Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE2322: Fundamentals of Digital Design and Computer Organization

Summer II 2022 – Homework 4

Submission Instructions

- For the programming problems (MIPS assembly and Verilog HDL):
    1. Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
    2. At the beginning of your source code files write your full name, students ID, and any special assembling/running instruction (if any).
    3. If required, test your code using the MARS assembler or the Vivado Simulator.

- For non-programming problems, include explanation of all steps of your answers/calculations not only the final results.

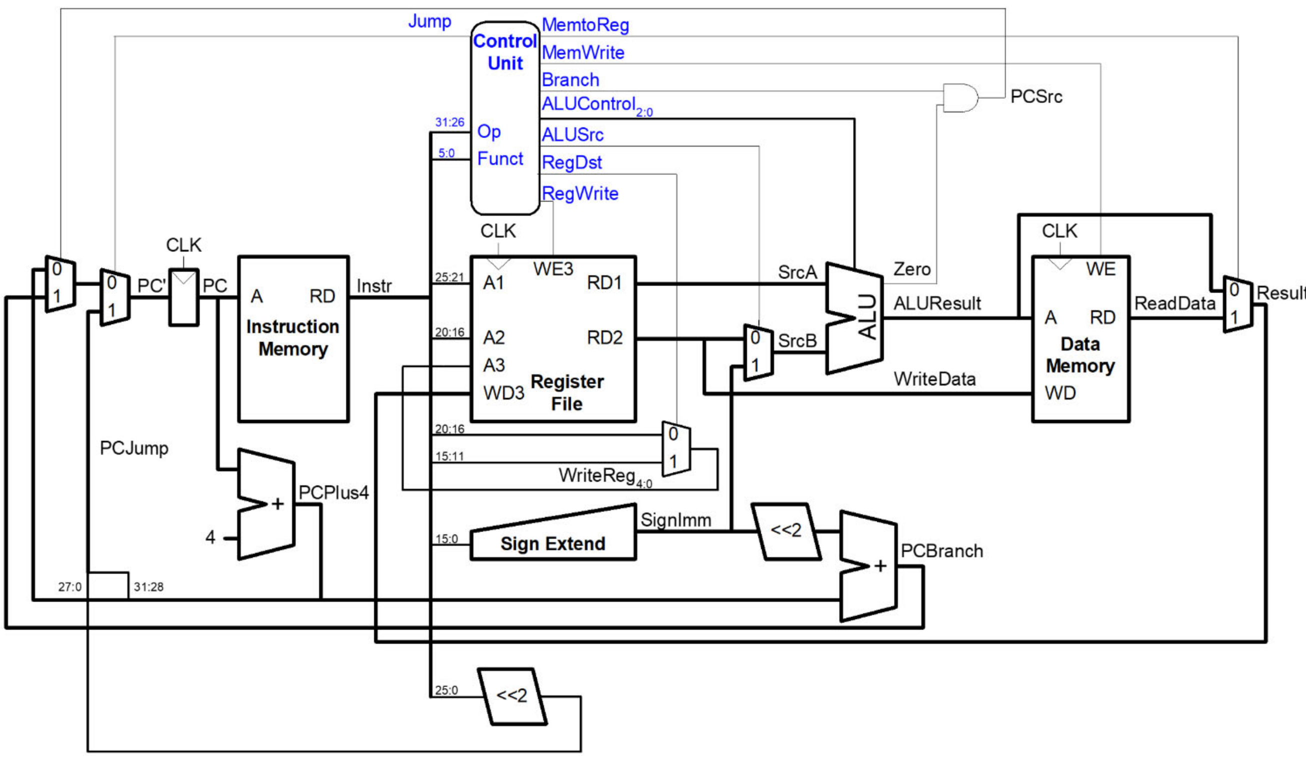
- Submit the following to the homework assignment page on Canvas:
    1. Your homework report developed by a word processor and submitted as one PDF file. For answers that require drawing and if it is difficult on you to use a drawing application, you can neatly hand draw the answer, scan it, and include it into your report. The report includes the following (depending on the assignment contents):
        a. Answers to the non-programming problems that show all the details of the steps you follow to reach these answers.
        b. A summary of your approach to solve the programming problems.
        c. If required, the screen shots of the sample runs of your programs.

    2. Your well-commented programs source code files (i.e., the .asm or .v files). Do not upload any whole project folders/files that are usually created by Vivado.

    Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

Note: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded (i.e., all required reports and files must be included in this last attempt).

Q1 (15 Points)

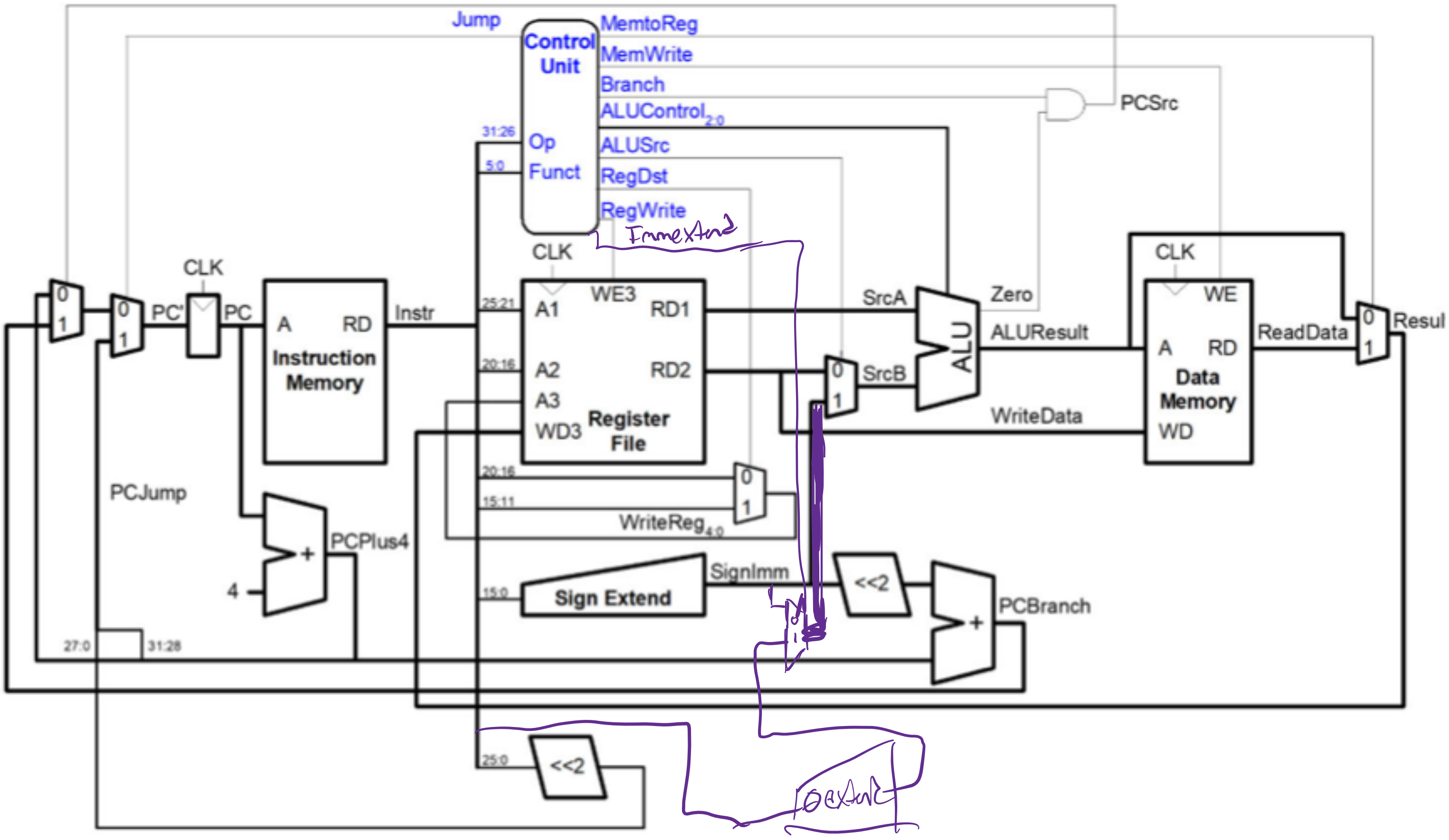


Extend the above single-cycle datapath and control to support for instruction **`ori`**. The **`ori`** instruction requires zero extension of the imm value instead of the current sign extension. You need to add a new control signal named **`ImmExtend`** that determines how the extension of the field imm is handled by different MIPS instructions. Each possible value for the new control signal has the following effects:

0 – Perform a sign extension of field imm. This is the current behavior of the above datapath.
1 – Perform a zero extension of field imm.

i) Draw the extended block diagram, including the new proposed control signal.
ii) Extend the *"Control Unit - Instruction Decoder 2"* table in the lecture slides to reflect the support for the new **`ori`** instruction. Add to the table a new column for the new **`ImmExtend`** signal.

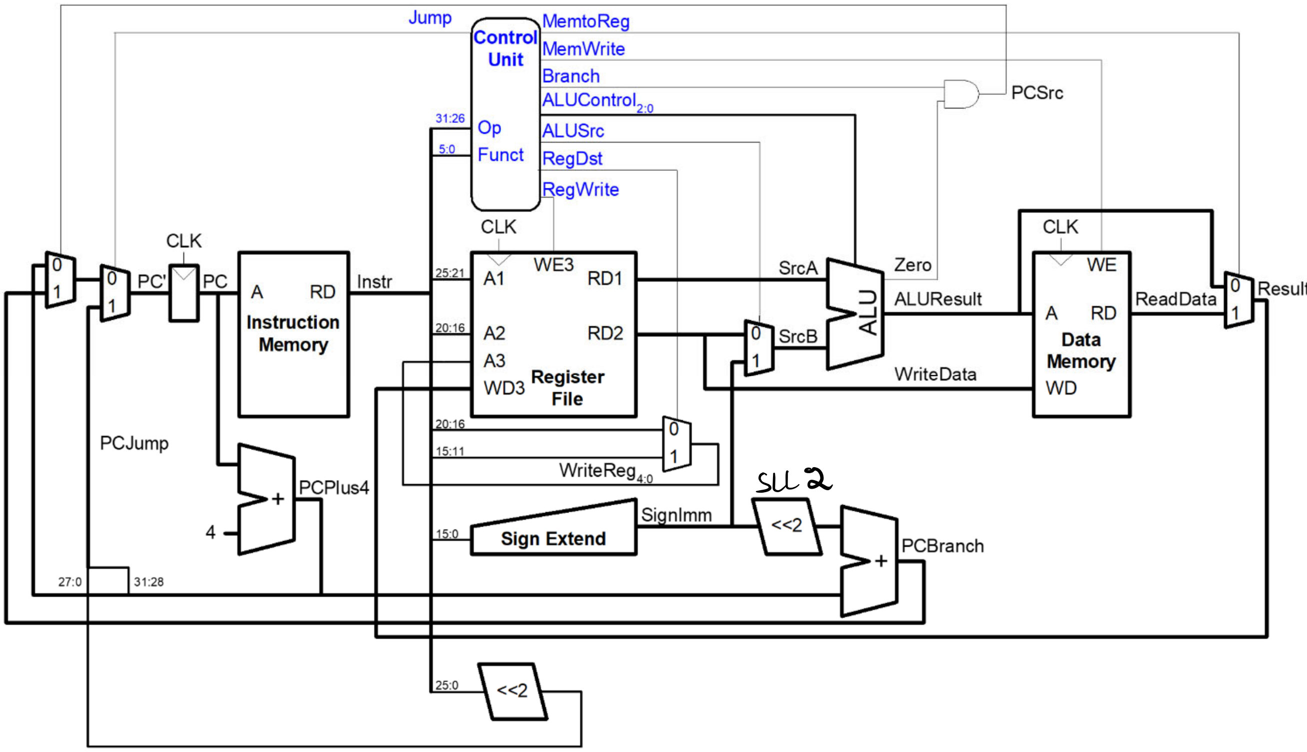
Slide 34



No need to Fully Put table

	opcode	ImmExtend
R	000000	X
lw	100011	0
sw	101011	0
beq	000100	X
addi	001000	0
j	000010	X
ori	001101	1

Q2 (15 Points)



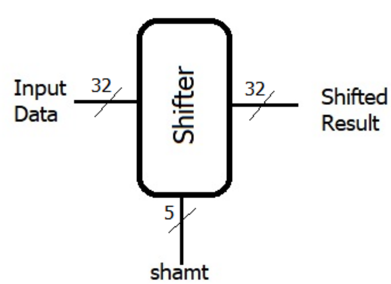
Extend the above single-cycle datapath and control by adding the shifter shown below to implement the **sll** instruction. This shifter is a separate circuit from the ALU. You will need to add a new control signal named **Shift**. The **sll** instruction is an R-Type instruction with the following format for instruction example

sll $s1, $s0, 7

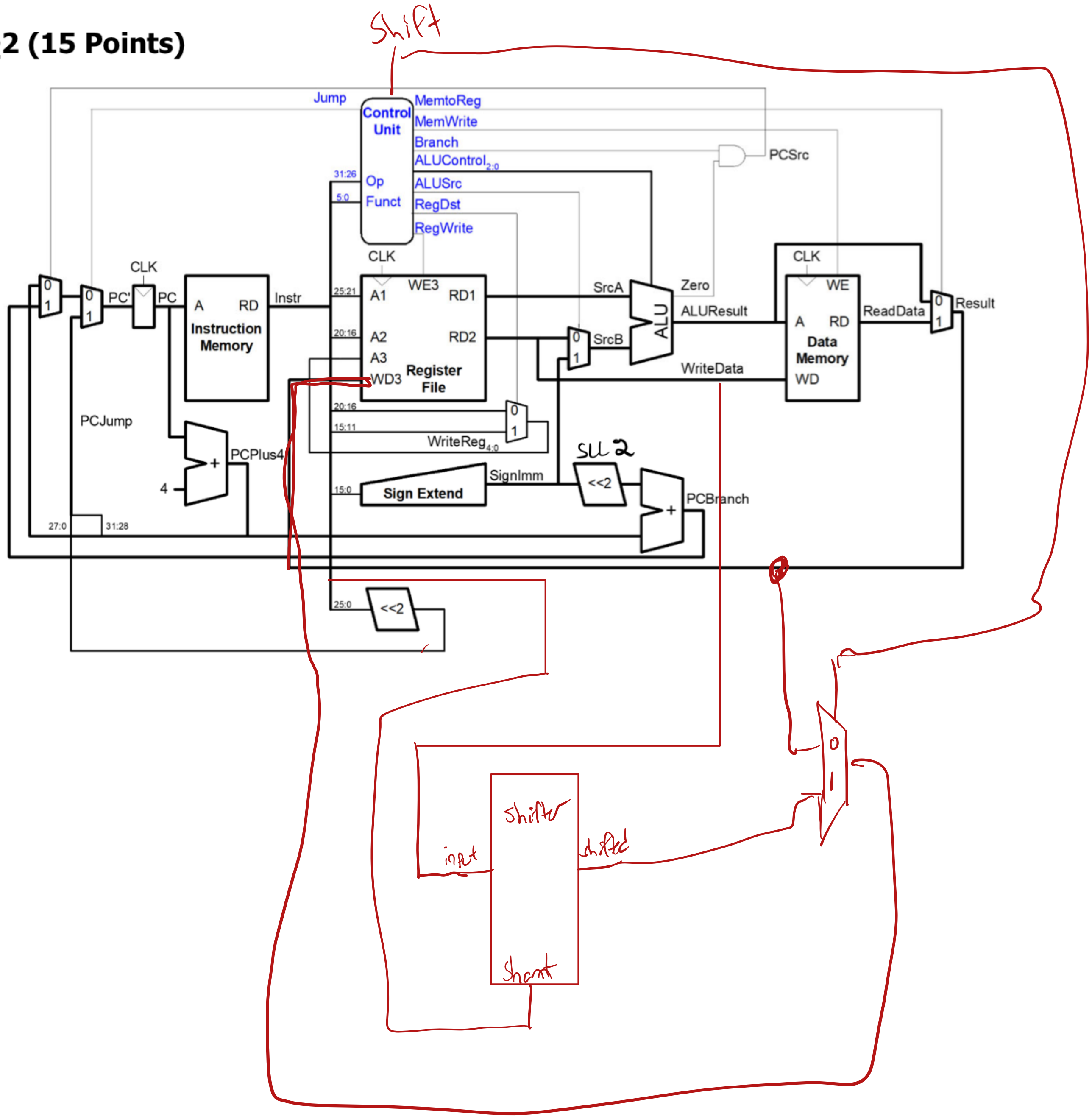
OpCode	Rs	Rt	Rd	shamt	function
000000	00000	$s0 code	$s1 code	00111	000000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

The instruction **sll $s1, $s0, 7** shifts the value in the Rt register (**$s0**) left by 7 bits and stores that value in the Rd register (**$s1**). The figure shows the shifter to be used.

Draw the extended block diagram the includes the new proposed control signal, the shifter circuit, and any additional circuits if needed. No need to provide the extended "Control Unit Instruction Decoder" table.



Q2 (15 Points)



Shift
Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
31:26
Op
5:0
Funct
PCSrc
CLK
WE3
A1
RD1
A2
RD2
A3
WD3
Register File
25:21
20:16
SrcA
SrcB
Zero
ALU
ALUResult
WriteData
CLK
WE
A
RD
Data Memory
WD
ReadData
Result
CLK
PC'
PC
A
RD
Instruction Memory
Instr
20:16
15:11
WriteReg4:0
PCJump
PCPlus4
4
15:0
Sign Extend
SignImm
SLL 2
<<2
PCBranch
27:0
31:28
25:0
<<2
shifter
input
shifted
shamt
0
1

Q3 (15 Points)

Assume the elements in the MIPS single-cycle datapath and control have the following delays in picoseconds (ignore the delay time through the Control Unit).

Element	Parameter	Delay (ps)
PC clock-to-Q	t_{pcq_PC}	30
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Data and Instruction Memory (read or write)	t_{mem}	250
Register file (read or write)	t_{RF}	150
Plus 4 Adder	t_{add4}	70
Shift-left 2	t_{shift}	10
Sign-extend	t_{sext}	15

For each of the following two instructions, what is the total time in *ps* needed from the moment the clock edge triggers the PC to read the instruction address until the data is ready to be written to its destination (register file or data memory) right before the next clock edge? Show all steps of your calculations.

<u>Note</u>: This is not the T_c as in the lecture and hence no need to consider the setup time of the register file or the data memory.

```
  i.   add $t1, $t2, $t3
 ii.   sw $t0, 32($s3)
```

$t_{add} = 30 + 200 + 25 + 250 + 150 + 25 + 150$

PCLQ + ALU tmux + memory + RF + mux + RF

$\boxed{830 \text{ For line 1}}$

$t_{sw} = 30 + 250 + 250 + 150 + 25 + 200$

PCLQ IDIM DIM RF mux ALU

910ps 2nd line

Slides 95, 102ish, 105

Q4 (15 Points)

Write a MIPS program that asks the user to input string. The program then removes all space characters from the string. The program finally displays the resulting string without spaces. The following C code shows the proposed algorithm for the program. The string is traversed with two indices, called `old_index` and `new_index`, where the latter always takes a value less or equal to the former. When a non-space character is found, the character at position `old_index` is copied to position `new_index`, and both indices are incremented. When a space is found, the current character is not copied, and only `old_index` is incremented. The algorithm stops when a null character is found.

```
#include <stdio.h>
int main() {
  // Read string
   char s[100];
   printf("Enter string: ");
   gets(s);
 // Remove spaces
    char c;
    int old_index = 0;
    int new_index = 0;
    do {
    // Read character
     c = s[old_index];
    // Old position moves ahead
     old_index++;
    // If it's a space, ignore
     if (c == ' ') continue;
    // Copy character
     s[new_index] = c;
    // New position moves ahead
     new_index++;
    } while (c);

  // Print result
    printf("New string: %s\n", s);
}
```

askusprompt?

```
.data
    .asciiz "Enter a string: _"
    .asciiz "new string:"
S1: .space 100
S2: .space 100
.text
main: li $v0,4
    la $a0, S1   #input of string
    li $a1, 100
    Syscall
    move $s0, $a0 #move data
    move $s1, $a0 #new
loop: lb $a0, 0($s0) # first letter into a0
    addi $s0, $s0, 1   # increment index by 1
    beq $a0, 32, loop  # if not empty continue
    beq $a0, $0, done  # else full, stop
    sb $a0, $s1   # update index to=c
    add $s1, $s1, 1   # increase new index by 1
    J loop
Done:
    li $t7
    sb $t7, $s1   #terminating old
```

i. Write the full well-commented MIPS program and submit it in a file named nospace.asm.

ii. Run your program on MARS and include in your homework report screen shots of the program sample runs. Make sure to enter a string containing spaces to demonstrate the correct functionality of the algorithm.

Q5 (20 Points)

Write a MIPS assembly language program that calls a procedure "*Occurrences*" to count the number of occurrences ***F*** of a given integer ***X*** in an array ***A*** of integers. The size of array ***A*** is stored in ***N***.

F and ***N*** are 8-bit unsigned integers, while ***A***'s contents and ***X*** are 16-bit unsigned integers. The *Occurrences* procedure receives three input parameters: the address of ***A***, the value of ***X***, and the value of ***N***. The procedure returns the count ***F***. In the main program, you need to store the returned value ***F*** in the data memory segment. Also, the contents of array ***A*** and its size ***N*** are defined in the memory data segment (i.e., are not received as inputs form the user). No need to store ***X*** in the data segment as it must be received as an input from the user during run time. Prompt the user for this value with a meaningful message.

iii. Write the full well-commented MIPS program and submit it in a file named `hw5Q1.asm`
iv. Run your program on MARS and include in your report screen shots of the program sample runs. Make sure to test your program with at least three array examples and one example with ***A*** has zero occurrences of the ***X***.

Q6 (20 Points)

Write a MIPS assembly language program that calls a procedure "*Reverse*" to display the contents of an array of integers ***A*** in reverse order. ***N*** is the number of integers in ***A***. All values in this program are stored in words of 4 bytes each. The procedure receives two input parameters: the address of ***A*** and the size of the array ***N***. In the main program the contents of array ***A*** and its size ***N*** are defined in the memory data segment.

Procedure "*Reverse*" must use a <u>recursive algorithm</u> to display the array in reverse order. This can be done by the procedure calling itself and passing as a first parameter the same array without its first element (i.e., starting from the second element of the array) and as a second parameter the new array size (i.e., ***N*** -1). After calling itself recursively, the procedure prints that first element.

i. Write the full well-commented MIPS program and submit it in a file named `hw5Q2.asm`
ii. Run your program on MARS and include in your report screen shots of the program sample runs. Make sure to test your program with at least three array examples with different sizes.