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

Homework 5

Problem 5.1

Solution:

a) First, we convert the numbers to binary:
* $14_{10} = 1110_2$
* $37_{10} = 100101_2$
Now, we perform the addition:

$$\begin{array}{rr} 0000 & 1110 \\ +\,0010 & 0101 \\ \hline 0011 & 0011 \end{array}$$

In the end, we convert the sum in decimal:

$$\overset{543210}{110011} \rightarrow 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 32 + 16 + 2 + 1 = 51$$

Therefore we calculated: $14 + 37 = 51$ using binary representation of the numbers.

b) We convert the unsigned numbers to binary:
* $12_{10} = 1100_2$
* $27_{10} = 11011_2$

Since we have a subtraction case, we consider the operation as an addition between a positive and a negative number $(12 + (-27))$, so we need to find the binary representation of (-27) using 2's complement:
* We add leading zeros to complete 8 bits: $0001\,1011_2$
* We invert the bits: $1110\,0100_2$
* We add one to the resulting binary: $1110\,0100_2 + 1 = 1110\,0101_2$

We perform the addition:

$$\begin{array}{rr} 0000 & 1100 \\ +\,1110 & 0101 \\ \hline 1111 & 0001 \end{array}$$

We finally convert the number to decimal using 2's complement. Sign bit is 1, so we have the negative number case:
* We subtract 1: $1111\,0001_2 - 1 = 1111\,0000_2$
* Inversion: $0000\,1111_2$
* Normal conversion to binary:

$$\overset{3210}{1111_2} \rightarrow 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1 = 15 \rightarrow -15_{10}$$

So, the result is: $12 - 27 = -15$

c) We convert the numbers in BCD respresentation:
* $69_{10} = 0110\,1001_{BCD}$
* $58_{10} = 0101\,1000_{BCD}$

We can notice that when we add 69 and 58, both additions of 9 and 8, as well as of 6 and 5, create a carry, so we have to add 6 (0110_{BCD}) to both the left counterpart and the right counterpart of the initial result:

	0110	1001
	+ 0101	1000
	1100	0001
	+ 0110	0110
0001	0010	0111

Finally, we convert the found value to decimal representation:

$$0001\ 0010\ 0111_{BCD} = 127_{10}$$

d) Using the same logic as before:
* $275_{10} = 0010\ 0111\ 0101_{BCD}$
* $642_{10} = 0110\ 0100\ 0010_{BCD}$

A carry is created when adding 7 and 4, so we need to add a 6 (0110_{BCD}) in the corresponding place:

0010	0111	0101
+ 0110	0100	0010
1000	1011	0111
+	0110	
1001	0001	0111

We convert the result to decimal:

$$1001\ 0001\ 0111_{BCD} = 917_{10}$$

e) We perform the addition as follows:

6	A	F
+ 2	3	C
8	E	B

Explanation: F+C is equal to 27, which is bigger than 15, so we subtract 16 $(27-16 = 11 \rightarrow B)$, and carry 1 to the next position. Then, $1+A+3 = 14 \rightarrow D$, so no carry to the next position, which is $6 + 2 = 8$.

f) For the hexadecimal subtraction we can use the same method as for binary numbers (hex $\rightarrow$ binary $\rightarrow$ 2's complement $\rightarrow$ hex), but there is also a simpler method where we subtract each digit of the unsigned number from F, and then add 1 to the result:

F	F	F
- 3	- A	- 8
C	5	7
+		1
C	5	8

After this, we perform normal addition:

	5	9	4
+	C	5	8
1	1	E	C

So, what we have done is: $4 + 8 = 12 \rightarrow C$, then $9 + 5 = 14 \rightarrow E$, and finally $5 + C = 17 > 15 \implies$ a carry is created, so $17 - 16 = 1$ and 1 is the carry also and the result is 11EC. The carry is disregarded as the result of the addition of the 2's complement system must be the same size as the inputs'. Therefore, $594 - 3A8 = 1EC$.

An alternate way to find the result would be directly by subtracting the numbers:

5	9	4
- 3	A	8
1	E	C

The calculations are performed as follows: $4 - 8 < 0 \implies$ we borrow from the next column, which means reducing 1 from 9 and lending 1 (16 in hexadecimal) to 4. So, we have $4 + 16 - 8 = 12 \rightarrow C$. Then, moving to the next column, we have $8 - A < 0$, so again we borrow from 5, which becomes 4, and we have $8 + 16 - A = 14 \rightarrow E$. Finally, we have $4 - 3 = 1$, and the result is $1EC$.

Problem 5.2

Solution:

a)
```
add $t0, $s0, $s1    # a = b + c → a stores addition of b and c
```

b)
```
subtract $t0, $s0, $s2     # a = b - d → a stores difference of
                           # b and d first
add $t0, $t0, $s1          # a += c → a now stores b - d + c
```

c)
```
add $t0, $s0, $s0     # a = b + b = 2 * b
add $t0, $t0, $s0     # a += b → a stores b + b + b = 3 * b
```

d)
```
addi $t0, $s0, 1     # a stores 1 + b using add immediate
add $t0, $t0, $t0    # a = a + a → a stores 2 * a = 2 * (1 + b)
```

Problem 5.3

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
add	$s0	$s1	$t0	unused	-
0	16	17	8	0	32
000000	10000	10001	01000	00000	100000

Therefore, the MIPS instruction in binary is:
000000 10000 10001 01000 00000 100000

b) Using the same idea as before, we get:

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

Therefore, the MIPS instruction set is:
000000 10000 10010 01000 00000 100010
000000 01000 10000 01000 00000 100000

Problem 5.4

Solution:

```
lw $t0, 16($s0)     # the value stored in A[4] is loaded and stored
                    # in temporary register $t0
lw $t1, 8($s0)      # the value stored in A[2] is loaded and stored
                    # in temporary register $t1
add $t0, $t0, $t1   # addition of the values stored in temporary registers
                    # $t0 and $t1 is stored in $t0
sw $t0, 20($s1)     # the value stored in temporary register $t1→A[4]+A[2]
                    # is saved in B[5]

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

Problem 5.5

Solution:

```
# Calculate the address of A[x+7] → Derive correct offset for 4*(x+7)
addi $t1, $t0, 7    # store in temporary register $t1 the index [x+7]
add $t1, $t1, $t1   # duplicate value of $t1
add $t1, $t1, $t1   # duplicate value of $t1 again, so now we have
                    # stored in $t1 → 4*(x+7)
add $t1, $t1, $s0   # $t1 = address of A[x+7] → [4*(x+7) + $s0]

# Calculate the address of A[x+2] → Derive correct offset for 4*(x+2)
addi $t2, $t0, 2    # store in temporary register $t2 the index [x+2]
add $t2, $t2, $t2   # duplicate value of $t2
add $t2, $t2, $t2   # duplicate value of $t2 again, so now we have
                    # stored in $t2 → 4*(x+2)
add $t2, $t2, $s0   # $t2 = address of A[x+2] → [4*(x+2) + $s0]

lw $t3, 0($t1)      # $t3 → value stored in A[x+7]
lw $t4, 0($t2)      # $t4 → value stored in A[x+2]
add $t5, $t3, $t4   # $t5 → sum of the values stored in $t3 and $t4

# Calculate the address of B[x] → Derive correct offset for 4*x
add $t6, $t0, $t0   # store in $t6 the duplicated value of $t0
add $t6, $t6, $t6   # duplicate value of $t6, so now we have stored
                    # in $t6 → 4*x
add $t6, $t6, $s1   # $t6 = address of B[x] → [4*x + $s1]

sw $t5, 0($t6)      # the value stored in temporary register
                    # $t6 → A[x+7] + A[x+2], is saved in B[x]
```

Problem 5.6

Solution:

Since the number of purpose registers is reduced to 16, we need only 4 bits to represent their address. We conclude this from: $1111_2 = 15_{10}$ (including zero = 16), so we can express all of our registers. The 6 bits for the op code should not change, so since the registers bits are reduced from 5 to 4, we add 2 extra bits to the constant value, so that the register size will still be 32 bits.