Laboratory Work 2

2 ISA x86 Architecture

2.1 ISA x86 Architectural components

In the following paragraphs, the components of a processor which are visible and accessible to the assembly language are explained. Thus, a processor has a set of internal registers used for keeping temporary date, memory addresses or instructions. There are general purpose registers used in almost all arithmetic and logic operations, and special purpose registers, which have separate and different usages.

The X86 architecture contains 8 general purpose registers on 32 bits each, as follows: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. The first four are mostly used in arithmetic and logic operations (for keeping operand values), while the next four are used for keeping and calculating memory addresses for operands. The first four registers can also be addressed on half size (16 bit), but also on 8 bit. Hence, the 16 bit registers are AX, BX, CX, DX. The first 8 bits of register EAX is the inferior part "LOW" – noted AL (from bit 0 to bit 7) -, while the next 8 bits is the superior part "HIGH" – noted AH (from bit 8 to bit 15). Which means that the first half of register EAX is AX, and this half is further divided into AH and AL. This rule applies to registers EBX (BX, BH, BL), ECX (CX, CH, CL), and EDX (DX, DH, DL).

As a principle, the general purpose registers follow the orthogonality principle, meaning that any register is usable for almost all operations. However, some instructions impose the usage of specific registers, which denotes a certain specialization between registers, also noticeable in their name:

- Register EAX – **accumulator** in most arithmetic and logic instructions, usually keeps one operand and afterwards the result; for multiply and divide instructions, this register is used as default for the first operand and result.
- Register EBX – **base** register, used in arithmetic and logic instructions and also for calculating the base address when used.
- Register ECX – **counter,** used in arithmetic and logic instructions, as default in loop instructions.
- Register EDX – **data,** used in arithmetic and logic instructions, for ports in in/out instructions and also as an extension of register EAX for multiply and divide operations.
- Registers ESI and EDI – **source** and **destination index,** used for indexes for arrays, when calculating the indexed addresses.
- Register EBP – **base pointer,** used for calculating the addresses in base addressing.
- Register ESP – **stack pointer,** for addressing the stack of the running program.

Beside these registers, there are some special and control registers. In Intel processors, just a part of these registers are visible and accessible to the programmer. These registers control the working schedule of the CPU and/or allow the execution of special instructions that handle the memory space. Some of these registers are below:

- Register PC – **program counter –** keeps the address of the following instruction, is not directly addressable (by name), but its contents can be modified by executing jump instructions

- Register PSW – **program status word –** keeps status indicators/flags of the CPU:
  - Flags that store the result of an instruction
    - ZF – zero result
    - SF – the sign of the result
    - OF – overflow of quantity/size in the last arithmetic instruction
    - PF – parity of the result
    - CF – carry flag
    - AC – auxiliary carry flag (after the first 4 bits)
  - Flags that control the work procedure of the CPU:
    - IF – interrupt flag, if set to 1, then all maskable interrupts are blocked; otherwise all are validated
    - DF – direction flag, shows the travel direction on arrays (ascending or descending memory addresses)
- Segment registers – used for calculating the addresses of operands and instructions, they divide the memory in segments and there are 6 segment registers:
  - Register CS – **code segment,** used for addressing instructions, it contains the start address of the code segment (the source code of the program)
  - Register DS – **data segment,** used for addressing the operands from the memory
  - Register SS – **stack segment,** used for addressing the memory of the program stack
  - Register ES – **extra-data segment,** also used for data, like the DS
  - Registers FS and GS – similar to ES

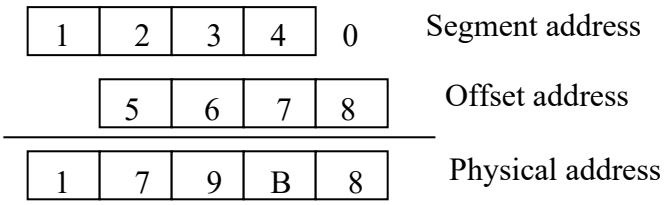
2.2 Memory addressing

The ISA x86 architecture promotes an original method for memory addressing. The memory space is divided into regions named segments, and the address of a memory location is expressed as a pair: <segment_address> : <offset_address>. The segment address indicates the beginning of a memory area/region, while the offset address is the relative address of the memory location, based on the beginning of the data segment.

In ISA x86 the used segmentation technique depends on the CPU: real-mode addressing or protected-mode addressing. The first types of CPUs implemented real-mode addressing, but was surpassed by the protected-mode addressing due to the lack of multitasking capabilities and limited RAM size.

2.2.1 Real-mode addressing

In this type of addressing, the maximum amount of RAM is 1 MB. This memory is divided into fixed-size segments of 64 KB each. The physical memory location is calculated as the sum between the segment address and the offset address. The segment address is obtained by multiplying the content of the register with 16. The offset address is calculated based on the addressing type and the address contained in the operand of the instruction. The result of the sum is the physical address on 20 bits, enough for addressing a total memory space of 1 MB. The example below shows such a memory address calculation in hexadecimal:



This type of addressing has some disadvantages:

- Maximum size for the RAM is 1 MB
- A segment must start at an address multiple of 16
- A segment can have a maximum size of 64 KB
- Segments can overlay partially or completely
- The same physical address location can be expressed through several pairs of addresses (segment : offset)
- Few possibilities for protecting the memory zones
- Any program can address any memory location, no restrictions imposed (multitasking is not possible)

2.2.2 Protected-mode addressing

This type of addressing was implemented on the ‘386 CPU and further improved on the ‘486 CPU. This addressing mode was necessary due to the limitations of the real-mode addressing.

The address calculation is similar to the previous addressing mode, but with some improvements:

- A segment register stores a segment selector
- The segment selector points to a segment descriptor
- The segment descriptor is a data structure on 32 bit that contains the segment address, segment length, access indicators, etc.
- The offset address is on 32 bit

These design modifications produce several improvements over the real-mode addressing, such as:

- Maximum addressing space is extended to 4 Gb (2^{32})
- A data segment has variable length, from 1 byte to 4 GB
- 3 levels of protection (0 is the highest priority)
- A data segment is accessible only to the assigned task and maybe to the operating system
- Some data segments can be blocked during the write operation (e.g. code segment)

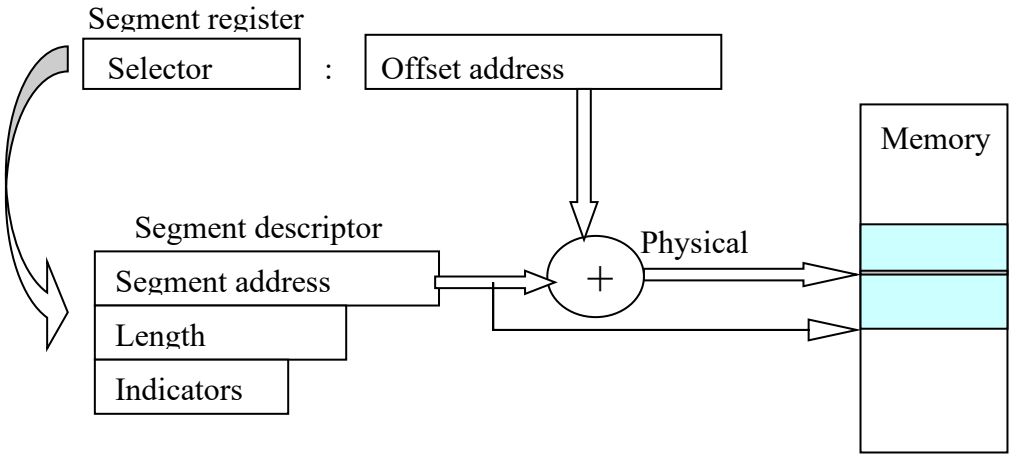


Fig. 2.1 Physical address calculation in protected-mode addressing

2.3 Exercises

1. Please follow the online document at:
   http://users.utcluj.ro/~madalin/teaching/SM/labs/W2.pdf