ECE 3504: Fundamentals of Computer Architecture

**Homework 2**

You must work on this assignment individually. Your submission must be your original work.

Please follow the submission instructions posted on canvas exactly.

### Question 1 – Assembly (15 points)

Consider the code snippet below:

add $t0, $t1, $zero

loop: addi $t1, $t0, -5

lw $s0, 40($9)

add $t0, $t0, 4

beq $s0, $s1, loop

sll $s0, $5, 5

Answer the following questions:

1. What is the instruction format (R/I/J) for the **lw** instruction? (2 points)

I

1. Give the name of an instruction that uses J-type formatting. Give the range of addresses that you can jump to using your chosen instruction. (5 points)

Jump (j): 26 bits + 2 bits 🡪0 to 268435456

1. What is the machine code representing the **beq** instruction in the above snippet? Assume loop is located at address 0x004100A0. (8 points)

Opcode: 0b0001 00

RS: 1 0000

RT: 1 0001

IMM: -3 🡪FFFD

0b0001 0010 0001 0001 1111 1111 1111 1101

0x1211FFFD

### Question 2 – Disassembly (20 points)

One of the below instructions is a store instruction.

0x14040002

0x00c23021

0xac240204

Which one of the above is a store instruction? (4 points)

0xac240204

Disassemble the other two (NOT the store) instructions into assembly. Write down the assembly instruction using register names and decimal values, if it uses any registers or immediate values. (16 points)

0x14040002 🡪 000101 00000 00100 0000000000000010

BNE $zero $a0 2

0x00c23021 🡪 000000 00110 00010 00110 00000 100001

ADDU $a2 $a2 $v0

**Question 3 – Compiler tool chain (20 points)**

The following code is a combination of true and pseudo instruction. Create the symbol table (6 points) and relocation table (8 points) after the first pass of the assembly. Put a cross on the side of any row that is eliminated after the second pass of assembly. Put all your addresses relative to top of function, (e.g. Func+8 and so on)

Consider these 4 times during code translation:

1) after 1st pass of assembly

2) after 2nd pass of assembly

3) after linking

4) after loading

In the dotted space in front of each instruction, indicate at what point from above times, each instruction is fully resolved into machine code.

You can assume *otherFunc* is defined in another file that will be linked to this file later. On the other hand, *nearFunc* is locally defined in the same assembly file.

.globl Func

Func: ~~la $a0, Array~~  2

addi $v0, $0, 0 2

loop: addi $s0, $s0, 1 2

sll $t0, $s0, 2 2

jal otherFunc 3

beq $v0, $s0, loop 2

jal nearFunc 2

jr $ra 2

nearFunc: some instructions here 2

.data

Array: .word 5, 4, 3, 2, 1, 0

Symbol table:

|  |  |
| --- | --- |
| **Symbol** | **Value** |
| Func | Func + 0 |
| loop | Func + 8 |
| nearFunc | Unknown |

Relocation table:

|  |  |  |
| --- | --- | --- |
| Address | Symbol | Type |
| Func + 16 | otherFunc | jal |

### Question 4 – Use of the stack (25 points)

The following code has three different functions, *Main*, *SumFunc*, and *LoadFunc*. The functions do **not implement MIPS caller/callee conventions**. Each function has a description above it to give you an overview of the function. An example .data section is given at the end of the code section.

The purpose of this question is to evaluate your understanding of stack operations in MIPS.

Add as many instructions as needed to implement caller/callee conventions. Do not put unnecessary data on the stack. There might be more than one correct answer. Use the provided callouts to insert your instructions. Some callouts may remain empty.

.text

## main is a test case for the SumFunc internal function.

# In memory, are the value A and the value B.

# The compute sum will be stored in memory at Result.

main: la $t0, A #loads address of A

la $t1, B #loads address of B

la $t3, Result #loads address of Result

move $a0, $t0 #address of A is input to SumFunc

move $a1, $t1 #address of B input to SumFunc

**addi $sp, $sp, -12**

**sw $a0, 8($sp)**

**sw $a1, 4($sp)**

**sw $t3, 0($sp)**

jal SumFunc #calls SumFunc

**lw $a0, 8($sp)**

**lw $a1, 4($sp)**

**lw $t3, 0($sp)**

**addi $sp, $sp, 12**

sw $v0, 0($t3) #saves sum into Result

##SumFunc takes two inputs

# ($a0 –address of A, $a1- address of B)

# and returns a value in $v0

# returned is the sum of the values at those memory locations

SumFunc:

**addi $sp, $sp, -4**

**sw $ra, 0($sp)**

move $s0, $a0 #moves addr of A into saved register

move $s1, $a1 #moves addr of B into saved register

move $a0, $s0 #addr of A is input to 1st LoadFunc call

jal LoadFunc #call LoadFunc to get value of A

move $s0, $v0 #moves returned value to $s0

#(this is now the value of A)

move $a0, $s1 #base addr of array is AddressFunc input

jal LoadFunc #call LoadFunc to get value of B

move $s1, $v0 #moves returned value to $s0

#(this is now the value of B)

add $s1, $s1, $s0 #adds the two values together

move $v0, $s1 #return sum as output of SumFunc

**lw $ra 0($sp)**

**addi $sp, $sp, 4**

jr $ra # return to main

## AddressFunc takes one input ($a0 – address of value)

# and returns a value in $v0 – the value at addr passed in

LoadFunc:

lw $v0, 0($a0) #loads value of address

jr $ra #return to SumFunc

.data # User Data segment

A: .word 5

B: .word 7

Result: .word 0

**Question 5 – MIPS load/store and branches (20 points)**

Assume that *A* and *B* are equal-length integer arrays. They are both defined in data segment with labels ***ArrayA*** and ***ArrayB****.* Also assume that their size is defined in data segment with the label ***SizeN***.

Write a piece of code that starting at index 0, searches for the first occurrence of the case, where both *A[i]* and *B[i]* are 0 and saves that index in a memory location already defined as ***Result***. If such case does not exist, the result will be equal to ***SizeN.***

For the following example, ***Result*** will be 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | 1 | -2 | 0 | 0 | 2 |
| B | 6 | 0 | 1 | 0 | 2 |

Make sure your code does not access memory locations outside the boundary of the arrays. With the exception of **li**, **la** and **move**, you should use the MIPS core instruction set only. For this question, you are NOT responsible for saving registers to stack before using them and can freely use any register. Make sure to comment your code sufficiently.

.text

and $t0 $t0 $zero // counter

addi $t1 $zero SizeN // size of array

sll $t1 $t1 2 // mult 4

Loop:

lw $t2 ArrayA($t1) // load array a elem

lw $t3 ArrayB($t1) // load array b elem

beq $t2 $t3 exit // equal

addi $t1 $t1 4 // increment to next data spot

bne $t0 $t1 Loop // loop as long as in bounds

exit:

srl $t1 $t1 2 // shift left to get index instead of mem offset

sw $t1 Result // store result in mem address Result