|  |  |
| --- | --- |
| Student name and surname |  |

EFREI CA2023F

Computer Architecture – Assembly language programming

# Arithmetic operations

x86 CPU can execute several arithmetic instructions:

|  |  |
| --- | --- |
| Mnemonic | Operation |
| ADD src, dest | dest=dest+src |
| SUB src,dest | dest=dest-src |
| DEC dest | dest=dest-1 |
| INC dest | dest=dest+1 |
| NEG dest | dest= -dest |
| MUL/IMUL src | %eax=%eax\*src ; src can only be a register |
| DIV/ IDIV src | %eax = (%edx,%eax)/src; %edx =(%edx,%eax)modulo src |
| CMP src, dest | dest- src; result not stored, only condition flags set |

Almost all of them set condition flags based on the result of their execution in EFLAGS register.

## EFLAGS register

This is a 32-bit control-status register that contains current state of the x86 CPU. Structure of this register is shown on image below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ID | VIP | VIF | AC | VM | RF | 0 | NT | IOPL | | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

Bits marked in orange are named *condition codes* and are affected by the result of execution almost all arithmetic instructions. The meaning of certain bits is as follows:

CF – result of last operation caused carry (the result is larger than its destination)

ZF- result was 0 (zero)

SF- result was negative

OF – result caused arithmetic (signed)overflow.

## Simple equation calculation

Analyze and execute example that calcualte the value of given expression. It’s source code is available in **calculate.asm** file.   
It calculates the value of equation: *b = 4a^2 + 3a -2* using integer numbers only.

## Division and accuracy

Let’s consider two very similar equations:

***a=(7/5)\*5***

***b=(7\*5)/5***

When we calculate the value of this equations in rational or real numbers domain, the result of the equation **a** and **b** is the same. But if the calculations is made in integer numbers domain, the results will be different because the result 7 divided by 5 in integer numbers domain is 1, so the final result of equation ***a*** is 5. The value of equation **b** is 7.

There is a sample program (available in **calculate2.asm**) that shows how to use division instruction and illustrates the problem of accuracy. Try to run them with different values and observe the results.

## Assignment – calculating arithmetic equation

Write a program that evaluate one of the following expressions (in integer numbers domain):

1. ((2x^2+ 3y)/2 +4z)\*3
2. (4x^3+ 2y)/6z^2
3. 2x^3+ 3y^2-7
4. (5x+4)/2 + 3y

Chose only one formula: a,b,c,or d. The formula should be chosen as a reminder of division your birthday by 4 (if the reminder is 0 you chose formula a), if the reminder is 1 you chose formula b), etc.).

Program should prompt user for require values, calculate the result and print them back to the user. Result should be as accurate as possible.

Upload source code of a properly working program on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the source code (Author section).

# Conditional instructions

Programs written in high-level languages ​​contain code fragments that are executed conditionally. They are built using well-known if-then, if-then-else, switch-case, etc. constructions. In assembly language, such constructions are created using conditional instructions. These types of instructions include, for example, branch instructions.

## Branch instructions

Almost all modern processors have instructions that allow programmer to change the order in which the other instructions are executed. A group of this instructions is called *branch* or *jump* instructions. In this group, two major types can be distinguished: instructions that are always executed and those that are executed under certain conditions. This second group of the instructions uses the **condition flags** described in section 1.1 to decide whether or not to execute the jump.

x86 CPU offers a rich set of conditional branch instructions:

JA - Jump if above (CF=0 and ZF=0)

JAE - Jump if above or equal (CF=0)

JB - Jump if below (CF=1)

JBE - Jump if below or equal (CF=1 or ZF=1)

JC - Jump if carry (CF=1)

JCXZ - Jump if CX register is 0

JECXZ - Jump if ECX register is 0

JE - Jump if equal (ZF=1)

JG - Jump if greater( ZF=0 and SF=OF)

JGE - Jump if greater or equal (SF=OF)

JL - Jump if less (SF<>OF)

JLE - Jump if less or equal (ZF=1 or SF<>OF)

JNA - Jump if not above (CF=1 or ZF=1)

JNAE - Jump if not above or equal (CF=1)

JNB - Jump if not below (CF=0)

JNBE - Jump if not below or equal (CF=0 and ZF=0)

JNC - Jump if not carry (CF=0)

JNE - Jump if not equal (ZF=0)

JNG - Jump if not greater (ZF=1 or SF<>OF)

JNGE - Jump if not greater or equal (SF<>OF)

JNL - Jump if not less (SF=OF)

JNLE - Jump if not less or equal (ZF=0 and SF=OF)

JNO - Jump if not overflow (OF=0)

JNP - Jump if not parity (PF=0)

JNS - Jump if not sign (SF=0)

JNZ - Jump if not zero (ZF=0)

JO - Jump if overflow (OF=1)

JP - Jump if parity (PF=1)

JPE - Jump if parity even (PF=1)

JPO - Jump if parity odd (PF=0)

JS - Jump if sign (SF=1)

JZ - Jump if zero (ZF=1)

Those instructions, in connection witch arithmetic instructions (especially cmp) are used to create conditionally executed blocks of assembly code, the operation of which is similar to the conditional constructs known from high-level languages such as *if-then*, *if-then-else*, etc.

## Simple if-then statement

Source code of an example program that uses simple *if-then* statement is available in **condition1.asm** file.

The program asks user for input an integer number ( it may be positive or negative), then checks if the user number is less than zero. If the number is less than 0, a proper message informs user about it.

There is part of the source code that check the condition

First, the user value must be placed into register because scanf() function put them into memory at label userval:

**…**

**movl (userval),%eax # put user number into %eax**

Next we may check if te userval is less or greater than 0. We use **cmpl** instruction to do this

**# if (userval >= 0 )**

**cmpl $0,%eax # compare %eax with 0**

cmpl instruction substracts from 0 value of %eax . If %eax contains number greater than 0 the result will be a negative number and the SF conditional flag will be set after this operation.

**jge end\_if # if %eax is greater or equal from 0 -> goto end\_if**

jge instruction checks the value of SF flag and if it’s set, executes the jump to the first instruction after label end\_if. If the SF is not set, the user value is less than 0 and proper message is printed.

**# now we are know that the user value is less than 0**

**# printing a message that tells the userval is less than zero.**

**…**

**end\_if:**

**…**

## If-then-else statement

The program form section 2.3 can be easy expanded to demonstrate full if-then-else statement. Try to do necessary modifications so that the example program also displays information that the entered number is greater or equal zero.

## Conditional instructions assignment

Write a program that reads a single character from user and then checks it. If the character is a number (0-9), it prints it in return without change. If the character is uppercase (A-Z), converts it to lowercase and prints it. If the character is lowercase (a-z), converts it to uppercase and prints it as well.

Upload source code of a properly working program on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the source code (Author section).