**Seminar I. Introduction to the IA-32 assembly language**

The IA-32 computing architecture is the microprocessor (CPU) architecture introduced by the Intel Corporation in 1985 for their 80386 microprocessor. It is an abstract model of a microprocessor specifying the microprocessor’s elements, structure and instruction set. The IA-32 is a 32-bit computing architecture (basically meaning that its main elements have 32 bits in size) and it is based on the previous Intel 8086 computing architecture.

**I.1. The elements of the IA32 assembly language**

An **algorithm** is, as you well know, a sequence of steps/operations necessary in order to solve a specific (mathematical or not) problem. For example, the algorithm for solving the 2nd degree algebraic equation *a*\**x2*+*b*\**x*+*c*=0 contains the steps:

1) compute the value of *delta*

2) if *delta* is greater or equal to zero, compute the solutions *x1* and *x2* using the well-known formulas.

But besides this sequence of steps/operations, al algorithm also includes a set of data/entities on which those steps/operations operate. For our example, the data of the algorithm is: *a, b, c, delta, x1 and x2*.

So, an algorithm is two things:

* a set of data/entities
* and a sequence of operations/steps

An algorithm can be described using the natural language (e.g. romanian language, English language etc.) or it can be described in a programming language (e.g. C, Java, python, php etc.). When an algorithm is specified in a programming language, we refer to this algorithm as a **program.** In a similar way, a program contains: a) a set of data/entities and b) a set of operations/instructions.

Throughout the semester we will study the IA-32 assembly language. We will first describe the data part of the assembly language and later the operational part (instructions). All data used in an IA-32 assembly program is essentially numerical (integer numbers) and can have 3 basic types:

* byte – that data is represented on 8 bits
* word – that data is represented on 16 bits
* doubleword – that data is represented on 32 bits

In the IA-32 assembly language we have data that changes its value throughout the execution of the program (i.e. constant data or constants) and data that does not change its value throughout the execution of the program (i.e. variable data or variables).

**I.1.1 Constants**

We have 3 types of constants in the IA-32 assembly language:

* numbers (natural or integer):
  + written in base 2; ex.: 101b, 11100b
  + written in base 16; ex.: 34ABh, 0ABCDh
  + written in base 10; ex.: 20, -114
* character; ex.: ‘a’, ‘B’, ‘c’ ..
* string (sequence of characters); ex.: ‘abcd’, “test” …

**I.1.2 Variables**

The IA-32 assembly language has 2 kinds of variables: pre-defined variables and user-defined variables. A variable has a name, a data type (byte, word or doubleword), a current value and a memory location (where the variable is stored).

Pre-defined variables (CPU registers):

The CPU registers are memory areas located on the CPU which are used for various computations. The IA-32 CPU registers are:

1) General registers (each register has 32 bits in size):

* EAX (the lower or least significant part of EAX can be referred by AX and AX is formed by two 8-bit subregisters, AL and AH)
* EBX (the lower or least significant part of EBX can be referred by BX and BX is formed by two 8-bit subregisters, BL and BH)
* ECX (the lower or least significant part of ECX can be referred by CX and CX is formed by two 8-bit subregisters, CL and CH)
* EDX (the lower or least significant part of EDX can be referred by DX and DX is formed by two 8-bit subregisters, DL and DH)
* ESP (the lower or least significant part of ESP can be referred by SP)
* EBP (the lower or least significant part of EBP can be referred by BP)
* EDI (the lower or least significant part of EDI can be referred by DI)
* ESI (the lower or least significant part of ESP can be referred by SI)

2) Segment registers (each register has 16 bits):

CS, DS, SS, ES, FS, GS – are not used in a program

3) Other registers (32 bit registers): EIP and Flags.

User-defined variables:

For these variables, the programmer has to define the name, data type and initial value.

Examples:

1) *a DB 23*  : defines the variable with the name “a”, data type byte (DB-Define Byte) and initial value 23

2) *a1 DW 23ABh*  : defines the variable with the name “a1”, data type word (DW-Define Word) and initial value 23ABh

3) *a12 DD -101*  : defines the variable with the name “a12”, data type doubleword (DD-Define DoubleWord) and initial value -101.

**I.1.3 Instructions**

**MOV** – assignment instruction

*Syntax*: mov dest, source

(where dest and source are either registers, variables or constants of type byte, word or dword; dest can not be a constant)

*Effect*: dest := source

*Examples*: mov ax, 2

mov [a], eax

**ADD** – addition instruction

*Syntax*: add dest, source

(where dest and source are either registers, variables or constants of type byte, word or dword; dest can not be a constant)

*Effect*: dest := dest + source

*Examples*: add bx, cx

add [a], 101b

**SUB** – substraction instruction

*Syntax*: sub dest, source

(where dest and source are either registers, variables or constants of type byte, word or dword; dest can not be a constant)

*Effect*: dest := dest - source

*Examples*: sub ax, 2

sub [a], eax

**I.2. The 1st 32bit 8086 assembly language program**

;

; Comments are preceded by the ‘;’ sign. This line is a comment (is ignored by the assembler)

; This program computes the expression: x:= a + b – c = 3 + 4 – 2 = 5.

;

;

bits 32

; declare the EntryPoint (a label defining the very first instruction of the program)

global start

; declare external functions needed by our program

extern exit ; tell nasm that *exit* exists even if we won't be defining it

import exit msvcrt.dll ; *exit* is a function that ends the calling process. It is defined in msvcrt.dll

; our data is declared here (the variables needed by our program)

segment data use32 class=data

; ...

a dw 3

b dw 4

c dw 2

x dw 0

; our code starts here

segment code use32 class=code

start:

mov ax, [a]

add ax, [b]

sub ax, [c]

mov [x], ax

; *exit*(0)

push dword 0 ; push the parameter for *exit* onto the stack

call [exit] ; call *exit* to terminate the program