CDA 4101
Computer Assignment # 1

*Please type each problem neatly with clear labels and an explanation showing your work. Your **name**, **course no** and the **homework number** should be clearly given on the top right hand side of the first page at a minimum and, ideally, all pages. Submit the PDF of your solution in Canvas by the deadline.*

Submission Details: For each assignment there will be a report. This report will consist of a text that follows the report guideline, your code file, and screenshots of your working code. These files should be placed together in a single PDF file that needs to be submitted in Canvas.

Honor Code: Honor Code This assignment should be completed individually to maximize learning. It is important that you adhere to the Course Policies, particularly the section on Programming Assignments. Also this assignment should follow FIU Honor Code Policy. Authorized help for all assignments are limited to the lab handouts, the LAs, and the Professor. Copying work from another student or the Internet is an honor code violation as is giving help to another student, which will result in a zero on the assignment and possibly further sanctions. Your code may be subject to evaluation by MOSS (Measure of Software Similarity), which is used to detect inappropriate similarities among programs.

Pre-Knowledge: In order to complete this lab you will need an understanding of the MIPS instruction set and some knowledge of assembly. Before you start this assignment, you may read parts of Appendix B. Note: if you are having a hard time finding out what syscall does, see its definition in Zybook 5.7.

Objectives: This first computer assignment is partially a tutorial. In this assignment, the goal is to get the feel for the process a computer goes through when running a program. In addition, I want you to understand how to write a small machine code program and step through its execution as a computer would. We will be using the machine described in the text and the Spim simulator that allows you to read in a text file of assembly code and run assembly code as if it were on an actual MIPS machine.

It will walk you through several simple MIPS programs and then turn you loose to write some of your own code. We will use QtSpim as the simulator for running your assembly code. Learning objectives of this assignment are:

- Become familiar with QtSpim
- Assemble and run MIPS programs
- Debug MIPS programs
- Write simple MIPS programs
- Have a basic understanding of the processor will decode and execute each instruction
- Determine the final result of a simple assembly language program on the state of the machine in the simulator

Required Tools: In this assignment, we will be using a MIPS simulation tool called SPIM. SPIM can be downloaded from this link: https://sourceforge.net/projects/spimsimulator/files/ o ChoosetheQTSpimoptionforwhateveroperatingsystemyoumayberunning

Teamwork: This computer assignment can be done in a TEAM of 2 STUDENTS. You may work with another student, but I expect you to turn in separate reports and be able to reproduce the work on your own later, if required.

ASM Programming Structure: Assembly code conforms to the following structure:

```
.data # variable declarations follow this line
    # Format would be like "label:"
    # storage_type value(s)
.text # instructions follow this line
main: # indicates start of code (first instruction to execute)
```

The data portion of the code is where variables and other values can be stored. The text portion is where the machine instructions go. Functions in the text portions have labels so that a user can easily jump or branch to that instruction line. It is also worth noting that the # symbol causes any text after that symbol to be commented out.

MIPS Register Set: A table describing registers available in the MIPS processor is as follows:

Register Number	Alternative Name	Description
0	zero	the value 0
1	$at	(**a**ssembler **t**emporary) reserved by the assembler
2-3	$v0 - $v1	(**v**alues) from expression evaluation and function results
4-7	$a0 - $a3	(**a**rguments) First four parameters for subroutine. Not preserved across procedure calls
8-15	$t0 - $t7	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls
16-23	$s0 - $s7	(**s**aved values) - Callee saved. A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls
24-25	$t8 - $t9	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls.
26-27	$k0 - $k1	reserved for use by the interrupt/trap handler
28	$gp	**g**lobal **p**ointer. Points to the middle of the 64K block of memory in the static data segment.
29	$sp	**s**tack **p**ointer Points to last location on the stack.
30	$s8/$fp	**s**aved value / **f**rame **p**ointer Preserved across procedure calls
31	$ra	**r**eturn **a**ddress

The registers that we frequently use are the values, arguments, and temporaries registers. These registers can be referenced either by their number or letter. So, register 2 would be $2 or $v0.

- The 0 register, which is always 0.
- The values registers are used for storing returned values from a function as well as selecting which system call to use.
- The arguments registers are used to pass in arguments to a function that the system is calling

- The Temporaries registers are used to store temporary data for calculations
    - These registers are Caller Saved, this means that if one function wishes to call another function and guarantee that the values in these registers are untouched, that function must save these to memory. Thus the function caller must save the values
    - This is opposed to Callee Saved registers which, if used, must first be saved by the calling function. Thus the Callee must save the values.

PART 1. Getting Started

Step 1. First install QtSpim software

- MAC users see this installation video: https://www.youtube.com/watch?v=sn0zDYdLyac
- Windows users see this video: https://www.youtube.com/watch?v=PxJtwXFnQQk
- You can learn more about SPIM in Zybook Appendix A (7.9).

Step 2. Setup your assembly files

Download the .asm files from Canvas onto the desktop or the folder where QtSpim lives.

Step 3. Open QtSpim

Open QtSpim. You should see a QtSpim window and a Console window.

Step 4. Choose settings

- Open the Settings dialog box using Simulator Settings on the menu bar.
- Select the "allow pseudo instruction" checkbox.

PART 2. AN ASM EXAMPLE

ASM Programming instructions: For this part your goal is to run some example assembly code (provided below), and then modify the code to read in and add 3 numbers instead of just 2.

```
1.  .data # variable declarations follow this line
2.  .text # instructions follow this line
3.  main:
4.  ## Code Part 1: Get first number from user, put into $t0.
5.  ori $v0, $0, 5 # OUR CODE BEGINS HERE: load syscall read_int into $v0.
6.  syscall # make the syscall.
7.  addu $t0, $0, $v0 # move the number read into $t0.
8.  ## Get second number from user, put into $t1.
9.  ori $v0, $0, 5 # load syscall read_int into $v0.
10. syscall # make the syscall.
11. addu $t1, $0, $v0 # move the number read into $t1.
12. add $t2, $t0, $t1 # compute the sum.
13. ## Print out $t2.
14. addu $a0, $0, $t2 # move the number to print into $a0.
15. ori $v0, $0, 1 # load syscall print_int into $v0.
16. syscall # make the syscall.
17. ori $v0, $0, 10 # syscall code 10 is for exit.
18. syscall # make the syscall.
19. ## end of add2.asm.
```

A discussion of each line of the code is as follows:

The first two lines declare the .data and the .text segments of our code. There is nothing after the .data segment because we have no variables to store. All of our instructions follow the .text segment.

Next comes the declaration of the main: label in line 3 which is required for any code we wish to run. Lines 4-7 read in and store a value. To read a value we must make a system call (or syscall), and tell our processor we need input. Each syscall (there are many types) has an associated number. In order

to make a syscall we put the call's associated number in $v0 and then execute the syscall instruction. In our case the associated number for reading in a numerical value is 5. Once a syscall finishes its return values are stored in $v0 and/or $v1.

Lines 8-11 repeat the reading process but store the value in a different register. Line 12 calculates the sum of the two values we read in.

Lines 13-16 handle printing out the computed sum. In this case we are using the print_int syscall whose number is 1. This syscall prints out whatever value is stored in the argument register $a0.

Finally we must exit the main method. Lines 17 and 18 accomplish this by using the exit syscall.

Question 1 [10 pt]. Modify the above piece of code so it can add 3 numbers. Include your code in the report.

Question 2 [10 pt]. What modifications were needed to add the 3 numbers?

PART 3: MIPS Arithmetic

Step 1. Open and run the assembly file

- Open the file that you downloaded called ALU2015.asm with your favorite text editor (notepad, etc)

- Open the assembly file in QtSpim using the File Open icon (an open folder).

- Run the assembly file by clicking on Simulator Go. The program should start at the first instruction in memory (0x00400000). You may have to change the address to match your program start.

- *Hint:* for debugging, you may find it easier to single step through the program. To do this, first reload it and reinitialize everything. Then, set the PC (use Simulator Set Value; PC; 0x00400000). Then Single Step (F10) and observe how each instruction affects the machine state. Again, you may have to reset the address to start at your first instruction.

Step 2. Answer the following problems

Question 3 [15 pt]. Look at code provided in ALU2015.asm. Write down what it does.

Question 4 [15 pt]. What is the result and where is the result stored? Explain how it gets there.

Question 5 [10 pt]. Why are they using the ori instruction?

Question 6 [10 pt]. Why a complement operation is used in this code?

PART 4. Misc. MIPS

In addition to the basic instructions that you have learned so far, MIPS supports several other things to help you program. This section of the assignment will help you to explore pseudo-instructions, assembler directives, and system calls.

Pseudo-instruction means "fake instruction". These are instructions that are useful to the programmer, but they do not actually exist in the instruction set that the machine understands. Instead, the assembler converts the pseudo-instructions into actual executable instructions. This not only makes it easier to program, but can also add clarity to the program.

Assembler directives help the assembler to understand code. You read about these in Zybook Appendix A (7.2). They start with a '.' and convey information to the assembler.

System calls (Zybook ch 5.7 and Appendix A-Fig 7.9.1) are simple operating-system-like services that MIPS provides. To run a system call, set up the appropriate registers (usually $a0- $a2), store the system call number in $v0, and use "**syscall"** in your assembly program.

Step 1. Open and run the assembly file

- Open the file that you downloaded called MIPSMisc.asm in your favorite text editor(notepad, etc).

- Open the assembly file in QtSpim

Step 2. Answer the following problems

Question 7 [15 pt]. What does this program do? What value does the register s1 hold once you run this program? What does s1 represent?

Writing Assembly

Now it is your turn to write some assembly code! Make sure to include appropriate comments in your code. (Remember appropriate comments do not tell me what individual instructions do, but rather explain code at a higher level.)

Question 8[15 pt]. Your job is to implement the following pseudo-code in assembly. You may use pseudo-instructions as needed.

Algorithm 2.1 The multiples program.

---

1. Get A from the user.
2. Get B from the user. If $B \leq 0$, terminate.
3. Set sentinel value $S = A \times B$.
4. Set multiple $m = A$.
5. Loop:
   (a) Print m.
   (b) If $m == S$, then go to the next step.
   (c) Otherwise, set $m = m + A$, and then repeat the loop.
6. Terminate.
