First part of topic 10

„The instruction set architecture (ISA) of Intel x86 processors (registers, addressing modes, instructions, memory architecture, interrupt system)”

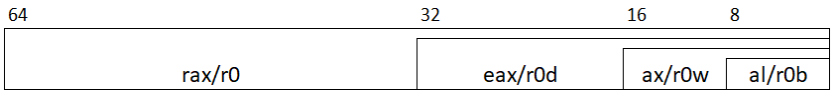
The x86 notation refers to a set of microprocessors according to the instruction set architecture (ISA). Some historically important members of this series: Intel 8086, Intel 80386, Intel Pentium, AMD Athlon, Intel Core i3/i5/i7, AMD Ryzen 3/5/7. The 32-bit versions of them are also commonly referred as IA-32 architecture. Continuous improvements since 1978 have sought backward compatibility. The components of the processor family have been implemented by several manufacturers and have been widely used in personal computers. The x86 architecture uses fixed-precision arithmetic but is interoperable with an x87-based floating-point unit (FPU) that uses real numbers.

To get to know a system based on CISC (complex instruction set computing), we need to review the main elements of the instruction set architecture, i.e. the parts of the computer architecture related to programming:

- registers
- instructions
- addressing modes
- memory management
- interrupt system

Registers

The programmer can use registers of different sizes, however, the names of the smaller registers are always aliases of parts of the larger registers. Register names refer to size as well. Registers with name ending in ‘h’ and ‘l’ are 8-bit, registers where the second character of the name is ‘x’ are 16-bit, and three-character register names beginning with ‘e’ are 32-bit. (Later, 8-byte registers beginning with the 'r' character will also appear in the 64-bit x86-64 extension, although smaller parts of it should be referenced differently.)



The programmer can use 8 general purpose registers: `eax`, `ebx`, `ecx`, `edx`, `esp`, `ebp`, `edi`, `esi`. However, these are not completely equivalent, some registers have a special meaning/property. For example, the `esp` register (stack pointer) stores the address of the top element of system stack. The `ebp` register (base pointer) is used when addressing dynamic variables with dynamic lifetime management. The `eax` register plays a special role in the division and multiplication operation, and functions returning with an integer value also return the value here. The `edx` register also plays an important role in division and multiplication, as well as in I/O operations. The `ecx` register can be used as a loop counter. The `edi` and `esi` registers can be used as an index during string operations and also play an

important role in parameter passing. For x86-64, we have an additional 8 64-bit general purpose registers (`r8-r15`).

There are other special, system registers. The `eip` (`rip`) register (instruction pointer, program counter) contains the memory address of the next machine instruction, so it is essential for sequential execution and control transfer. The `eflags` (`rflags`) register acts as a status register. Its bits indicate the presence or absence of states. These include carry bit (CF), overflow bit (OF), sign bit (SF), zero bit (ZF), interrupt mask bit (IF), and so on. Segment selector registers (CS, DS, SS, ES) required to use different memory segments can also be used.

Floating-point x87 registers and registers provided by additional extensions (MMX, SSE, AVX, AVX-512) can also be used.

Instructions

Many complex instructions are available since it is a CISC system. Instructions can access memory, that is, in most cases the operand can be a constant, a register content, and a value stored in memory. Mostly two-address instructions can be used and the length of the statements varies. The instructions also specify the size of the operand. The most important instruction categories are:

- Data transfer instructions (`mov, push, pop, lea, movsx, xchg, cdq`, …)
- Arithmetic and logic instructions (`add, sub, imul, idiv, inc, dec, neg, cmp, and, or, xor, not, shl, shr, sar, ror, rcr`, …)
- Control flow instructions (`jmp, je, jne, jg, jge, jl, jle, jb, ja, jz, jnz, jc, jo, call, ret, leave, int, loop`, …)
- Other instructions (`nop, cpuid, sti, cli`, …)

Addressing modes

We can define “access modes” of operands of instructions using different addressing modes. It is not permitted to specify a memory reference for both the target and source operands within a statement. The main addressing methods are the follows:

- Immediate: The operand is a constant that is part of the machine code instruction in the code segment.
- Register Addressing: The operand is in the specified register.
- Direct addressing: The operand is a memory address from which the required value can be read/written.
- Register indirect: A specified register contains the memory address of the value to be used.
- Register relative: Use a memory area that is a certain offset from the memory address in a base register.
- Indexed Addressing: To a specific memory address, a value in a register used as an index must be added to find the operand. The value of the index can also be multiplied by a constant value before being added to the starting address (scaling).

The following terms summarize the possibilities of using these addressing methods in combination:

```
[<segment>:] [<base>] [+<index> [*<scale>]] [+<offset>]
```

Here, `<segment>` means segment selector registers, `<base>` and `<index>` can be general-purpose registers, and `<scale>` and `<offset>` can be constant, the former can take values of 1,2,4 or 8. (Square brackets represent an option.)

Memory management

Due to the technical limitations of early systems, it was necessary to use segmented memory. It later survived for a long time for compatibility reasons. The main point is to be able to generate 20-bit physical addresses using the two 16-bit registers used by the 8086 processor. The value of a segment selector register is shifted by 4 bits (multiplication by 16) and then must be added to the value of a general register to obtain the address of the required operand. So, a segment is 64kB in size and the total address space was 1MB.

It was used by the original mode of operation, which was named *real mode* after the introduction of a new *protected mode* that implemented address space extension and certain protection features. In 32-bit systems we can also use so-called *virtual mode*, and *long mode* on 64-bit systems.

The register set supports the use of different segments. The machine code of the program is located in the code segment. The segment register is the `CS` (which, for example, “works with” the `EIP`). The data segment contains the variables with static lifetime management, the `DS` register is required to achieve this. The system stack is located in the stack segment, by default the `ESP` and `EBP` registers work in conjunction with the `SS` segment register.

Interrupt system

We can use 3 types of interrupts:

- Hardware (external devices can indicate their need for interruption, IRQ)
- Software (the program requests an interrupt with the `int` instruction)
- Exceptions (processor-generated interrupt, e.g. division-by-zero, page fault)

Interrupt requests that also have priority are processed by a PIC (programmable interrupt controller, such as Intel 8259A) and forwarded to the processor. In the processor, interrupt handling can be masked by the `IF` bit of the `EFLAGS` register. Using a vector interrupt, the address of the corresponding ISR (interrupt handling routine) is found based on the IDT (interrupt description table) entry associated with the interrupt requester. Before interrupt handling, of course, a context save is required (i.e., the contents of the `EIP`, `CS`, `EFLAGS` registers, among others, will be saved to the stack).

Comment:

The ability to use the programming tools associated with the topic may be required. Wider familiarity with the subject is not a disadvantage. This is just a short summary.

Related subjects:

- *Computer architectures*
- *Assembly programming*

Further suggested readings:

- Joseph Cavanagh: X86 Assembly Languages and C fundamentals (CRC Press, 2013) chapter 2 and 3.
- Richard Blum: *Professional Assembly Language* (Wiley, 2005) chapter 2.