Computer Architecture and Programming Languages
Jacobs University Bremen
Romelda Blaceri

Course: CO20-320241
21 October, 2019

Homework 6

Problem 6.1

Solution:
MIPS has in total 32 general purpose registers. Considering the fact that $11111_2 = 31_{10}$, we see that we can express all the 32 registers (including 0) using only 5 bits.

Problem 6.2

Solution:

a) Considering the fact that `op=0` and `funct=34`, we say that the operation will be subtraction (`sub`). The source registers `rs=8` and `rt=9` stand for `$t0` and `$t1` respectively. The destination register `rd=10` corresponds to `$t2`. Therefore, the MIPS instruction will be:

```
sub $t2, $t0, $t1    # $t2 = $t0 - $t1
```

a) Since `op=0x23`, we have 0x23 $\rightarrow 23_{16} = 2 \cdot 16 + 3 = 35_{10}$, so `op=35`, which means that the operation is `lw`. The registers `rs=17` and `rt=18` stand for `$s1` and `$s2` respectively. The constant term is 0x4 $\rightarrow 4_{16} = 4_{10}$, which means that the bit addres is 4. Therefore, we have the following MIPS instruction:

```
lw $s2, 4($s1)    # value stored in A[1] (for some array A with base
                  # address in $s1) is loaded in $s2
```

Problem 6.3

Solution:
a) Considering the example given in the lecture slides, we have:

op	rs	rt	rd	sahmt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
0	8	9	10	0	34
000000	01000	01001	01010	00000	100010

Therefore, the MIPS instruction in binary for $\rightarrow$ `sub $t2, $t0, $t1`, is:
000000 01000 01001 01010 00000 100010

b)

op	rs	rt	const
6 bits	5 bits	5 bits	16 bits
0x23 = 35_{10}	17	18	0x4 = 4_{10}
100011	10001	10010	0000000000000100

The MIPS instruction in binary for $\rightarrow$ `lw $s2, 4($s1)`, is:
100011 10001 10010 0000000000000100

Problem 6.4

Solution:
In the beginning, `slt` stores in `$t2` the condition `$t0 < $t1`, which in our case is true (If we start comparing the first 4 bits for both values, we have 0010 < 0011, so `$t0 < $t1`). Since the condition is true, 1 will be stored in temporary register `$t2`. Then, `beq` checks if `$t2` is equal to 0 or not. Since it is false (`$t2` $\neq$ `0`), we go to the next line, which makes us skip the `ELSE` instruction and jump to `DONE:`. Therefore, the final value of `$t2` after executing the given MIPS instructions, will be 1 $\rightarrow$ `$t2 = 0000 0000 0000 0000 0000 0000 0000 0001`.

Problem 6.5

Solution:

```
# We need to perform:  A[6]+=$s1:

lw $t0, 24($s0)      # the value stored in A[6] is loaded and stored
                     # in temporary register $t0
add $t0, $t0, $s1    # addition of the values stored in registers $t0
                     # and $s1 is stored in $t0
sw $t0, 24($s0)      # the value stored in temporary register $t0
                     # is now stored in A[6], so the content of $s1
                     # is added to A[6]

# Note that when we save the array values in temporary registers, to
# access the array data, we multiply the index by 4.
```

Problem 6.6

Solution:

In order to load 0000 0000 0010 0011 0000 0000 0010 0011, we follow this procedure: load upper 16 bits first using `lui`, and then add the lower 16 bits using `ori`. Therefore, we have:

* upper 16 bits are $0000000000100011 = 35_{10}$
* lower 16 bits are $0000000000100011 = 35_{10}$

The MIPS assembler instructions will be:

```
lui $s4, 35         # resets lower 16 bits
ori $s4, $s4, 35    # loads 0s into upper bits of constant
```

Final value of `$s4` is 0000 0000 0010 0011 0000 0000 0010 0011.

Problem 6.7

Solution:

```
# The MIPS assembler code:

addi $t0, $0, 0    # store 0 in $t0 (initialize i=0)

# Start the for loop
LOOP:   slti $t1, $t0, 8     # check whether value stored in $t0 is
                             # lower than 8, so check if i<8, and store
                             # 1 or 0 in $t1
        beq $t1, $0, EXIT    # check whether $t1 is 0, so whether i<8
                             # is false, and exit loop
        addi $s0, $s0, 4     # if we haven't reached EXIT in the previous
                             # line(if i<8), we do $s0+=4 (a = a + 4)
        addi $t0, $t0, 1     # store in $t0 its incremented value (i++)
        j LOOP               # go in the next iteration of the loop

EXIT:   # case i>=8
```