4

i: .space 4

a: .word 7,8,9,10,8

.text 0x3000

.globl main

main:

sw $0, 0($0) # sum = 0;

sw $0, 4($0) # for (i = 0;

lw $9, 4($0) # allocate register for i

lw $8, 0($0) # choose register $8 to hold value for sum

loop:

sll $10, $9, 2 # covert "i" to word offset

lw $10, 8($10) # load a[i]

addu $8, $8, $10 # sum = sum + a[i];

sw $8, 0($0) # update variable in memory

addi $9, $9, 1 # for (...; ...; i++

sw $9, 4($0) # update memory

slti $10, $9, 5 # for (...; i<5;

bne $10, $0, loop

end:

ori $v0, $0, 10 # system call 10 for exit

syscall # we are out of here.

1. **Fibonacci1.asm**:

斐波纳契数列就是，从0和1开始，前面的数加上这一个数，持续下去，就是斐波纳契数列, 具体的斐波纳契数列：1，1，2，3，5，8，13，21，34，55，89，144，233，377，610，987，1597，2584，4181，6765，10946，17711，28657，46368，75025等等.

Single-step through it. This program computes the largest Fibonacci number that is less than 100. Follow the logic of the program and see where the Fibonacci numbers are stored. See where the final answer (largest Fibonacci less than 100) is stored in data memory.

# Find the largest Fibonacci number smaller than 100

.data 0x0

x: .space 4

y: .space 4

.text 0x3000

.globl main

main:

sw $0, x($0) # x = 0;

addi $9, $0, 1 # y = 1;

sw $9, y($0)

lw $8, 0($0)

while: # while (y < 100) {

slti $10, $9, 100

beq $10, $0, endw

add $10, $0, $8 # int t = x;

add $8, $0, $9 # x = y;

sw $8, 0($0)

add $9, $10, $9 # y = t + y;

sw $9, y($0)

beq $0, $0, while # }

endw:

ori $v0, $0, 10 # system call 10 for exit

syscall # we are out of here.

### MARS syscall

There are several library routines provided by MARS that an assembly program can use. These are called *system calls*, or **syscall**. These services include support for printing integers and strings (similar to the **printf()** function in C), reading integers and strings from the keyboard (similar to **scanf()** in C), memory allocation (similar to **malloc()** in C), exiting from a program (similar to return from **main()** in C), etc.

An assembly program accesses those services using the **syscall** command. There is only one **syscall** command for all these services, but which service is requested is determined by the values provided in certain registers. The value in register **$v0** determines which service is requested, and often parameters are passed to the service using registers **$a0**, **$a1**, **$a2** and **$a3**. If a value needs to be returned to the program (e.g., reading an integer from keyboard), it is typically returned in register **$v0**.

For a full listing of system calls available in MARS, please refer to <http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html>. We will mostly be using system calls numbered 1 to 17.

For example, to exit a program, you would use **syscall** with 10 in **$v0**:

ori $v0, $0, 10 #System call code 10 for exit

syscall #exit the program

Note: Sometimes the instruction ori $v0, $0, 10 is shortened to the pseudo instruction li $v0, 10, which stands for ("load the immediate value 10 into $v0"). As another example, to print a string located at location myString, you would use **syscall** with 4 in **$v0**:

li $v0, 4 #System call code 4 for printing a string

ori $a0, $0, myString #address of string to print is in $a0

syscall #print the string

Study all of the system calls from 1 to 17.