5 Addressing types

5.1 Addressing modes in the x86 architecture

The addressing modes are related to a mechanism used to determine the operands of an instruction. An operand can exist in several ways: constant, register, memory location or even interface port. An addressing mode indicates the search mechanism of the operand and eventually the calculation method of the address, if it's stored in the memory. The addressing mechanism influences the execution speed of instructions and determines the length of it. Hence, complex addressing mechanism require a longer execution time, but offer a greater flexibility in finding and retrieving data.

5.1.1 Immediate addressing

This is the simplest type of addressing. The operand is a constant and when the instruction is read, the operand is read as well. The constant can be in decimal form (default), hexadecimal form (the letter "h" appears at the end), binary form ("b" at the end) or ASCII codes (character between ‘’). The constant is always the second operand in an instruction.

Examples:

```
mov    ax, 1234h
add    cl, 30
and    bh, 01111b
```

This addressing mode is relatively fast because it doesn't need additional transfer for retrieving the operand value, but its flexibility is limited because an instruction uses a single constant.

5.1.2 Direct addressing

This type of addressing assumes the presence of an address for the operand in the instruction. The operand is a memory location specified in a variable declaration. The address of the operand can be expressed as a value or as a symbolic name given to the variable. In order to avoid confusion, any memory address expressed as a value is placed between brackets. Note that this addressing mode allows the access of a single memory location.

Examples:

```
mov    ax, [100h]      ; 100h if the offset address of the operand
add    [200h], ch
cmp    dx, var16       ; var16 must be declared in the data section as a 16 bit variable
```

5.1.3 Register addressing

In this addressing mode, the operand is in a CPU register. This mode is actually a more efficient form of direct addressing:

- The registers are inside of the CPU, which eliminates the additional memory transfer.
- The address of a register is expressed on 3-4 bits, which reduces the dimensions of instructions and hence, faster execution.
- The transfers between registers are performed at higher speeds due to the usage of internal buses of the CPU.

Examples:

```
mov    ax, bx
cmp    ah,bl
```

However, the major disadvantage is the limited number of internal registers and not all variables of a program can be kept in registers.

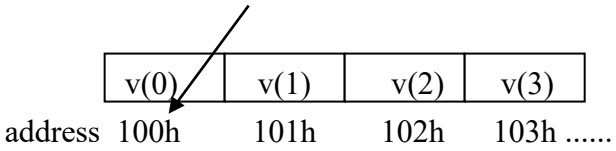
5.1.4 Indexed addressing

The following types of addressing are indirect types of addressing because the instruction does not contain the operand or the address, but an indication of the place where the address is. Usually the address is stored inside a register or is calculated as a sum of registers and constants.

The indexed addressing mode allows the retrieval of an array's elements, by simply incrementing or decrementing an index register (ESI or EDI). The address of an element from an array is specified through an index register or as a sum between the content of an index register and a constant.

Examples:

```
mov    ax, [esi]
and    var[esi], 30h          ; var is interpreted as a constant which is added to ESI
mov    [edi], cl
add    ah, [esi+100h]         ; ESI contains the index of the searched element
```



The next element of the array is found by incrementing or decrementing the index register (1 for byte, 2 for word and 4 for double).

This addressing mode is slower than the previous because it implies a mathematical calculation and an additional memory addressing. On the other hand, it is a flexible method for addressing array and vectors, especially when loops are used.

5.1.5 Based addressing

Based addressing is similar to indexed addressing (same method for calculating the address), but it has a different purpose. This addressing mode is used for structures and records. Usually, a structure contains data of different types and sizes, which makes the indexed addressing unfit. Hence, an element from the structure is retrieved by specifying the start address of the structure and the distance to the element from the start. The address of the structure is specified in registers EBX or EBP, while the relative distance through a constant.

Examples:

```
mov    ax, [ebp]
add    [ebx+100h], 50
sub    dx, var[ebx]
```

As displayed above, the indexed and based addressing are identical. Only the registers differ. Furthermore, an index register can be used as a base register and vice-versa. Only the address interpretation matters. Hence, the based addressing mode has the same advantages and disadvantages as the indexed addressing.

5.1.6 Indexed-based addressing

This addressing method is a combination of the indexed and based addressing modes. It can be used in complex data structures such as: array of records, bi-dimensional arrays, etc. The address of the operand is calculated as sum between an index register, a base register and a constant. This is the most flexible addressing mode allowed by the ISA x86, but, at the same time, the least efficient. The addressing of the operand implies 2 additions and an additional memory transfer

Examples:

```
mov    ax, [esi+ebx]
sub    var[esi+ebp],ch
add    ax, [edi+ebx+100h]
```

5.1.7 Array addressing

Array addressing is a special form of indexed addressing. In this addressing mode, the index registers are not specified, they are defined by default: ESI for the source array and EDI for the destination array. Furthermore, the CX register is used as a counter. At each execution of the instruction, the index registers are automatically increased or decreased to move the next elements in the array. By using instructions starting with REP or REPZ, a single instruction is used for transferring or processing a single data block.

Example:

```
mov    esi, offset sir1
mov    edi, offset sir2
mov    ecx, lung_sir
rep    movsb
```

5.1.8 Stack addressing

This addressing mode used by default the ESP register for addressing an operand. This addressing method is used only in 2 instructions, the ones that work with the top of the stack: PUSH and POP. The ESP register modified after each instruction such that it always addresses the top of the stack. In the ISA x86 architecture, the program stack increases to the smaller addresses (ESP decrements when adding/saving data to/in the stack) and decreases to higher addresses (ESP increments when retrieving/extracting an element from the stack).

Examples:

```
push    ax
push    cs
pop     ds
push    var16
```

5.2 Exercises

1. Please follow the online document at:
   http://users.utcluj.ro/~madalin/SM/labs/W5.pdf