**How linear address is calculated ?**

**The effective address(offset) is added with segment base address to calculate linear address.**

**How physical address is calculated ?**

**To form a physical memory address, appropriate segment register contents are shifted by left by 4 positions and then added to 16-bit offset address formed using one of addressing modes, in same way as in the 80386 real address mode.**

**What are The instructions available in the 80386 that are not available in its real address mode ?**

**All the instructions of 80386 are available in this mode except for those designed to work with or for protected address mode**.

**80386 ADDRESSING MODE**

On the 80386 you may specify any general purpose 32 bit register when using the register indirect addressing mode. [eax], [ebx], [ecx], [edx], [esi], and [edi] all provide offsets, by default, into the data segment. The [ebp] and [esp]addressing modes use the stack segment by default.  
**SOME BASIC DEFINATIONS**  
  
Effective address:te offset calculated for memory operand is called operand’s EA it can be computed by adding any combinations of following four components. Displacement:8 or 32 bit immediate data following the instruction.16 bit may also be used.  
Base:the content of any general purpose register can be used.

**REGISTER ADDRESSING MODE**  
  
In this addressing mode ,Data is in the registers and the

instruction specifies the particular register.  
In this mode execution of instruction is faster.  
Example:MOV EAX,EDX  
It will copy the content of EDX register to EAX register

**IMMEDIATE ADDRESSING MODE**  
  
Immediate data is the constant data contained in an instruction.  
It maybe 8,16 or 32bits in length  
Source operand is the part of instruction  
Example:  
MOV ECX,20304050H  
EA=20304050H  
  
**REGISTER INDIRECT ADDRESSING MODE**In this mode ,a base or index register contains the operand’s effective address.  
EA={base register}or{index register}  
Example:  
MOV EBX,[ECX]  
EA=ECX  
PA=DS:ECX  
  
**BASED SCALED INDEX**  
The contents of an index register are multiplied by a scaling factor and the result is added to base register to compute EA.  
EA={base register}+{index register\*scaling factor}  
Example:  
MOV ECX,[EDI\*4][ESP]  
EA=[EDI\*4]+[ESP]  
PA=DS: [EDI\*4]+[ESP]  
  
Reference: <http://seminarprojects.com/Thread-addressing-modes-of-80386#ixzz3RbhO3ens>

**80386 Register Addressing Modes**

The 80386 (and later) processors provide 32 bit registers. The eight general-purpose registers all have 32 bit equivalents. They are eax, ebx, ecx, edx, esi, edi, ebp, and esp. If you are using an 80386 or later processor you can use these registers as operands to several 80386 instructions.

**4.6.4 80386 Memory Addressing Modes**

The 80386 processor generalized the memory addressing modes. Whereas the 8086 only allowed you to use bx or bp as base registers and si or di as index registers, the 80386 lets you use almost any general purpose 32 bit register as a base or index register. Furthermore, the 80386 introduced new scaled indexed addressing modes that simplify accessing elements of arrays. Beyond the increase to 32 bits, the new addressing modes on the 80386 are probably the biggest improvement to the chip over earlier processors.

**4.6.4.1 Register Indirect Addressing Modes**

On the 80386 you may specify any general purpose 32 bit register when using the register indirect addressing mode. [eax], [ebx], [ecx], [edx], [esi], and [edi] all provide offsets, by default, into the data segment. The [ebp] and [esp]addressing modes use the stack segment by default.  
  
Note that while running in 16 bit real mode on the 80386, offsets in these 32 bit registers must still be in the range 0...0FFFFh. You cannot use values larger than this to access more than 64K in a segment. Also note that you must use the 32 bit names of the registers. You cannot use the 16 bit names. The following instructions demonstrate all the legal forms:

mov al, [eax]

mov al, [ebx]

mov al, [ecx]

mov al, [edx]

mov al, [esi]

mov al, [edi]

mov al, [ebp] ;Uses SS by default.

mov al, [esp] ;Uses SS by default.

**4.6.4.2 80386 Indexed, Base/Indexed, and Base/Indexed/Disp Addressing Modes**

The indexed addressing modes (register indirect plus a displacement) allow you to mix a 32 bit register with a constant. The base/indexed addressing modes let you pair up two 32 bit registers. Finally, the base/indexed/displacement addressing modes let you combine a constant and two registers to form the effective address. Keep in mind that the offset produced by the effective address computation must still be 16 bits long when operating in real mode.  
  
On the 80386 the terms base register and index register actually take on some meaning. When combining two 32 bit registers in an addressing mode, the first register is the base register and the second register is the index register. This is true regardless of the register names. Note that the 80386 allows you to use the same register as both a base and index register, which is actually useful on occasion. The following instructions provide representative samples of the various base and indexed address modes along with syntactical variations:

mov al, disp[eax] ;Indexed addressing

mov al, [ebx+disp] ; modes.

mov al, [ecx][disp]

mov al, disp[edx]

mov al, disp[esi]

mov al, disp[edi]

mov al, disp[ebp] ;Uses SS by default.

mov al, disp[esp] ;Uses SS by default.

The following instructions all use the base+indexed addressing mode. The first register in the second operand is the base register, the second is the index register. If the base register is esp or ebp the effective address is relative to the stack segment. Otherwise the effective address is relative to the data segment. Note that the choice of index register does not affect the choice of the default segment.

mov al, [eax][ebx] ;Base+indexed addressing

mov al, [ebx+ebx] ; modes.

mov al, [ecx][edx]

mov al, [edx][ebp] ;Uses DS by default.

mov al, [esi][edi]

mov al, [edi][esi]

mov al, [ebp+ebx] ;Uses SS by default.

mov al, [esp][ecx] ;Uses SS by default.

Naturally, you can add a displacement to the above addressing modes to produce the base+indexed+displacement addressing mode. The following instructions provide a representative sample of the possible addressing modes:

mov al, disp[eax][ebx] ;Base+indexed addressing

mov al, disp[ebx+ebx] ; modes.

mov al, [ecx+edx+disp]

mov al, disp[edx+ebp] ;Uses DS by default.

mov al, [esi][edi][disp]

mov al, [edi][disp][esi]

mov al, disp[ebp+ebx] ;Uses SS by default.

mov al, [esp+ecx][disp] ;Uses SS by default.

There is one restriction the 80386 places on the index register. You cannot use the esp register as an index register. It's okay to use esp as the base register, but not as the index register.

**4.6.4.3 80386 Scaled Indexed Addressing Modes**

The indexed, base/indexed, and base/indexed/disp addressing modes described above are really special instances of the 80386 scaled indexed addressing modes. These addressing modes are particularly useful for accessing elements of arrays, though they are not limited to such purposes. These modes let you multiply the index register in the addressing mode by one, two, four, or eight. The general syntax for these addressing modes is

disp[index\*n]

[base][index\*n]

or

disp[base][index\*n]

where "base" and "index" represent any 80386 32 bit general purpose registers and "n" is the value one, two, four, or eight.  
  
The 80386 computes the effective address by adding disp, base, and index\*n together. For example, if ebx contains 1000h and esi contains 4, then

mov al,8[ebx][esi\*4] ;Loads AL from location 1018h

mov al,1000h[ebx][ebx\*2] ;Loads AL from location 4000h

mov al,1000h[esi\*8] ;Loads AL from location 1020h

Note that the 80386 extended indexed, base/indexed, and base/indexed/displacement addressing modes really are special cases of this scaled indexed addressing mode with "n" equal to one. That is, the following pairs of instructions are absolutely identical to the 80386:

mov al, 2[ebx][esi\*1] mov al, 2[ebx][esi]

mov al, [ebx][esi\*1] mov al, [ebx][esi]

mov al, 2[esi\*1] mov al, 2[esi]

Of course, MASM allows lots of different variations on these addressing modes. The following provide a small sampling of the possibilities:

disp[bx][si\*2], [bx+disp][si\*2], [bx+si\*2+disp], [si\*2+bx][disp], disp[si\*2][bx], [si\*2+disp][bx], [disp+bx][si\*2]