Assembler

5 laboratory work

Lecturers: dr. Pavel Stefanovič, Rokas Štrimaitis, dr. Tomas Petkus

The main aim of laboratory work

- To remember the IA-32 assembler commands learned in the first course.
- To learn how to use SSE instructions.
- Improving programming skills.

5 laboratory work (I part)

(until *2019.12.20*)

Why?

- Knowing the length of a number string (in which all ASCII characters are numbers, maximum number **10 characters**), you have to create a program using IA-32 commands, that converts a number string to its numeric decimal equivalent.
- For simplicity, the calculation of the string length and the output of the result can be done in C language.

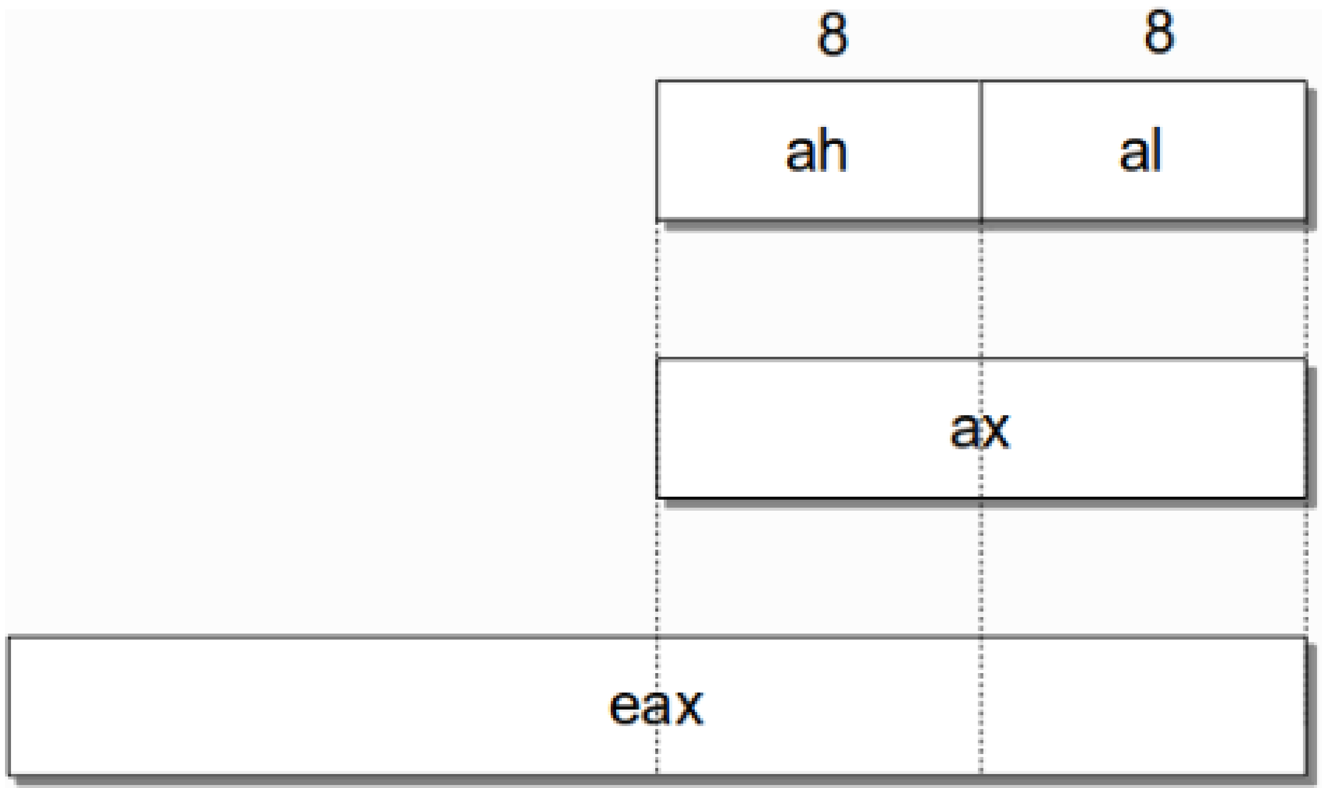
STRING “5245345” → INT 5245345

IA-32 registers (1)

- In the *IA-32* processor there are 32 bits registers, which can be used for:
  - 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*).

Data type	Saved in:
char (8 bits)	AL
short (16 bits)	AX
int (32 bits)	EAX

IA-32 registers (2)



Main instructions

Command	Example	Value
Addition	add eax, ebx	eax = eax + ebx
Subtraction	sub eax, ebx	eax = eax - ebx
Multiplication	mul ebx	edx:eax = eax × ebx
Multiplication	imul eax, ebx	eax = eax × ebx
Division	div ebx	edx:eax = eax / ebx
Logical AND	and eax, ebx	eax & ebx
Logical OR	or eax, ebx	eax | ebx
Logical XOR	xor eax, ebx	eax ^ ebx
Assign	mov eax, ebx	eax = ebx
Compare	cmp eax, ebx	Compare values between registers
Increase by one	inc eax	eax = eax + 1
Decrease by on	dec eax	eax = eax - 1

Program frames which can be used

```
#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 );
}
```

MSVC

```
#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 );
}
```

GCC / DEV C++

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
 }
```

More examples are in the „MOODLE“

5 laboratory work (II part)

- Numbers x, y and z are Pythagoreans numbers, if the condition is met:

$$x^2 + y^2 = z^2$$

- Your task is to find all integer Pythagorean numbers where:

$$x, y \in \{1,\ldots,1000\}.$$

- You must use SSE instructions to perform checking are the x, y integer Pythagorean numbers.
- Looping, output may be done in C.

SSE komandos

- The advantage of SSE instruction is that its is possible to make calculation parallel.

SSE															
R_4				R_3				R_2				R_1			
8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}	8_{bits}

- All instruction are given in the link below.

https://en.wikipedia.org/wiki/X86_instruction_listings

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

SSE realization

1	2	3	4	xmm0	First 4 values
2	3	4	5	xmm1	Second 4 values
1	4	9	16	xmm2	Squares of first
4	9	16	25	xmm3	Squares of second
5	13	25	41	xmm4	Sum of squares
2,23607	3,60555	5	6,40312	xmm5	Square root of previous sum
2	3	5	6	xmm5	integer of previous square
4	9	25	36	xmm5	square of previous integer
-1	-4	0	-5	xmm5	difference or comparison of xmm5 and xmm4

Found !

Example of the roots calculation

```
int main ( int argc, char** argv) {

        __declspec(align(16))float fmas[16]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
        __declspec(align(16))float fgmas[16];
        int imsize = sizeof(fmas)/sizeof(float);

        float* fptr;
        float* fgptr;

        for ( int i = 0; i < imsize; i+=4){
                fptr = fmas + i;
                fgptr = fgmas + i;
                __asm{
                        mov eax,fptr
                        movaps xmm0,[eax]
                        sqrtps xmm0,xmm0
                        mov eax,fgptr
                        movaps [eax],xmm0
                }
        }
        for ( int i = 0; i < 16; i++) {
                printf("Squere from %.0f is equal to %.20f\n", fmas[i], fgmas[i]);
        }
        system("pause");
}
```

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Evaluation

MAX: 10 points

MIN: 5 points

REALIZATION

COLLECTED SCORE

PRESENTATION

PRESENTED ON TIME

General requirements (1 point)

1. One program for both parts (0.5).
2. Program code is optimized (0.5).

I part (4 points)

1. Input data took from command line (1).
2. Restrictions on the data have to be programmed (1).
3. The program works correctly (2).

II part (4 points)

1. The program works correctly (2).
2. Filter: output only primary numbers (1)
3. The results are presented in a separate file (1).