CS220 Quiz#3

General instructions: Please write brief explanation for your answers. If you submit multiple times, your last submission will be used for grading. Please provide an email address below where your responses can be sent.

Email address *

sohamg@iitk.ac.in

Your name *

Soham Ghosal

5.5/6

Very good!

Your roll number *

180771

Q1. Write the branch instruction names of the MIPS ISA that will be generated by a reasonably optimized compiler from this C statement: if ((x != y) || (x < 0)). [1 point]

This is an OR statement, or a disjunction.
So we should look for the first true statement.

The branch instruction names that will be generated are:

1

1. bne(check if not equal)
2. bltz(check if less than zero)

Q2. Write the branch instruction names of the MIPS ISA that will be generated by a reasonably optimized compiler from this C statement: if ((x != y) && (x <= 0)). [1 point]

This is an AND statement, or a conjunction.
So we should look for the first false statement.

The branch instruction names that will be generated are:
We will basically invert the conditions inside the if statement

1

1. beq(check if equal)
2. bgtz(check if greater than zero)

Q3. Consider a function f written using the C language. The function f calls another function g having ten arguments all of type integer. The function f can allocate all its local variables in registers without spilling. The structure of f is as follows: {... return g (a, b, c, d, e, ...);}. How much stack space in bytes should be allocated to f when compiling for 32-bit MIPS? [1 point]

The function f can allocate all its local variables in registers without spilling.
g has 10 arguments, all integers, each argument is of 4 bytes.
4 of those arguments can fit in $a0,$a1,$a2,$a3
The remaining 6 have to be spilled into memory, and stack space has to be allocated.
Additionally, $ra needs to be stored, so that we can return back to f after the execution of g.
In total we need 7*4=28 bytes of stack space.

1

Q4. Consider the following sequence of 32-bit MIPS instructions separated by semi-colons: [addi $t0, $0, 0xb2; sll $t0, $t0, 0x18; addi $t1, $0, 0x4; srav $t0, $t0, $t1]. What is the final hexadecimal value in $t0? [1 point]

1. 0xb2 will be sign extended and stored in $t0.Thus, $t0 stores 0xFFFFFFB2
2. We left shift by 24 bits.That is multiply by 2^24. Thus, $t0 stores 0xB2000000
3. 0x4 is sign extended and stored in $t1. Thus $t1 stores 0x00000004
4. Shift right arithmetic variable-->Shift right by 4 bits, and extend the sign bit.
Thus $t0 stores 0xFB200000

The final value stored in $t0 is 0xFB200000

1

Q5. Consider the following segment of C code: [for(i=0;i<18;i++) { if (i%2==0) { // Some non-branch statements } else if (i%3==0) { // Some non-branch statements } }]. This code is translated to 32-bit MIPS such that the translated code has the minimum number of branch/jump instructions. When the translated code is executed, calculate how many forward branches/jumps and how many backward branches/jumps are executed. [1+1 points]

i%2==0 --> 0,2,4,6,8,10,12,14,16
i%3==0 --> 3,9,15
None--> 1,5,7,11,13,17
Lets count jumps purely due to for loop.
After every iteration, the value of i increases, and there is a backward jump to the start of the loop. This happens 18 times[corresponding to i=0,1,2...17]
So purely due to the for loop, we have 18 backward jumps, and 0 forward jumps.

Now coming to the conditional,

Lets say we first check if i is divisible by 2,and then check if i is divisible by 3.
If i is divisible by 2, we have a forward jump(to label 1), otherwise there is fallthrough.
Then for the fallthrough case, if i is not divisible by 3, there is a forward jump out of the conditional otherwise fallthrough, there is a forward jump out of the conditional.

We also put label 2 above label 1, since, number of elements hitting label 1 is more and we ought to minimise the number of jumps.
Thus every element hitting label 2, has to jump, to avoid label 1.

In total due to the conditional we have 9+6+3=18 forward jumps.

Rough structure:

(i%2==0) if yes: jump to label 1, otherwise fallthrough
(i%3==0) if yes: fallthrough, otherwise jump outside conditional
j outsideconditional

label 1:.....

1.5

So total, we have 18 backward and 18 forward jumps.