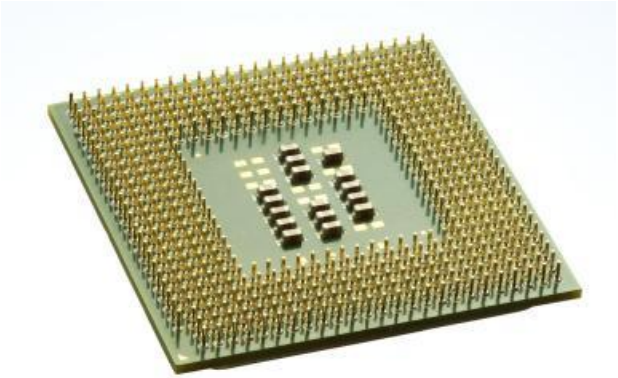


VILNIUS GEDIMINAS
TECHNICAL UNIVERSITY
FACULTY OF FUNDAMENTAL SCIENCES

IA-32

LECTURER: DR. PAVEL STEFANOVIČ

VILNIUS, 2017

6 laboratory work (1)

- Knowing the length of a given *C* string (all characters are numbers), create assembly code that would convert the string to integer representation.
- String length calculation and input/output operations to be done in *C* language for simplicity.
- String, given as command line parameter, will consist of *ASCII* characters, so each character to be subtracted '*0x30*' hex code (*ASCII '0'*).

6 laboratory work (2)

- Work can be done in 2 ways:
  - Easy way: string has fixed 4 character length. You can do it without any loop (**0.1 points**).
  - Hard way: string has variable length (10 characters max) (**0.2 points**).
- To understand how to make loops, add, multiply, etc. in assembler, you can find examples in the *Moodle*.

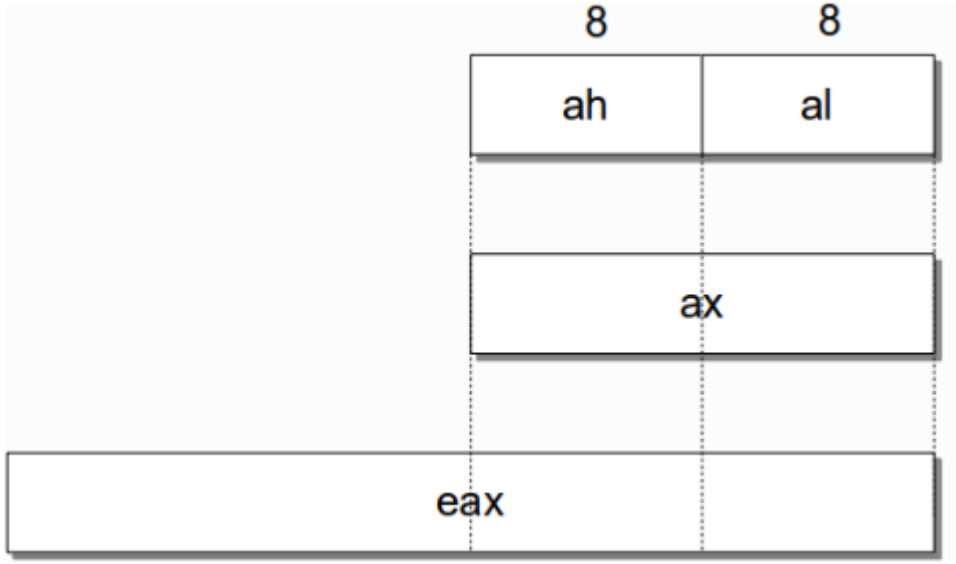
IA-32 microprocessor registers (1)

The *IA-32* processors provides four 32-bits data registers, they can be used as:

- Four 32-bits registers (*EAX, EBX, ECX, EDX*)
- Four 16-bits registers (*AX, BX, CX, DX*)
- Eight 8-bits registers (*AL, AH, BL, BH, CL, CH, DL, DH*)

Return Value Data Type	**Is Saved in Register**
char	AL
short (16-bit)	AX
int (32-bit)	EAX

IA-32 microprocessor registers (2)



Arithmetic

Instruction	Example	Meaning
add	add eax, ebx	eax = eax + ebx
subtract	sub eax, ebx	sub = eax + ebx
multiply	mul ebx	edx:eax = eax × ebx
multiply	imul eax, ebx	eax = eax × ebx
divide	div ebx	edx:eax = eax / ebx

Logical

Instruction	Example	Meaning
and	and eax, ebx	eax & ebx
or	or eax, ebx	eax | ebx
xor	xor eax, ebx	eax ^ ebx
assign	mov eax, ebx	eax = ebx
push	push eax	creating register
pop	pop eax	cleaning register

Other

Instruction	Example	Meaning
cmp	cmp eax, ebx	compare
increment	inc eax	eax = eax + 1
decrement	dec eax	eax = eax - 1

http://webcompiler.cloudapp.net/

Frame (1)

- Frame for those who using *Microsoft Visual C++*:
- *MS* compiler does not optimize register usage, so before entering and leaving *asm* block you have to save registers. Easiest save is to the stack using *PUSH* and *POP* (performed in "mirrored" sequence).

```
#include <stdio.h>
int main(int argc, char** argv ) {

        int   iOut = 0;
        char* pcInp;

        if( argc < 2 ) {
                printf("Missing parameter: number\n");
                return(0);
        }

        pcInp = argv[1];

          __asm {
                push eax
                push ebx
                push ecx
                push edx

                /* put code here */

                pop edx
                pop ecx
                pop ebx
                pop eax
        }

        printf("The number was processed as %d\n", iOut );
}
```

Frame (2)

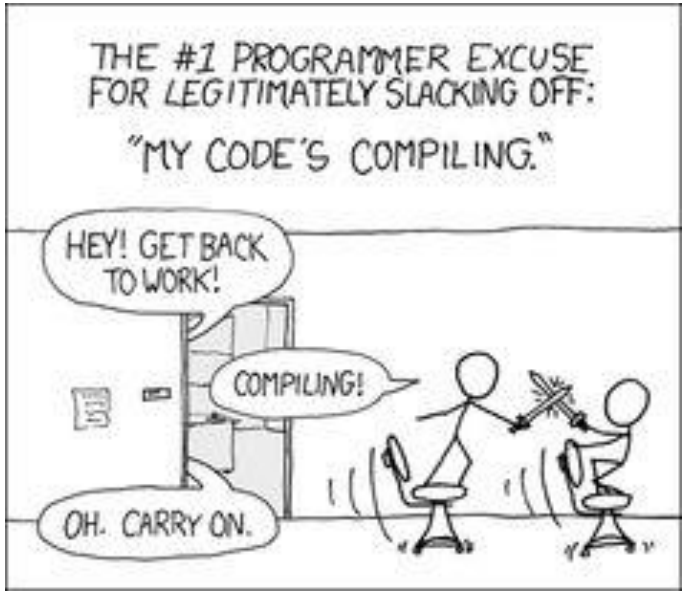
- Frame for those who will use *gcc/mingw Dev C++* (next slide):
- *Gcc* compiler uses *gcc* assembly syntax by default ("percentage" style syntax). You may want to switch on Intel syntax, then pass "*-masm=intel*" to the compiler.
- You also may want to compile 32 bit if you run 64 bit app. In this case use compiler switch "*-m32*"
- Full compiler launch command line sample:

gcc sample.c -masm=intel -m32 -o sample

Frame (3)

```
#include <stdio.h>
int main(int argc, char** argv ) {


      int   iOut = 0;
      char* pcInp = argv[1];

      if( argc < 2 ) {
            printf("Missing parameter: number\n");
            return(0);
      }

      // use %0 for iOut and %1 for pcInp
       asm (
            /* put code here */

            : "=m" (iOut)
            : "m" (pcInp)     // gcc inline asm input specification
                              // "m" is memory (pointer)
            : "eax", "ebx", "ecx", "edx"    // clobbered registers
                                            //(that are changed inside asm() )
        );

      printf("The number was processed as %d\n", iOut );
}
```

Example




```
#include <iostream>
using namespace std;

int main()
{
    int result;  // final result
 __asm {
    push eax     // creating register
    mov eax, 0   // assign eax = 0
    for_loop:     // loop name
        cmp eax, 10      // if eax = 10
        je exit_loop     // if true, exit the loop
        inc eax          // eax++
    jmp for_loop         // back to loop
    exit_loop:           // exit loop
        mov result, eax  // result = eax
    pop eax              // cleaning register
 }
 printf("Result=%d\n",result);    //showing the result
 }
```