Homework 1- EECS 388: Embedded Systems, Spring 2024

Note: The answer must be submitted as a "typed" MS Word/PDF document. Total points: 2 (total 100 points in the course)

Q1 (a) Write the MIPS assembly code for the assignment statement as given below. Include comment with each line of assembly to describe the purpose/function of the line. (**0.5 points).**

N[150] = N[130] + k

Use the MIPS instruction reference guide on canvas. N is an array stored in the memory. Assume that register $s1 is storing the base address of array N and $s3 is storing the variable k. Use register $t1 for storing temporary values.

Note: Follow section "2.4 Representing Instructions in the Computer" from the book Computer Organization and Design: The Hardware/Software Interface. Third Edition (Link). Also use the reference documents uploaded on Canvas.

Solution Q1(a):

Assembly code:

```
lw $t1, 520($s1)    # temporary register $t1 gets N[130]
add $t1, $t1, $s3   # temporary register $t1 gets N[130] + k
sw $t1, 600($s1)    # store N[130]+k back into N[150]
```

(b) For each line of the assembly code in question 1(a), write the MIPS machine code in hex using the example table below that labels the different hex segment. The table is for R type instruction. For I and J type, use appropriate table. (**0.5 points).**

Solution Q1(b):

op	rs	rt	rd	Address/ shamt	funct

Note: if you write in either decimal/hex binary in the table correctly, the answer is acceptable. If the table is complete and correct, full marks are given. But please note how the Final Hex is calculated.

Instruction	Format	Op (6bit)	Rs (5bit)	rt(5bit)	Address (16bit)
lw $t1, 520($s1) or lw$t1,0x208($s1)	I-Format	Decimal:35 Bin: 0b100011	Decimal: 17 Bin: 0b10001	Decimal: 9 Bin: 0b01001	Decimal: 520 Bin:0b0000001000001000

Final Hex: 0x8e290208

Homework 1- EECS 388: Embedded Systems, Spring 2024

Instruction	Format	op(6bit)	rs(5bit)	rt(5bit)	rd(5bit)	Shamt (5bit)	funct(6bit)
add $t1, $t1, $s3	R-Format	Decimal: 0 Binary:000000	Decimal:9 Binary:01001	Decimal:19 Binary:10011	Decimal:9 Binary:01001	Decimal:0 Binary: 00000	Decimal:32 Binary: 100000

Final Hex: 0x01334820

Instruction	Format	op	rs	rt	address
sw $t1, 600($s1)	I-Format	Decimal: 43 Binary: 101011	Decimal: 17 Binary: 10001	Decimal: 9 Binary: 01001	Decimal: 600 Binary:0000001001011000

Final Hex: 0xae290258

Note: You can use different label names and other registers instead of $t2. If you tried an alternative approach and was not graded correctly, please email me.

Q2 Write the MIPS assembly code for the C segment below **(0.6 points).**

```
while(arr[i] == k){
        i + = 3;
}
```

Assume that variable i corresponds to register $s3 and k corresponds to register $s4. The base of the array arr is in $s1

Solution Q2(a):

```
loop_start:
   sll $t1, $s3, 2        # $t1 = i * 4, converting index i to byte offset for word-sized elements
   add $t1, $t1, $s1      # $t1 = address of arr[i], adding base address of arr to offset
   lw $t2, 0($t1)         # $t2 = arr[i], loading the word at arr[i] into $t2 (or other register available)
   bne $t2, $s4, loop_end  # if arr[i] != k, exit the loop.
   addi $s3, $s3, 3       # i += 3, incrementing i by 3
   j loop_start           # jump back to the start of the loop to check condition again
loop_end:
```

Homework 1- EECS 388: Embedded Systems, Spring 2024

Q3 (a) Briefly describe the need for stack for function call. **(0.2 points)**

(b) Describe the difference between caller and callee saved registers and what specific registers are caller and callee saved in MIPS. **(0.2 points)**

Answer:

(a) Key points: to save registers that could get overwritten during the function call. For example, a general purpose registers (e.g., $s0) used by the caller function could be needed after the function call. If the Callee function overwrites it, the rest of the operations after returning to the caller function could be incorrect. Also, return addresses must be stored to the stack, if there are subsequent function calls in the callee function.

(b) Caller-saved registers are those that the calling function must save before calling another function if it wishes to preserve their values. Callee-saved registers must be saved by the called function if they are used, as it is responsible for restoring their original values before returning. In MIPS, registers $t0 to $t9, $a0 to $a3, and $v0 to $v1 are caller-saved, while $s0 to $s7 (saved registers) are callee-saved. The register $ra (return address) is also callee-saved as the called function must return control to the caller.