#Use the MARS simulator to write and execute MIPS assembly code that will perform

#the high level language pseudocode assignment statements.

#Use this file as a base and perform each exercise in order according to the

#descriptions given in each comment.

#No pseudo instructions are allowed, use only the following basic MIPS instructions:

# add - arithmetic add

# sub - arithmetic subtract

# and - bitwise and

# or - bitwise or

# slt - set on less than

# nor - bitwise nor

# addi - add with immediate data

#

#YOUR SUBMISSION WILL BE ASSIGNED A SCORE OF ZERO IF IT CONTAINS:

# PSEUDOINSTRUCTIONS

# SYNTAX ERRORS

# INSTRUCTIONS NOT LISTED ABOVE

########################################

#########################################

## Example of how to perform exercises ##

#########################################

#Variables A through C have the following register mapping:

# A:$16 B:$17 C:$18

#For temporary storage use registers $13, $14, or $15 as needed for complex operations.

#Example Exercise:

# Initial Variable Values: A = 50; B = 100; C = 150;

# Assignment statement to perform: C = A + B - C;

addi $16, $0, 50 #A = 50;

addi $17, $0, 100 #B = 100;

addi $18, $0, 150 #C = 150;

add $13, $16, $17 #Temp = A + B;

sub $18, $13, $18 #C = A + B - C;

#########################################

## END OF EXAMPLE SECTION ##

#########################################

########################################

########################################

#Now for your assignment, complete the following exercises in order as they appear.

#The result of each exercise can depend on the result of previous exercises.

#Variables A through F have the following register mapping:

# A:$10 B:$11 C:$12 D:$13 E:$14 F:$15 G:$16 H:$17 I:$18

#For temporary storage use register $1 through $9 as needed for complex operations.

#Exercise 1: Complex Assignment statement

# Initial Variable Values: G = 10000; C = 12000; D = 5678;

# Assignment statement to perform: H = G+(C-D)

addi $16, $0, 10000

addi $12, $0, 12000

addi $13, $0, 5678

sub $1, $12, $13

add $17, $16, $1

#Exercise 2: Sum of differences

# Initial Variable Values: C = 2000; D = 1250; E = 5000; F = 3500;

# Assignment statement to perform: I = (C-D)+(E-F)

addi $12, $0, 2000

addi $13, $0, 1250

addi $14, $0, 5000

addi $15, $0, 3500

sub $1, $12, $13

sub $2, $14, $15

add $18, $1, $2

#Exercise 3: Add without adding

# Initial Variable Values: A = 25; B = 50

# Assignment statement to perform: G = A - (~B + 1);

addi $10, $0, 25

addi $11, $0, 50

nor $1, $0, $11

addi $2, $1, 1

sub $16, $10, $2

#Exercise 4: Subtract without Subtracting

# Initial Variable Values: C = 1000; D = 750;

# Assignment statement to perform: F = C + (~D + 1)

addi $13, $0, 1000

addi $14, $0, 750

nor $1, $0, $14

addi $2, $1, 1

add $15, $2, $13

#Exercise 5: Less Than Comparison

# Initial Variable Values: A = 20000; B = 30000

# Assignment statement to perform: E = (A<B) ? 1 : 0; //ternary operator

addi $10, $0, 20000

addi $11, $0, 30000

slt $14, $10, $11

#Exercise 6: Exclusive OR without XOR

# Initial Variable Values: B = 0x0F0F; C = 0x0FFF;

# Assignment statement to perform: D = (B & ~C) | (~B & C)

addi $11, $0, 0x0F0F

addi $12, $0, 0x0FFFf

nor $1, $0, $11

nor $2, $0, $12

and $3, $11, $2

and $4, $1, $12

or $13, $3, $4

Convert the following MIPS assembly code into machine language. Show details, show binary and

hexadecimal values

|  |  |  |
| --- | --- | --- |
| MIPS Assembly | Binary | Hex |
| add $t0, $s0, $s1 | 0000 0010 0001 0001 0100 0000 0010 0000 | Ox02114020 |
| lw $t0, 0x20($t7) | 1000 1101 1110 1000 0000 0000 0010 0000 | 0x8DE80020 |
| addi $s0, $0, -10 | 0010 0000 0001 0000 1111 1111 1111 0110 | 0X2010FFF6 |
| addi $s0, $0, 73 | 0010 0000 0001 0000 0000 0000 0100 1001 | 0x20100049 |
| sw $t1, -7($t2) | 1010 1101 0100 1001 1111 1111 1111 1001 | 0XAD49FFF9 |
| sub $t1, $s7, $s2 | 0000 0010 1111 0010 0100 1000 0010 0010 | 0x02F24822 |

Convert the following program from machine language into MIPS assembly language.

The numbers on the left are the instruction addresses in memory, and the numbers on the right give the

instruction at that address. Then reverse engineer a high-level program that would compile into this

assembly language routine and write it. Explain in words what the program does. $a0 is the input, and it

initially contains a positive number, n. $v0 is the output.

|  |  |  |
| --- | --- | --- |
| Address | Instruction | MIPS |
| 0x00400000 | 0x20080000 | addi $t0, $0, 0 |
| 0x00400004 | 0x20090001 | addi $t1, $0, 1 |
| 0x00400008 | 0x0089502A | slt $t2, $a0, $t1 |
| Ox0040000C | 0x15400003 | bne $t2, $0, 0x0003 |
| 0x00400010 | 0x01094020 | add $t0, $t0, $t1 |
| 0x00400014 | 0x21290002 | addi $t1, $t1, 2 |
| 0x00400018 | 0x08100002 | j 0x0100002 |
| 0x0040001C | 0x01001020 | add $v0, $t0, 0 |
| Ox00400020 | 0x03E00008 | jr $ra |



