Computer Architecture and Programming Languages Course: CO20-320241
Jacobs University Bremen October 21, 2019
Jovan Shandro

Homework 6

Problem 6.1

Solution:

Since we use 5 bits, it mean that the largest value we can express is 11111_2, which has a decimal value of 31. Together with register 0, 5 bits suffice to represent 32 registers, which is the maximal number of registers in MIPS.

Problem 6.2

Solution:

(a) We have op = 0, so we can say that the operation is either addition or subtraction. Since funct = 34, the operation is subtraction. The first register has value rs = 8, so we have $t0. The second one is rt = 9, so we have $t1. The value will be stored in $t2 since rd = 10.

$\longrightarrow$ So the MIPS assembler instructions that will store the difference $t0 - $t1 in register $t2, are: **sub** $t2, $t0, $t1

(b) We can see that some of the values are in hexadecimal, so we first convert them into decimal. Therefore, we get: $0x23 = 2 \cdot 16^1 + 3 \cdot 16^0 = 35_{10}$ and $0x4 = 4 \cdot 16^0 = 4_{10}$. So now we need to find the MIPS instructions for op = 35, which is operation lw, rs = 17, which is register $s1, rt = 18, which is register $s2, and finally const = 4, which is a 16 bit constant = 4.

$\longrightarrow$ So the MIPS assembler instructions that will load the value of an array with starting address in $s1 in the register $s2, are: **lw** $s2, 4($s1)

Problem 6.3

Solution:

(a) According to the slides, we have:

op (6)	rs (5)	rt (5)	rd (5)	sahmt (5)	funct (6)
sub (0)	$t0 (8)	$t1 (9)	$t2 (10)	unused (0)	34
000000	01000	01001	01010	00000	100010

Where,
$\rightarrow$ op (6) means op occupies 6 bits;
$\rightarrow$ $t0 (8) means $t0 has register number 8.

The instruction code is: 000000 01000 01001 01010 00000 100010

(b)

op (6)	rs (5)	rt (5)	const (16)
lw (0x23)	$s1 (17)	$s2 (18)	const (0x4)
100011	10001	10010	0000000000000100

The instruction code is: 100011 10001 10010 0000000000000100

Problem 6.4

Solution:

→ **slt** $t2, $t0, $t1 : In temporary register $t2 is stored either 0 or 1, depending on whether $t0 is smaller than $t1 or not. Since, we have $t0 = 0010 0100 1001 0010 0100 1001 0010 0100 = 613566756_{10} and $t1 = 0011 1111 1111 1000 0000 0000 0000 0000 = 1073217536_{10}, we see that $t0 < $t1, so $t2 becomes 1.

→ **beq** $t2, $0, ELSE : Checks whether the value of $t2 is 0. If yes, it jumps to ELSE, if not we continue to the next line. Since the value in $t2 is 1, we read the next line.

→ **j** DONE : We jump to DONE, skipping ELSE: **addi** $t0, $0, 2.

→ So, the value stored in $t2 in the end is 1. Since we're storing the values in 32 bits, $t2 stores 0000 0000 0000 0000 0000 0000 0000 0001.

Problem 6.5

Solution:

We load the value from A[6] and store it in temporary register $t0, then perform the addition and store it again in A[6], so that in the end we would have in A[6] the sum of its previous value and the content of $s1. When trying to access the array values, we multiply the position by 4.

```
lw $t0, 24($s0)      # $t0 stores temporarily the value of A[6]
add $t1, $s1, $t0    # sum of the values stored in $s1 and $t0 is stored in $t1
sw $t1, 24($s0)      # the sum A[6]+$s1 is saved back to A[6]
```

Problem 6.6

Solution:

According to the information given in the lecture slides, to load values that are larger than 16 bits we need to load the upper 16 bits first using lui and then add the lower 16 bits using ori. In our case, if we see the number carefully, the lower and the upper 16 bits are the same, so in both cases we have: 0000 0000 0010 0011 = 35_{10}. So, the required MIPS code is:

```
lui $s4, 35          # 35 is loaded $s4 after shifting by 16 bits
ori $s4, $s4, 35     # 35 (the remaining 16 bits) is also stored in $s4, so that the
value of #s4 becomes 0000 0000 0010 0011 0000 0000 0010 0011
```

Problem 6.7

Solution:

The procedure we follow is: first we declare i=0, then we start the loop that checks if i<8, exits if the condition is not satisfied, and performs a += 4 otherwise. In the end, we increment i and jump to the beginning of the loop.

```
add $t0, $0, $0    # i=0
addi $t1, $0, 8    # value of $t1 becomes 8
FOR:    slt $t2, $t0, $t1      # check $t0 < $t1 and change $t2 accordingly
        beq $t2, $0, DONE      # if $t2=0 (i<=8), go out of the loop
        addi $s0, $s0, 4       # $s0 = $s0 + 4 or a = a + 4
        addi $t0, $t0, 1       # increase i by 1
        j FOR                  # begin the loop again

DONE:   # after i reaches 8 and the loop has finished
```