Homework - Computer System Architecture – Lab 2

Read the website summary of lab 2 on simple arithmetical instructions and fill in the blanks.

A. Additions, subtractions

1. The instruction used to add two numbers is add, and it takes 2 operands. The result is stored in the first operand, for which reason this operand can never be a constant. The two operands must have the same type (byte, word or doubleword) and can be ~~both~~ registers, but we can't be both memory locations at the same time. The same is true for the subtraction instruction named sub.
2. The instruction for incrementing is called inc and has 1 operands, which can be a register or memory location of any size. The result is stored in the same operand. The instruction dec has one operand and will perform decrementation (-1). The instruction neg has one operand and will store the result in the operand, which is computed (<reg>/<mem> ← 0 - <reg>/<mem>) 0-operand

B. Multiplications, divisions

1. The unsigned multiplication instruction is called mul and has only one implicit operand which can be specified, the second operand being fixed (explicit), having (binary) the same size: AL, AX or EAX. The result is stored on a double size of the operands in a fixed location: 8, 16 and respectively 32 bits. (register/memory loc)
2. Instruction div performs unsigned division and takes one explicit operand, which can be a register or a memory location, but not an immediate value (i.e. a constant). The implicit operand is double size of the explicit operand and it is fixed, so for operand size byte it is AL, for word it is AX and for doubleword it is EAX. The division has two results, the quotient and the reminder, stored in AL and AH, AX and DX and respectively EAX and EDX. If the result does not fit, the program will ~~become a loop~~ throw an error, as if it divided by zero (division overflow).

C. Declaring variables, constants

1. In assembly, we can declare variable with initial value, for which purpose we use directives DB, DW, DD, DQ. For example, if we want to declare a byte a1 initialized with the hexa value *Ah* it would be a1 DB 0Ah. Or, if we want to declare a2 as several bytes containing the first 5 lowercase letters of the alphabet, it would be a2 DW 'a''b''c''d''e'. Or, to declare a variable a3 containing a word initialized with decimal value 20, it would be a3 DW 20.
2. If we just want to reserve some space for variable, without providing initial values, we use directives RESB, RESW, RESQ. So, to reserve in variable a4 30 words, we would write a4 RESW 30. Constants are declared using keyword EQU. For example, we declare constant a5 with value 1000 a5 EQU 1000.

My questions: _____

homework : Lab 2 - prob 18 (Multiplications, divisions)

how the memory looks like

13AB h → AB 12 10 FF C8 28

~~FF10~~ → 10 FF

FF10

memory

00 00 0218 27AA 3F25 h → 25 3F AA 27 18 02 00 00 07

db 7

hw - pb 18 - sect 3 (L2) - diff prog/exercise

read L3

Homework - Computer System Architecture – Lab 3

Read the website summary of lab 3 on signed arithmetical instructions and answer the following questions.

A. Signed additions, subtractions

a. State one similarity and one difference between *add* and *adc*.

S: For both instructions, the operands should have the same type (b, w, dw)

D: For ADC the CF is added to the sum of the 2 operands

b. What is the meaning of the CF for the *sbb* operation? substract with borrow

The value of the carry flag is substracted from (dest - source)

meaning: borrow digit

B. Signed multiplications, divisions

a. State one difference and one similarity between *mul* and *imul*.

D: imul → signed integers, mul - unsigned integers

S: the explicit operand can be a register or a variable, but it cannot be an immediate value (a constant)

b. Can we divide two bytes? Why yes/not?

We cannot divide 2 bytes, because the size of the first operand must be double the size of the second operand

C. Conversions

a. Match conversions from the two columns, complete those missing (...)

unsigned: complete the rest with 0

EAX to EDX:EAX unsigned mov edx, 0	mov ab, 0
EAX to EDX:EAX signed cdq	cwd
AX to EAX signed cwde	cdq
AX to DX:AX signed cwd	mov edx, 0
AX to EAX unsigned	mov dx, 0
AL to AX unsigned mov AH, 0	cwde → word → double extended
AL to EAX unsigned	...
AL to AX signed cbw	cbw
AX to DX:AX unsigned ~~cdq~~ mov dx, 0	...

b. Why do we need conversions? Give an example.

My questions: _____

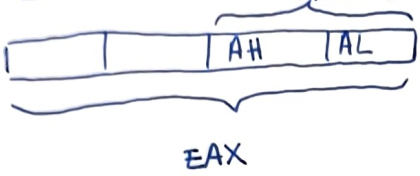
_____

Laboratory - ASC - 12.10.2022

Registers

EAX = 32 bits = 4 byte

AX = 2 bytes = word



EBX, ECX, EDX

byte : AH, CL, DH

word : CX, DX, BX

doubleword : EAX, ECX, EDX

mul BL

BL · AL → AX

✓ fixed

CX · AX → DX · AX

EBX : EAX → EDX : EAX

	quotient	remainder
div BL; AX / BL →	AL	AH
CX; DX : AX / CX →	AX	DX
EBX; EDX : EAX / EBX	EAX	EDX

AX + BX

AH : AL BH : BL

AH : AL +
BH : BL
AH+BH+CF AL+BL

add AL, BL
adc AH, BH

(an intermediate operation might change the CF)
(they are used together)

adc set the carry flag ← 1 / 0

DL : AH : AL
CL : BH : BL

adc adc add CF
↳CF ↳CF

100 - 11 = 89

sub AL, BL → set CF ← 1 / 0
sbb AH, BH → - CF

2W + 2W

we split it in EAX and EDX

11 22 33 44 55 66 77 88

EDX : EAX
EBX : ECX

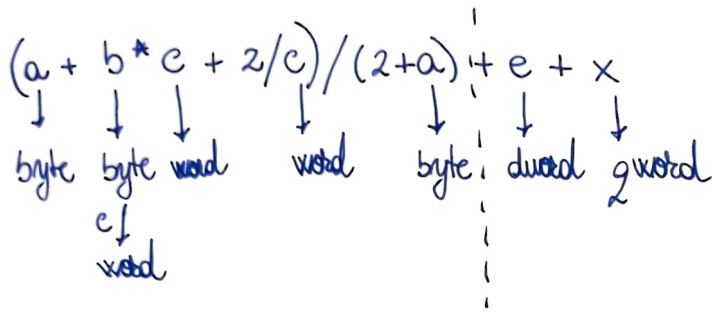
adc add

low part

88 77 66 55 44 33 22 11
a a+1 a+4 a+7

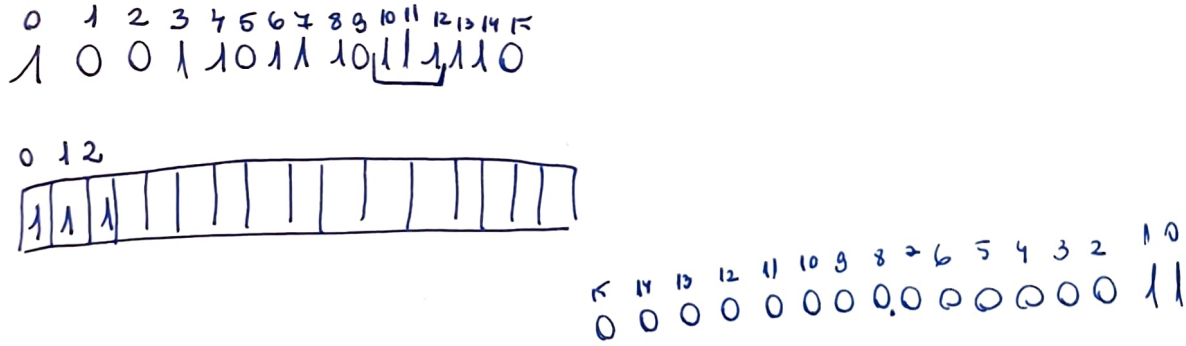
adress representation (debugger)

mov EAX, [a] ; EAX = 55 66 77 88 h
mov EDX [a+4] ; EDX = 11 22 33 44 h



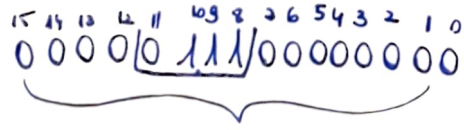
2/c ⇒ 2 → dword

c
word



0 00 1 1 1. 0000000000

0 0000000000 000 111



for small_pos − pos
(A) (B)

Homework - Computer System Architecture – Lab 4

Read the website summary and watch the youtube video of lab 4 on bitwise operations and answer these questions:

1. Name two instructions that do the same thing and explain why.
   SHR and SAL
   The bits stored in destination are shifted number positions (modulo 32) to the right. The leftmost bits are filled with 0. The last disappearing bit is kept in CF.
2. Which instructions use the value of the CF?
   SHL, SHR, SAL, SAR, ROL, ROR, RCL, RCR
3. Which instructions are using the value of the sign bit?
   SAR
4. Which instructions lose information? Explain why.
   AND, OR, XOR
   → they are storing the result in the first operand
5. What is the effect of the instruction *test*? Give an example.
   → executes the logical operation AND, without storing the result in the first operand
   TEST AH, 0110h
6. Which instruction do we use to select specific bits from a number? Give an example.
7. Which instruction do we use to place specific bits inside a number? Give an example.

My questions: _____

_____

LABORATORY 1 – HOMEWORK

1. Convert the following numbers from base 10 to 2 and then to 16:

$4_{(10)} = 100_{(2)} = 4_{(H)}$

$10_{(10)} = 1010_{(2)} = A_{(H)}$

$15_{(10)} = 11111_{(2)} = F_{(H)}$

$32_{(10)} = 100000_{(2)} = 20_{(H)}$

2. Convert the following numbers from base 10 to 16 then to 2:

$3_{(10)} = 3_{(H)} = 11_{(2)}$

$11_{(10)} = B_{(H)} = 1011_{(2)}$

$16_{(10)} = 10_{(H)} = A_{16} \quad 10000_{(2)}$

$17_{(10)} = 11 = B_{(H)} = 10001_{(2)}$

3. Convert the following numbers from base 2 to 16:

$1010_{(2)} = A_{(H)}$

$0111_{(2)} = 7_{(H)}$

$1111_{(2)} = F_{(H)}$

$1000\ 1010_{(2)} = 8A_{(H)}$

$11010\ 1111_{(2)} = 1AF_{(H)}$

4. Convert the following numbers from base 16 to base 2:

$3_{(H)} = 11_{(2)}$

$A_{(H)} = 1010_{(2)}$

$F_{(H)} = 1111_{(2)}$

$2B_{(H)} = 101011_{(2)}$

$2F8_{(H)} = 10\ 1111\ 1000_{(2)}$

5. Compute the following expressions directly in base 2 (without converting to base 10)

$1+1 = 10$

$10_{(2)} + 10_{(2)} = 100_{(2)}$

$111_{(2)} + 1_{(2)} = 1000_{(2)}$

$1010_{(2)} - 1_{(2)} = 1001_{(2)}$

$1000_{(2)} - 10_{(2)} = 0110_{(2)}$

$\begin{array}{r} {\scriptstyle 1}10 + \\ 10 \\ \hline 100 \end{array}$ $\quad \begin{array}{r} {\scriptstyle 11}111 + \\ 1 \\ \hline 1000 \end{array}$ $\quad \begin{array}{r} 1010 - \\ 1 \\ \hline 1001 \end{array}$ $\quad \begin{array}{r} 1000 - \\ 10 \\ \hline 0110 \end{array}$

6. Compute the following expressions directly in base 16 (without converting to base 10)

$9_{(H)} + 1_{(H)} = A_{(H)}$

$B_{(H)} + 2_{(H)} = D_{(H)}$

$F_{(H)} + 1_{(H)} = 10_{(H)}$

$10_{(H)} + A_{(H)} = 1A_{(H)}$

$10_{(H)} - 2_{(H)} = D_{(H)}$

$B_{(H)} - 3_{(H)} = 8_{(H)}$

$\begin{array}{r} {\scriptstyle 1}A_{(H)} + \\ A_{(H)} \\ \hline 1A_{(H)} \end{array}$

$16 + 10 = 26$

$26 : 16 = 1$ r $10 = 1$ r A

16

10

7. Check, using at least two of the complementary code rules, if:

$(9A7D)_{(16)}$ and $(7583)_{(16)}$ are complementary in a location of 2 bytes

I $\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1}9A7D + \\ 7583 \\ \hline 10000 \end{array}$

II $\begin{array}{r} 1001\ 1010\ 0111\ 1101 \\ 0111\ 0101\ 1000\ 0011 \\ \hline 0001\ 0000\ 0000\ 0000\ 0 \end{array}$ not complementary

I $\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1}000F095D + \\ FFF0F6A3 \\ \hline 100000000 \end{array}$

II $\begin{array}{r} 0000\ 0000\ 0000\ 1111\ 0000\ 1001\ 0101\ 1101 \\ 1111\ 1111\ 1111\ 0000\ 1111\ 0110\ 1010\ 0011 \\ \hline 1\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array}$

complementary

I $\begin{array}{r} {\scriptstyle 1\ 1}43A1 + \\ 5C93 \\ \hline A834 \end{array}$

II $\begin{array}{r} 0100\ 1011\ 1010\ 0001 \\ 0101\ 1100\ 1001\ 0011 \\ \hline 1010\ 1000\ 0011\ 0100 \end{array}$ not complementary

I 1
   7F +
   81
   100

II 1111 1111
    0111 1111
    1000 0001
   10000 0000

complementary

I 1
   432A +
   4358
   B682

II 1   11 1111
    0111 0011 0010 1010
    0100 0011 0101 1000
   1011 0110 1000 0010

not complementary

8. Write the 8 bits unsigned representation for the following numbers:

8 → 0000 1000

67 → 01000011

230 → 11100110

9. Write the 16 bits signed representation for the following numbers

-6

~~0000 0000 0110~~

6 = 0000 0000 0000 0110 (2)

-6 = 1111 1111 1111 1010 (2)

-121

~~121 = 0000 0111 1001~~

121 = 0000 0000 0111 1001

-121 = 1111 1111 1000 0111

70 = 0000 0000 1000 0110

① CH = ?

mov ECX, -1 << 12

1 = 0000 0000 0000 0001

1 : 0000 0001 ⇒ -1 = 1111 1111 = FFFFFFFF

-1 << 12 : ECX : FFFFF000

CH

② mov ax, 400h

mov bl, 0FEh → new value (signed)

idiv bl

~~256·256~~

~~256·4 : 15·16 = 64:15~~

16

256·4 = 1024

15·16 + 14 = 254

} 1024 : 254 = 4 r 8

③ a dd 0a344dh, 30h, 11223344h, 46ab89ch (dd from offset 9)

0 1 2 3 4 5 6 7 8 9

44 a3 00 00 30 00 00 00 44 33 22 11 9c b8 6a 04

~~99 11 22 33~~

0 1 2 3 4 5 6 7 8 9

44 a3 00 00 30 00 00 00 44 33 22 11 9c b8 6a 04

9c 11 22 33

④ a times 2 dd 1234h

b db 16h (byte from offset 6)

~~34 12 00 34 12 00~~

0 1 2 3 4 5 6

34 12 00 00 34 12 00 00 16

⑤ mov al, -2

mov bl, -128

imul al mul bl !!! ⇒ (-2) * (-2) = 4 = 1005

⑥ x dw 0ffffh

$\begin{array}{r|l} 16 & \\ 8 & \\ \hline 128 & 4 \end{array}$

mov ah 80h

mov bh -128

add ah, bh

2 interpretări → signed : { ah = 80h = 128(u) = −128(s) ∈ [-128,127] ; bh = −128 } ⊕ ah = −256 ∉ [-128,127]

→ unsigned

~~ah = 128~~

~~bh = 128~~

⇒ { CF = 1, SF = 0, OF = 1

mov ah, 80h // ah = -128 (128 ∉ [-128,127] → 256) 1000 0000

mov bh, -128 // bh = -128 ∈ [-128,127] 1000 0000

add ah, bh

1 0000 000 +
1 0 0 0 0 0 0 0
1 | 0 0 0 0 0 0 0 0

{ CF = 1 (peste ordin)
SF = 0 (în rep. tuneliată = 1 byte)
OF = 1 (rez. nu încape)