CMPEN 331 – Computer Organization and Design
Lab 2
Due Thursday, February 21, 2019 at 11:59 pm (Drop box in Canvas)

The goal is to learn to use the MARS simulator: You can download the MARS program from http://courses.missouristate.edu/KenVollmar/MARS/. This is version 4.5. It's a Java program, so it should run anywhere. We will follow the same homework written by Prof. Heller.

You can check the following tutorial by Prof. Gary Shute
http://www.d.umn.edu/%7Egshute/mips/Mars/Mars.xhtml

To start MARS, just double-click on its icon. You should see a window with four parts - some buttons at the top, a section with tabs Edit and Execute, below that a section with tabs Mars Messages and Run I/O, and a section that displays the processor's Registers. Within the Execute tab, there are panels for the Text Segment (the part of memory where the program's instructions are placed), the Data Segment (the part of memory where the program's global data is placed), and Labels (a symbol table, connecting labels in the program with memory locations). If you decide to resize the MARS window and its various sections, be sure that the Registers panel shows all three columns. If you don't see the Labels panel in the Execute tab, go to the Settings menu, and check the item Show Labels Window.

Let's begin with an absolute minimal program - it starts and ends, and does nothing else. Comments begin with # and go to the end of the line. Assembler directives begin with a dot, and are usually tab-indented. Instructions are usually tab-indented, and labels are at the left margin.

- Start MARS, then click on the New File button (farthest left, or menu File / New). Now you can type the program into the editor pane under the Edit tab. The default file name is mips1.asm. If you were to click on New File again, you would get a second editor pane for the file mips2.asm.
- Don't take a shortcut by using copy-and-paste -- you will miss the helpful information that MARS displays while you type. Copy-and-paste is ok later (well, subject to not cheating, of course).

```
# Version 1, do nothing, then exit

# switch to the Text segment
	.text

	.globl  main
main:
	# the rest of the main program will go here

	# end the program, no explicit return status
	li	$v0, 10
	syscall

# switch to the Data segment
	.data
```

global data will be defined here

- Save the file as version1.asm (fourth button, or menu File / Save as ...). The usual filename for an assembly language program would be like version1.s or version1.asm. On Windows, Visual Studio is usually associated with those filename extensions, and it doesn't like MIPS Assembler, so be careful about double-clicking from a folder display.
- QtSPIM only accepts .s or .asm files. MARS will accept any file name; using something like version1.mips will prevent any bad interactions when double-clicking icons, but it makes using the other simulator more difficult.
- Assemble the program (thirteenth button, first in the third group, looks like a screwdriver and wrench, or menu Run / Assemble). You should see in the Mars Messages panel below the Edit/Execute section. Of course, your directory names will be different.
    - **Note** that you can't assemble the file until it has been saved.
    - After the program has been successfully assembled, the display switches to the Execute tab, with machine code in the Text Segment. The Registers information indicates that the program will start at address 0x00400000, which is the content of register pc, the Program Counter; this register always holds the address of the next instruction to be executed. The Text Segment highlights that instruction.
    - Global variables are displayed in the Data Segment, except that there aren't any yet. The Labels panel shows global names, which will be needed later when your program uses functions and spans several files. For now, it just says that main is at location 0x00400000.
- Run the program (fourteenth button, second in the third group, looks like a rightward triangle, or menu Run / Go). You should see in the *Mars Messages panel*, and in the **Run I/O panel**.
- *Go: running version1.asm*
- *Go: execution completed successfully.*
- **-- program is finished running --**

- You can set breakpoints in the program by checking the Bkpt box at the left of each instruction in the Text Segment panel. More of that later.
- Note that the li instruction (load immediate) turned into an addiu instruction (add immediate unsigned). That's because li is actually a pseudo-instruction. Register $v0 turned into register $2, which is the same thing. Register $0 (or $zero) always contains 0 bits.
- The syscall instruction (system call) initiates an operating system action; which one depends on the value in register $v0 when the instruction is executed. Syscall 10 resembles the exit() function in C or C++, while syscall 17, which we will use later, really is the exit() function.
- Note that the Program Counter register (pc) started as 0x00400000 and ended as 0x00400008, which is one instruction past the last one in the program.
- If you want to rerun the program, you can reset the memory and registers to their default values using the second button from the right, or menu Run / Reset. This returns $v0 to 0x00000000 and pc to 0x00400000.
- If you want to run the program one instruction at a time (single-stepping), then reset the memory and registers, and repeatedly use the Run button with the subscript 1, or menu Run / Step. Pay attention to changes in registers $v0 and pc; $v0 is highlighted as it changes, but pc is not.

Next, we add some function definitions, which make it easier to deal with system calls. Note the second .text directive, so the new code isn't treated as data. The program is getting longer, so we use a comment as a dividing line for the different parts of the program.

```
# Version 2, do nothing, then exit

# switch to the Text segment
	.text

	.globl  main
main:
	# the rest of the main program will go here

	# call function Exit0
	jal     Exit0   # end the program, default return status

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# switch to the Data segment
	.data

	# global data will be defined here

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Wrapper functions around some of the system calls
# See P&H COD, Fig. A.9.1, for the complete list.

# switch to the Text segment
	.text

	.globl  Print_integer
Print_integer:  # print the integer in register $a0
	li      $v0, 1
	syscall
	jr      $ra

	.globl  Print_string
Print_string:   # print the string whose starting address is in register $a0
	li      $v0, 4
	syscall
	jr      $ra

	.globl  Exit
Exit:           # end the program, no explicit return status
	li      $v0, 10
	syscall
	jr      $ra     # this instruction is never executed

	.globl  Exit0
Exit0:          # end the program, default return status
```

```
    li   $a0, 0  # return status 0
    li   $v0, 17
    syscall
    jr   $ra   # this instruction is never executed

    .globl  Exit2
Exit2:      # end the program, with return status from register $a0
    li   $v0, 17
    syscall
    jr   $ra   # this instruction is never executed

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Repeat the previous steps, with suitable changes. It may help to first clear the Mars Messages and Run I/O panels.
- Note that pc finishes with the value 0x00400034, which is the address of the instruction following the syscall instruction in the function Exit0. Work through the Text Segment, Labels and Registers to verify this.

Note also that register $ra has the value 0x00400004, which is the address of the instruction following the jal Exit0 instruction. More about this later. It's essentially a coincidence that the first instruction of the function Print_integer is at address 0x004000004.

Next, the third version, the ubiquitous Hello World program.

```
# Version 3, print something, then exit

# switch to the Text segment
    .text

    .globl  main
main:
    # the main program goes here
    la   $a0, hello_string
    jal  Print_string

    jal  Exit0   # end the program, default return status

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# switch to the Data segment
    .data

    # global data is defined here
hello_string:
    .asciiz "Hello, world\n"

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
# Wrapper functions around some of the system calls
# See P&H COD, Fig. A.9.1, for the complete list.

# switch to the Text segment
	.text

	.globl  Print_integer
Print_integer:  # print the integer in register $a0
	li	$v0, 1
	syscall
	jr	$ra

	.globl  Print_string
Print_string:   # print the string whose starting address is in register $a0
	li	$v0, 4
	syscall
	jr	$ra

	.globl  Exit
Exit:		# end the program, no explicit return status
	li	$v0, 10
	syscall
	jr	$ra	# this instruction is never executed

	.globl  Exit0
Exit0:		# end the program, default return status
	li	$a0, 0  # return status 0
	li	$v0, 17
	syscall
	jr	$ra	# this instruction is never executed

	.globl  Exit2
Exit2:		# end the program, with return status from register $a0
	li	$v0, 17
	syscall
	jr	$ra	# this instruction is never executed

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Repeat the previous steps, with suitable changes. You should see in the *Mars Messages panel*, and in the **Run I/O panel**.

Assemble: operation completed successfully.
Go: running version3.asm
Go: execution completed successfully.
Hello, world

-- program is finished running --

Note that the label hello_string is not marked (global) in the Labels panel. That's because we didn't use the directive .globl hello_string . The label is defined and accessible within the file version3.asm, but it can't be seen outside the file. Try adding the directive, and see how the Labels panel changes.

Finally, here is a program derived from P&H COD, Fig. A.1.4, corrected and adapted for the MARS simulator. There are some differences between MARS and QtSPIM concerning the starting address for main(), and a few other things. This version will work with both simulators. Copy and paste seems completely appropriate at this point.

```
      .text
#      .align  2            # MARS doesn't like this

      .globl  main
main:
      subu    $sp, $sp, 40    # stack push, 40 bytes
      sw      $ra, 20($sp)    # save return address register (sw = store word)
      sd      $a0, 32($sp)    # save registers $a0, $a1 (sd = store doubleword)
      sw      $0, 24($sp)     # sum = 0
      sw      $0, 28($sp)     # i = 0
loop:
      lw      $t6, 28($sp)    # i
      mul     $t7, $t6, $t6   # i * i
      lw      $t8, 24($sp)    # sum
      addu    $t9, $t8, $t7   # sum + i*i
      sw      $t9, 24($sp)    # sum = sum + i*i
      addu    $t0, $t6, 1     # i + 1
      sw      $t0, 28($sp)    # i = i + 1
      ble     $t0, 100, loop  # if (i <= 100) goto loop

      la      $a0, str1
      jal     Print_string    # print the string whose starting address is in register $a0
      lw      $a0, 24($sp)    # sum
      jal     Print_integer   # print the integer in register $a0
      la      $a0, str2
      jal     Print_string    # print the string whose starting address is in register $a0

      # this exit sequence can be used with QtSPIM, but not with MARS
#      move    $v0, $0         # return status 0
#      lw      $ra, 20($sp)    # restore saved return address
#      addu    $sp, $sp, 40    # stack pop (important - same 40 bytes as before!)
#      jr      $ra             # return from main() to the OS

      # MARS likes this, but we need to demo Exit2
#      jal     Exit            # end the program, no explicit return status
```

```
	# MARS likes this
	move	$a0, $0
	jal	Exit2		# end the program, with return status from register a0

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

	.data
	.align  0

str1:
	.asciiz "The sum from 0 .. 100 is :"
str2:
	.asciiz ":\n"

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Wrapper functions around some of the system calls
# See P&H COD, Fig. A.9.1, for the complete list.

	.text
#	.align  2		# MARS doesn't like this

	.globl  Print_integer
Print_integer:  # print the integer in register $a0
	li	$v0, 1
	syscall
	jr	$ra

	.globl  Print_string
Print_string:   # print the string whose starting address is in register $a0
	li	$v0, 4
	syscall
	jr	$ra

	.globl  Exit
Exit:		# end the program, no explicit return status
	li	$v0, 10
	syscall
	jr	$ra	# this instruction is never executed

	.globl  Exit0
Exit0:		# end the program, default return status
	li	$a0, 0  # return status 0
	li	$v0, 17
	syscall
	jr	$ra	# this instruction is never executed
```

```
        .globl  Exit2
Exit2:          # end the program, with return status from register $a0
        li      $v0, 17
        syscall
        jr      $ra     # this instruction is never executed

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- QtSPIM treats main() as a function that is called by the operating system, which is indeed the case on Unix-like operating systems. MARS treats main() as a function that is called (apparently) from nowhere, since $ra is initially 0.

Now we need to verify that you actually can use MARS successfully.

- When you run the previous program, what is printed?
- What is the value in register $t7 (in decimal) when the program ends?
- Set a breakpoint for the instruction at line 13 of the assembler source code. Run the program again; it should stop at the breakpoint. Now execute that one instruction. Which registers have changed as a result of executing that one instruction? You might need to continue past the breakpoint several times to see what's going on. Note that P&H COD Appendix A.10 has descriptions of all the instructions, but you can't just look up the answer. (You should look up the instructions in App. A.10, but the answer requires you to pull together several different pieces of information, not just one.)

Put the answers to these three questions in your Lab. You have to upload the source codes used as well, all on one zip file.