Computer Architecture and Programming Languages Course: CO20-320241
Jacobs University Bremen October 14, 2019
Jovan Shandro

Homework 5

Problem 5.1

Solution:

(a) We find: $14_{10} \rightarrow$ binary and $37_{10} \rightarrow$ binary

$14/2 = 7(0) \rightarrow\ 7/2 = 3(1) \rightarrow\ 3/2 = 1(1) \rightarrow\ 1/2 = 0(1) \longrightarrow 1110_2$

$37/2 = 18(1) \rightarrow\ 18/2 = 9(0) \rightarrow\ 9/2 = 4(1) \rightarrow\ 4/2 = 2(0) \rightarrow\ 2/2 = 1(0) \rightarrow$
$1/2 = 0(1) \longrightarrow 100101_2$

Binary addition:

$$\begin{array}{r} 00001110 \\ +\ 00100101 \\ \hline 00110011 \end{array}$$

We find: $00110011 \rightarrow$ decimal

$00110011 \longrightarrow 0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 32 + 16 + 2 + 1 = 51$

The result of our calculations: $14 + 37 = 51$.

(b) We find: $12_{10} \rightarrow$ binary and $27_{10} \rightarrow$ binary

$12/2 = 6(0) \rightarrow\ 6/2 = 3(0) \rightarrow\ 3/2 = 1(1) \rightarrow\ 1/2 = 0(1) \longrightarrow 1100_2$

$27/2 = 13(1) \rightarrow\ 13/2 = 6(1) \rightarrow\ 6/2 = 3(0) \rightarrow\ 3/2 = 1(1) \rightarrow\ 1/2 = 0(1) \longrightarrow$
11011_2

We need to subtract the numbers, so we find (-27) with 2's complement (we complete 8 bits, then invert the number, and add 1):

$\rightarrow 00011011_2$
$\rightarrow 11100100_2$
$\rightarrow 11100100_2 + 1 = 11100101_2$

Binary addition:

$$\begin{array}{r} 00001100 \\ +\ 11100101 \\ \hline 11110001 \end{array}$$

We find: $11110001 \rightarrow$ decimal with 2's complement method (sign bit = 1 $\implies$ negative number, so we subtract 1, invert the number and then convert it to decimal)
$\rightarrow 11110001_2 - 1 = 11110000_2$
$\rightarrow 00001111_2$
$\rightarrow 11110001 \longrightarrow 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1 = 15$
The result is: $12 - 27 = -15$

(c) We find: $69 \rightarrow$ BCD and $58 \rightarrow$ BCD

$69_{10} = 01101001_{BCD}$; $58_{10} = 0101\ 1000_{BCD}$

Since 6+5 and 9+8 > 9, we have:

	01101001	
+	01011000	
	11000001	←Invalid code groups, add 6 twice
	01100110	
	000100100111	

We find: $000100100111_{BCD} \rightarrow$ decimal
$000100100111_{BCD} = 127_{10}$

(d) We find: $275 \rightarrow$ BCD and $642 \rightarrow$ BCD

$275_{10} = 0010\ 0111\ 0101_{BCD}$; $642_{10} = 0110\ 0100\ 0010_{BCD}$

Since 7+4 > 9, we have:

	001001110101	
+	011001000010	
	100010110111	←Invalid code group, add 6 in the middle
	0110	
	100100010111	

We find: $100100010111_{BCD} \rightarrow$ decimal
$100100010111_{BCD} = 917_{10}$

e) Hexadecimal numbers are added in the following way:

```
   6 A F
 + 2 3 C
 -------
   8 E B
```

$\rightarrow F + C = 27 > 15 \implies 27 - 16 = 11 \rightarrow B$, and carry 1.
$\rightarrow A + 3 + 1 = 14 \rightarrow D$, and no carry.
$\rightarrow 6 + 2 = 8$.

f) The subtraction of hexadecimals is performed in the following way:

```
   5 9 4
 - 3 A 8
 -------
   1 E C
```

$\rightarrow 4 - 8 < 0 \implies$ get 1 (16) from other column $\rightarrow 4 + 16 = 20 \rightarrow 20 - 8 = 12 \rightarrow C$
$\rightarrow$ we took 1 for the other column $\implies 9 - 1 = 8 \rightarrow 8 - A < 0 \implies$ get 1 (16) again $\rightarrow 8 + 16 = 24 \rightarrow 24 - A = 14 \rightarrow E$
$\rightarrow$ we took 1 for the previous $\implies 5 - 1 = 4 \rightarrow 4 - 3 = 1 \implies$ result is $1EC$.

Problem 5.2

Solution:

a) Instruction add makes possible to store the sum of \$s0 and \$s1 in \$t0:

```
add $t0, $s0, $s1     # a = b + c
```

b) 'subtract' stores the difference between \$s0 and \$s2 in \$t0, 'add' stores in \$t0 the previous value of \$t0 added to \$s1:

```
subtract $t0, $s0, $s2     # a = b - d
add $t0, $t0, $s1          # a = a + c  ⟹  a = b - d + c
```

c) In order to perform 3*b, since we can't use multiplication, we perform b+b+b:

```
add $t0, $s0, $s0     # $t0 = $s0 + $s0  ⟹  a = 2 * b
add $t0, $t0, $s0     # $t0 = $t0 + $s0  ⟹  a = 3 * b
```

d) In this case we have an addition with a constant, so we use add immediate, which stores in \$t0 the sum of \$s0 and 1:

```
addi $t0, 1, $s0      # a = 1 + b
add $t0, $t0, $t0     # $t0 = $t0 + $t0 ⟹ a = (1 + b) * 2
```

Problem 5.3

Solution:

a) According to the slides, we have:

op (6)	rs (5)	rt (5)	rd (5)	sahmt (5)	funct (6)
add (0)	\$s0 (16)	\$s1 (17)	\$t0 (8)	unused (0)	32
000000	10000	10001	01000	00000	100000

Where,

→ op (6) means op occupies 6 bits;

→ \$s0 (16) means \$s0 has register number 16.

The instruction code is: 000000 10000 10001 01000 00000 100000

b)

op (6)	rs (5)	rt (5)	rd (5)	sahmt (5)	funct (6)
sub (0)	\$s0 (16)	\$s2 (18)	\$t0 (8)	unused (0)	34
000000	10000	10010	01000	00000	100010
add (0)	\$t0 (8)	\$s1 (17)	\$t0 (8)	unused (0)	32
000000	01000	10001	01000	00000	100000

The instruction code is:

000000 10000 10010 01000 00000 100010

000000 01000 10001 01000 00000 100000

Problem 5.4

Solution:

We load the value from A[2] and A[4], store them in \$t0 and \$t1 registers, then perform the addition and store it in B[5]. When trying to access the array values, we multiply the position by 4.

```
lw $t0, 8($s0)       # $t0 stores temporarily the value of A[2]
lw $t1, 16($s0)      # $t1 stores temporarily the value of A[4]

add $t2, $t0, $t1    # sum of the values stored in $t0 and $t1 is stored in $t2
sw $t2, 20($s1)      # the sum A[2]+A[4]is now saved in B[5]
```

Problem 5.5

Solution:

First we find the correct addresses of B[x], A[x+7], and A[x+2], which are respectively: 4x+\$s1, 4(x+7)+\$s0, and 4(x+2)+\$s0

```
add $t1, $t0, $t0    # $t1 holds two times the value of $t0 → 2x
add $t1, $t1, $t1    # $t1 = $t1+$t1 → 4x
add $t1, $t1, $s1    # $t1 holds address of B[x]

addi $t2, $t0, 7     # $t2 = $t0+7 → x+7
add $t2, $t2, $t2    # $t2 = $t2+$t2 → 2(x+7)
add $t2, $t2, $t2    # $t2 = $t2+$t2 → 4(x+7)
add $t2, $t2, $s0    # $t2 holds address of A[x+7]
```

```
addi $t3, $t0, 2      # $t3 = $t0+2 → x+2
add $t3, $t3, $t3     # $t3 is duplicated → 2(x+2)
add $t3, $t3, $t3     # $t3 = $t3+$t3 → 4(x+2)
add $t3, $t3, $s0     # $t2 holds address of A[x+2]

lw $t4, 0($t2)        # load the value stored in $t2 (A[x+7]) and store it in $t4
lw $t5, 0($t3)        # load the value stored in $t3 (A[x+2]) and store it in $t5
add $t6, $t4, $t5     # $t6 = $t4 + $t5

sw $t6, 0($t1)        # $t6 = B[x] = A[x+7] + A[x+2]
```

Problem 5.6

Solution:
The new number of registers is now 16. Considering the fact that $1111_2 = 15_{10}$, if we include 0, we can see that we now need only 4 bits to represent the registers address and not 5. Therefore, we make the following changes:
→ op doesn't change (6 bits)
→ rs - 4 bits
→ rt - 4 bits
→ immediate - 18 bits (2 more bits than before)
(So that in total we still have 32 bits.)