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

Homework 7

Problem 7.1

Solution:

The corresponding MIPS assembler code would be:

```
my_function:
   addi $t0, $0, 10    # store 10 in temporary register $t0
   slt $t1, $t0, $a0   # check whether 10 is smaller than register $a0,
                       # which is the first argument a function takes,
                       # in our case x (so, check if x>10)
   bne $t0, $0, ELSE   # if the above condition is not equal to zero,
                       # which means if it is true (case x>10), go to
                       # ELSE section that performs x-y
   add $v0, $a0, $a1   # if $t1 was equal to 0, so if x<=10, we perform
                       # x+y, stored respectively in $a0 and $a1, and
                       # store the sum in the return value $v0
   jr $ra              # return

   ELSE:   sub $v0, $a0, $a1   # case x<10, perform x-y, so $a0 - $a,
                               # and store it in return value $v0
           jr $ra              # return
```

Problem 7.2

Solution:

The corresponding MIPS assembler code:

```
# Define function prod, using the operation mult:
prod:
   mult $a0, $a1 # using operator mult to calculate the product of
                 # arguments a and b stored respectively in $a0, $a1
   mflo $v0      # we get the product as the return value of the function
                 # using mflo, so $v0 = lo
   jr $ra        # return

# Define function is_more_than_fifty, which calls also the above defined
function prod:
is_more_than_fifty:
   addi $t0, $0, 50    # store 50 in register $t0
   addi $sp, $sp, -4   # adjust stack for one item
   sw $ra, 0($sp)      # save the return address
   jal prod            # call function prod using jal
   slt $t1, $t0, $v0   # check if the return value of prod(a,b) is more
                       # than 50, stored previously in $t0
   add $v0, $t1, $0    # the function will return the value stored in
                       # $t1, so if 50 < prod(a,b), slt will store in
                       # $t1 the value 1, otherwise the function will
                       # return 0
   lw $ra, 0($sp)      # load the return address
   addi $sp, $sp, 4    # pop one item off stack
   jr $ra              # return
```

Problem 7.3

Solution:

For the MIPS code given, the first 3 lines in the LOOP are trying to access some k^{th} element of the array A, since in the beginning we shift 2 bits, which is same as multiplying by 4, and then we add it to the base address of A, and load the value. After that, it is checked whether this value of A[k] is equal to value stored in $s5 or not (A[k] == -1). Since when the value is equal, the loop ends, the loop's condition will be A[k] != -1. Then, '`addi $s3,$s3,1`' is just the loop's variable increment. Therefore, the corresponding C code will be:

```
for(int k=0; A[k]!=-1; k++) {}
```

Problem 7.4

Solution:

For the first instructions `sll` and `add`, the R-type format is used:

op	rs	rt	rd	sahmt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

→ For `sll`, we have: op, rs, and func zero, rt, rd correspond to the specific register numbers, while shamt will be 2, since that's how many bits are shifted. The memory location will be 60000, since it's the first instruction.
→ For `add`, we have: op and shamt zero, rs, rt, rd correspond to the specific register numbers, func value is 32. The memory location will be 60004, since the addresses in MIPS are in bytes, so addresses of sequential words differ by 4, as stated in the exercise.

For instructions `lw`, `beq`, and `addi`, the following format is used:

op	rs	rt	const
6 bits	5 bits	5 bits	16 bits

→ For `lw`, we have: op = 35, rs, rt correspond to the specific register numbers, the constant value in this case is zero.
→ For `beq`, we have: op = 4 and rs, rt correspond to the specific register numbers. As for the constant value, we need to consider the EXIT part. It is also stated in the exercise that `beq` will jump relative to the following instruction, so since EXIT is located after 2 instructions, const will be 2.
→ For `addi`, we have: op = 8, rs, rt correspond to the specific register numbers, the constant value in this case is 1.

For instruction `j`, the following instruction format is used:

op	addr
6 bits	26 bits

→ For `j`, we have op = 2. As for the address, since we jump to `Loop`, we need to get the address 60000 (1110101001100000). Considering the fact that this address is in 32 bits (adding zeros in the left), we need to convert it to some 26-bit value. The removal of 6 bits is done in the following way: we remove 4 bits starting from the left, and 2 bits from the right. So, we will get 11101010011000, since in the beginning we had zeros, we only consider this representation. The decimal value is 15000.
Therefore, the table will be:

Memory	Code	Machine Language					
60000	`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:`	-					

Table in binary form:

Machine Language					
000000	00000	10011	01001	00010	000000
000000	01001	10110	01001	00000	100000
100011	01001	01000	0000000000000000		
000100	01000	10101	0000000000000010		
001000	10011	10011	0000000000000001		
000010	00000000000011101010011000				

Problem 7.5

Solution:

a)

(i) 0x0C000000

Since we are considering the b complement form, we check the leftmost bit, which is zero, so the number is positive and we perform normal conversion: $0x0C000000 = 12 \cdot 16^6 = 201326592_{10}$

(ii) 0xC4630000

Checking again the leftmost bit, which is C = 12 = 1100, so the number is negative, and we need to perform inversion and addition with 1:

F	F	F	F	F	F	F	F
- C	- 4	- 6	- 3	- 0	- 0	- 0	- 0
3	B	9	C	F	F	F	F
+							1
3	B	9	D	0	0	0	0

Now, we convert the number normally:
$0x3B9D0000 \rightarrow 3{\cdot}16^7 + B{\cdot}16^6 + 9{\cdot}16^5 + D{\cdot}16^4 = 1000144896_{10} \rightarrow -1000144896_{10}$

b)

(i) 0x0C000000 $= 12 \cdot 16^6 = 201326592_{10}$

(ii) 0xC4630000 $= C \cdot 16^7 + 4 \cdot 16^6 + 6 \cdot 16^5 + 3 \cdot 16^4 = 3294822400_{10}$

c)

(i) 0x0C000000 $\rightarrow$ 0000 1100 0000 0000 0000 0000 0000 0000

Considering 6 bits for op, we get op = 000011 = 3 and all the others are 0. Therefore, we have the operand `jal` since op=3, and the other part is zero, so the MIPS instruction is: `jal 0`

(ii) 0xC4630000$\rightarrow$ 1100 0100 0110 0011 0000 0000 0000 0000_2

Considering 6 bits for op, we have op = 110001 = 49 (instruction `lwci`), which means: Load Word to Floating-Point. This instruction uses the format: base(5 bits), ft(5 bits), offset(16 bits), where ft $\leftarrow$ memory[base+offset]. So, we have base = 00011 = 3, ft = 00011 = 3, and offset = 0, which means that the MIPS intruction will be: `lwc1 $f3, 0($v1)`