Computer Architecture and Programming Languages Course: CO20-320241
Jacobs University Bremen November 4, 2019
Jovan Shandro

Homework 7

Problem 7.1

Solution:
Since our function has two arguments, we will let $a0 and $a1 be the registers that store x and y respectively. The return value of the function will be stored in register $v0. I use *slt* and then *beq* to check if the value of x in register $a0 is greater than 10 or not and depending on that we return. In order to make the comparison we need to store 10 somewhere, so I am storing it in register $t0. So the code is:

```
my_function:  addi $t0, $0, 10
              slt $t1, $t0, $a0
              beq $t1, $0, ELSE
              sub $v0, $a0, $a1
              jr $ra
ELSE:         add $v0, $a0, $a1
              jr $ra
```

Problem 7.2

Solution:
In this code we have 2 procedures, is_more_than_fifty and prod. They both have 2 arguments so for both I will use $a0 for a and $a1 for b. Since neither of the two functions changes the value of a or b, there is no need to store them on the stack. So the only register we need to store in the stack is $ra before we call prod in the is_more_than_fifty function as it will change the value and we need to store the previous value so that the first function also knows where to return. As for finding the product, I am using the mult and mflo commands we learned in class. So the code is:

```
is_more_than_fifty:  addi $t0, $0, 50     # Now $t0 stores 50
                     addi $sp, $sp, -4    # Move stack top by one
                     sw $ra, 0($sp)       # Store $ra in the stack
                     jal prod             # Call prod
                     slt $t0, $t0, $v0    # Check if prod value greater than 50
                     add $v0, $t0, $0     # Since return values are the same
                     lw $ra, 0($sp)       # Load $ra from the stack
                     addi $sp, $sp, 4     # Update stack
                     jr $ra               # Return
prod:    mult $a0, $a1     # Multiply the registers
         mflo $v0          # Get the mult value
         jr $ra            # Return
```

Problem 7.3

Solution:
Let i be the variable stored in register $s3. In the first 2 lines, the value of i is multiplied by 4 and then added to $s6, so by the second line $t1 has the value &A[i]. In the third line we store the value of A[i] in $t0 and than compare it to $s5 which is -1. If there is equality, we break the loop, if not the next line increments i and then goes to the next iteration. So the equivalent C code is:

```
int i = 0;
while(A[i] != -1)
  i++;
```

Problem 7.4

Solution:

Since we are given that the loop is placed in memory location 60000, the first instruction will be stored in line memory location 60000, and since the addresses of sequential words differ by 4, the second instruction is stored in location 60004, the third one in 60008 and so on. Each of the commands has a specific value for the 6 bit op field. For sll and add it is 0, for lw it is 35, for beq it is 4, for $addi$ is 8, and for j it is 2. sll and add are of R-type, so they have all fields rs, rt, rd, shamt, and funct. We know \$t0 is the 8^{th} register, \$t1 is the 9^{th}, \$s3 is the 19^{th}, \$s5 is the 21^{st}, and \$s6 is the 22^{nd}. beq, lw, and $addi$ are I-type instructions, so they have the op, rs and rt fields and then the immediate. So now before being able to write the machine code, we only need to find the value of EXIT in the beq command, and the value of Loop in the j command. As also stated in the question, beq will jump relative to the following instruction, so from the following instruction EXIT is only 2 words away, so the value stored will be 2. As for the j command, it uses absolute addresses, and the absolute address of Loop is 60000, which in binary will be 1110101001100000. Remember that the address is in 32 bits, and the j command stores only 26, so 6 bits have to be removed, and these bits will be the leftmost 4, which in our case will be 0, and the rightmost 2, so ignoring the left side zeros the number stored will be 11101010011000 which is equal to 15000. So new we can translate the Mips code in machine code:

PC	Instruction	Machine Code (decimal)					
60000	Loop: sll \$t1, \$s3, 2	0	0	19	9	2	0
60004	add \$t1, \$t1, \$s6	0	9	22	9	0	32
60008	lw \$t0, 0(\$t1)	35	9	8	0		
60012	beq \$t0, \$s5, Exit	4	8	21	2		
60016	$addi$ \$s3, \$s3, 1	8	19	19	1		
60020	j Loop	2	15000				
60024	EXIT:						

So the machine code in binray representation will be :

PC	Machine Code (decimal)					
60000	000000	00000	10011	01001	00010	000000
60004	000000	01001	10110	01001	00000	100000
60008	100011	01001	01000	0000000000000000		
60012	000100	01000	10101	0000000000000010		
60016	001000	10011	10011	0000000000000001		
60020	000010	00000000000011101010011000				

Problem 7.5

Solution:

a) For the first bit pattern, since the leftmost bit is 0, so the number is positive so we just calculate the value as following:

$0x0C000000 = C \cdot 16^6 = 12 \cdot 16^6 = 12 \cdot 16,777,216 = 201,326,592_{10}$

For the second bit pattern the leftmost bit is C > 8, so we have to invert (with respect to F) and then add 1 to find the positive vale as the number itself will be negative. So we have:

$0xC4630000 \xrightarrow{inv} 0x3B9CFFFF \xrightarrow{+1} 0x3B9D0000$

Now converting it to decimal we get:
$0x3B9D0000 = 3{\cdot}16^7 + B{\cdot}16^6 + 9{\cdot}16^5 + D{\cdot}16^4 + 0 = 805,306,368 + 184,549,376 + 9,437,184 + 851,968 = 1,000,144,896_{10}$
so $0xC4630000 = -1,000,144,896_{10}$

b) For the first bit pattern, since the leftmost bit is 0, the result is same as in point a). Now we perform the calculations for the second pattern. We have:
$0xC4630000 = C{\cdot}16^7 + 4{\cdot}16^6 + 6{\cdot}16^5 + 3{\cdot}16^4 + 0 = 12{\cdot}16^7 + 4{\cdot}16^6 + 6{\cdot}16^5 + 3{\cdot}16^4 = 3,221,225,472 + 67,108,864 + 6,291,456 + 196,608 = 3,294,822,400_{10}$

c)Converting the first bit pattern in binary we get $0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
or $000011\ 00000\ 00000\ 00000\ 00000\ 000000_2$
The instruction with op field equal to 3 is jal, so it expect all other 26 registers to be the target and since they are all 0, the full instruction will be : **jal** 0

Converting the second bit pattern in binary we get $1100\ 0100\ 0110\ 0011\ 0000\ 0000\ 0000\ 0000_2$
or $110001\ 00011\ 00011\ 00000\ 00000\ 000000_2$
The instruction has op field equal to 49, so the instruction is lwc1 (load word in co-processor 1, reference : click here). The offset is 0 and both the source register and the destination registers are 3 so the full instruction would be:
lwci $f3, 0($v1)