Addressing Modes

Data Addressing Modes

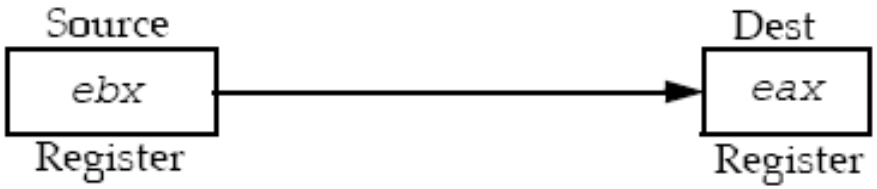
- Let's cover the data addressing modes using the ***mov*** instruction.
  - Data movement instructions move data (bytes, words and doublewords) between registers and between register / memory.
  - Only the ***movs*** (strings) instruction can have both operands in memory.
  - Most data transfer instructions do not change the **EFLAGS** register.

4-Byte Data Width

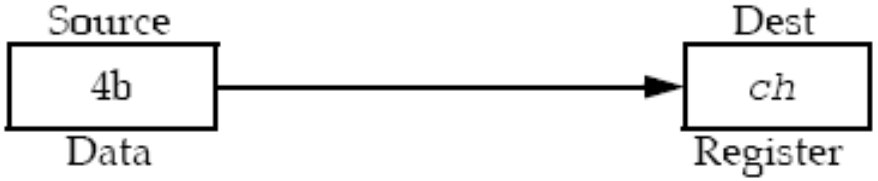
- Storage protocols
  - When an **n**-byte transfer is indicated by an address **a**, the memory bytes referred to are those at the address **a**, **a**+1, …, **a**+**n**-1
  - When an **n**-byte number is stored in memory, its bytes are stored in order of significance → little endian

Data Addressing Modes

- Register
  - mov eax, ebx

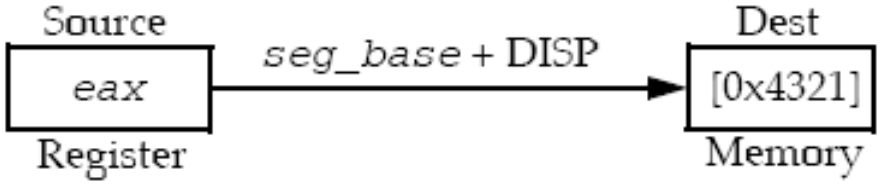


- Immediate
  - mov ch, 0x4b



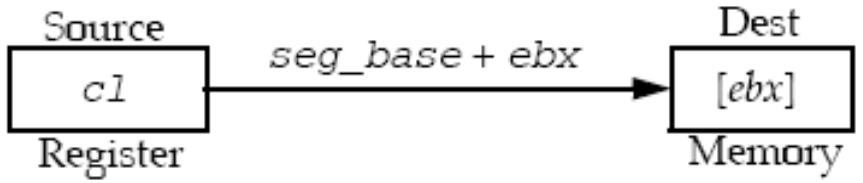
- Direct (eax), Displacement (other regs)
  - mov [0x4321], eax

Indirection operator!!

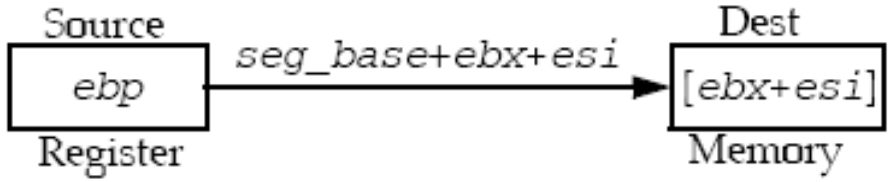


Data Addressing Modes

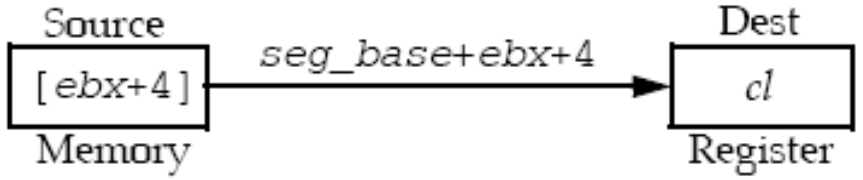
- Register Indirect
  - mov [ebx], cl
  - Any of eax, ebx, ecx, edx, ebp, edi or esi may be used.



- Base-plus-index
  - mov [ebx+esi], ebp
  - Any combination of eax, ebx, ecx, edx, ebp, edi or esi.

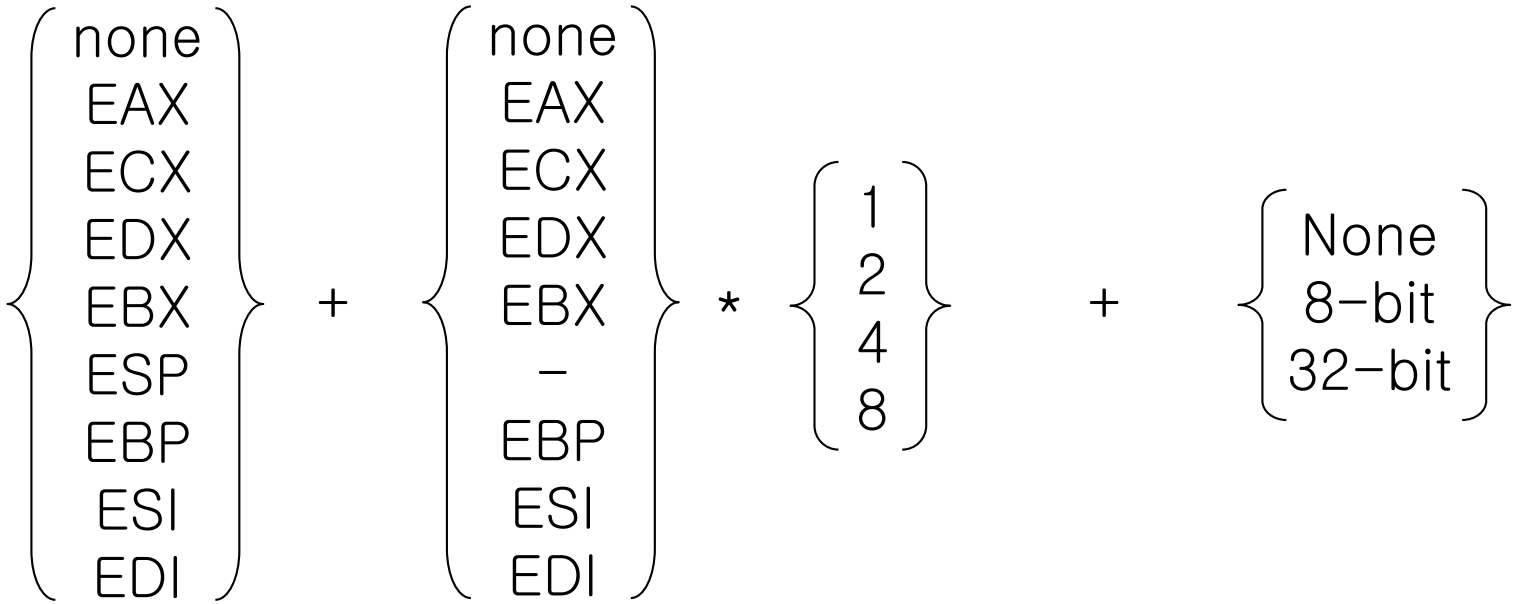


- Register relative
  - mov cl, [ebx+4]
  - A second variation includes: mov eax, [ebx+ARR]



X86 Indirect Addressing Modes

BASE + (INDEX * SCALE) + DISPLACEMENT



Register Addressing

- *mov* really COPIES data from the source to destination register.
- Never mix an 16-bit register with a 32-bit, etc.
- For example

```
mov  eax,  bx      ;ERROR: NOT permitted.
```

- None of the *mov* instruction effect the EFLAGS register.

Immediate Addressing

- The value of the operand is given as a constant in the instruction stream.

```
mov eax, 0x12345
```

- Use ***b*** for binary, ***o*** for octal, ***h (or 0x)*** for hexadecimal and nothing for decimal
- ASCII data requires a set of apostrophes:

```
mov eax, 'A'    ;Moves ASCII value 0x41 into eax.
```

- Immediate addressing example

```
mov eax, 0      ;Immediate addressing.
mov ebx, 0x0000
mov ecx, 0
mov esi, eax    ;Register addressing.
...
```

Displacement Addressing

- Displacement addressing
  - Displacement instructions are encoded with up to 7 bytes (32 bit register and a 32 bit displacement).
  - To access a statically allocated scalar operand

```
mov  cl,[DATA1]        ;Copies a byte from DATA1.
mov  edi,[SUM]         ;Copies a doubleword from SUM.
```

- Direct addressing
  - Transfers between memory and *al*, *ax* and *eax*.

```
mov  al,[DATA1]        ;Copies a byte from DATA1.
mov  al, [0x4321]      ;Some assemblers don't allow this.
mov  al, ds:[0x1234]
mov [DATA2], ax        ;Copies a word to DATA2.
```

Register Indirect Addressing

- Offset stored in a register is added to the segment register. Used for dynamic storage of variables and data structures

```
mov ecx, [ebx]
```

- The memory to memory ***mov*** is allowed with string instructions.
  - Any register EXCEPT **esp** for the 80386 and up.
  - For **eax**, **ebx**, **ecx**, **edx**, **edi** and **esi**: The data segment is the default.
  - For **ebp**: The stack segment is the default.
  - Some versions of register indirect require special assembler directives ***byte, word,*** or ***dword***

```
mov al, [edi]       ;Clearly a byte-sized move.
mov [edi], 0x10     ;Ambiguous, assembler can't size.
```

  - Does [*edi*] address a byte, a word or a double-word? Use

```
mov byte [edi], 0x10     ;A byte transfer.
```

Register Indirect Addressing

```
;
; Code which adds two 256-byte numbers y and x:
; y = y + x
;
; Assume the 256 bytes of y are stored starting at memory address 100H.
; Assume the 256 bytes of x are stored starting at memory address 200H.
; Use EDX to store a decrement counter for the y = y + x loop.
;
        MOV EDI, 100H ; Initialize pointer into y.
        MOV ESI, 200H ; Initialize pointer into x.
;
; y = y + x
        MOV EDX, 40H    ; Loop needs 64 iterations.
        CLC             ; Clear the carry flag.
XYZ:    MOV EAX,[ESI]   ; 4 Source bytes into the Processor.
        ADC [EDI],EAX   ; Do the addition.
        INC ESI         ;
        INC ESI         ;
        INC ESI         ; This is ugly.
        INC ESI         ;
        INC EDI         ; But using ADD here would
        INC EDI         ; clear the carry flag.
        INC EDI         ;
        INC EDI         ;
        DEC EDX         ; Decrement the loop counter.
        JNZ XYZ         ; See if the loop is finished.
```

Register Relative Addressing

- Effective address computed as: seg_base + base + constant.
- Same default segment rules apply with respect to **ebp**, **ebx**, **edi** and **esi**.
- Displacement constant is any *32-bit* signed value.

```
mov eax, [ebx+1000H]   ;Data segment copy.
mov [ARRAY+esi], BL    ;Constant is ARRAY.
mov edx, [LIST+esi+2] ;Both LIST and 2 are constants.
mov edx, [LIST+esi-2] ;Subtraction.
```

Register Relative Addressing

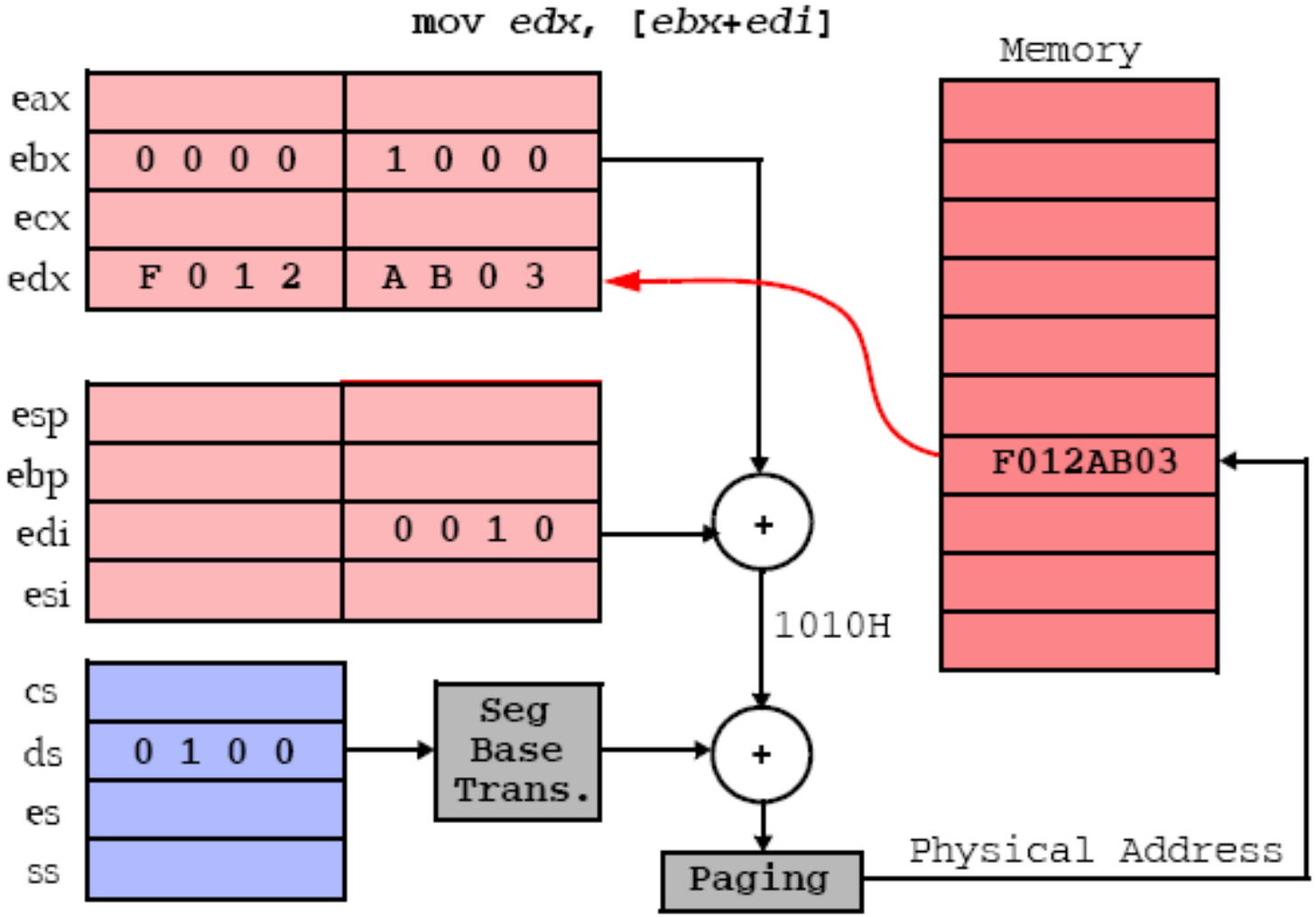
- Base+displacement
  - An index into an array when the element size is not 2, 4, or 8 bytes; The displacement encodes the static offset to the beginning of the array, while the base register holds the results of a calculation to determine the offset to a specific element within the array
  - To access a field of a record; the base register holds the address of the beginning of the record, while the displacement is an static offset to the field
  - A important special case is access to parameters in a procedure activation record (the base register in this case is EBP)
- (Index*scale)+displacement
  - Index into a static array when the element size is 2, 4, or 8 bytes

Base-Plus-Index Addressing

- Effective address computed as: seg_base + base + index.
- ***Base registers:*** Holds starting location of an array.
  - **ebp, esp** (stack)
  - **ebx, …** (data)
- ***Index registers:*** Holds offset location.
  - **edi**
  - **esi**
  - Any 32-bit register except **esp**.
- Dynamic array ??

```
mov ecx,[ebx+edi]    ;Data segment copy.
mov ch, [ebp+esi]    ;Stack segment copy.
mov dl, [eax+ebx]    ;EAX as base, EBX as index.
```

Base-Plus-Index Addressing

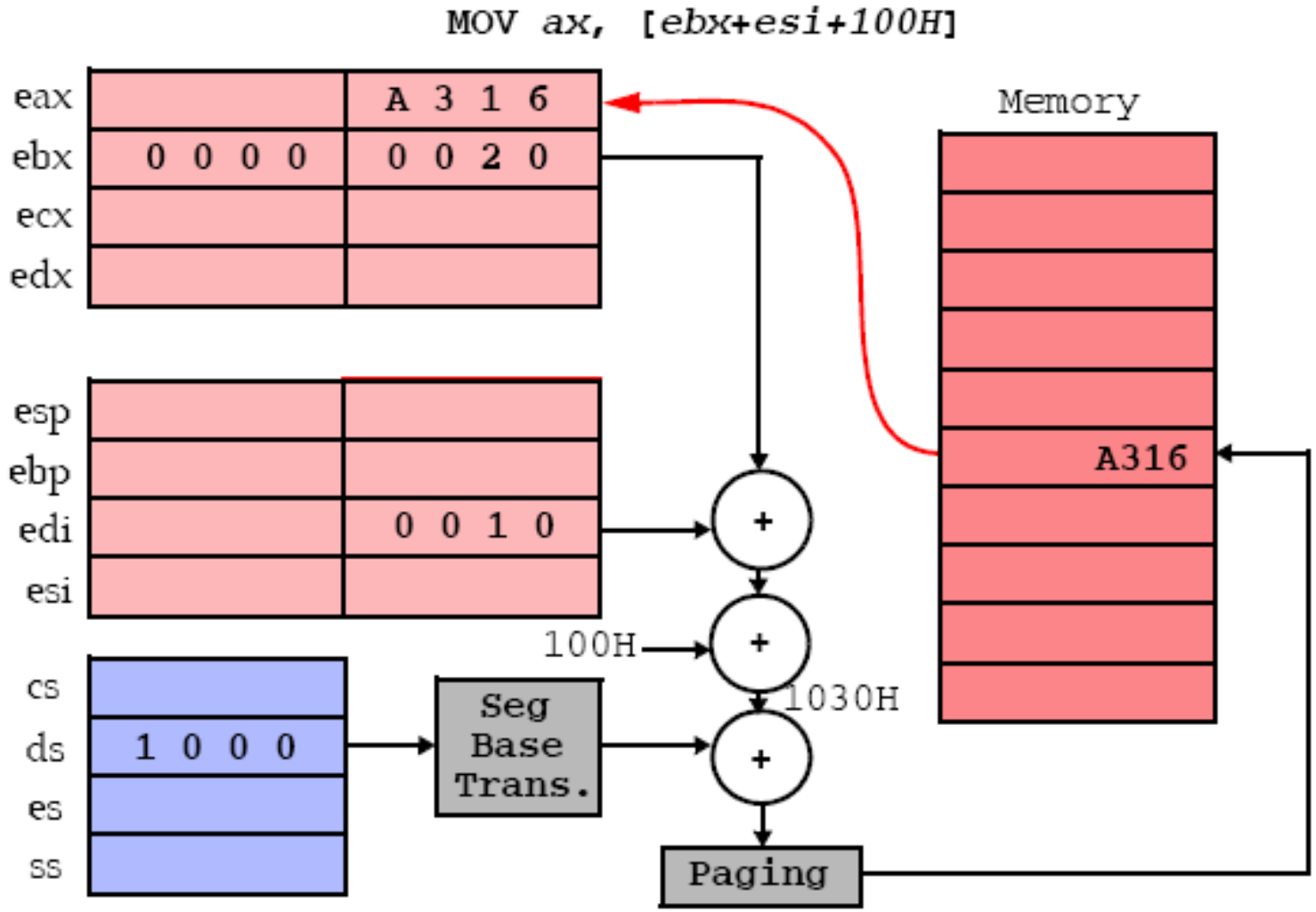


Base Relative-Plus-Index Addressing

- Effective address computed as: seg_base + base + index + constant.
- Designed to be used as a mechanism to address a two-dimensional array (the displacement holds the address of the beginning of the array)
- One of several instances of an array of records (displacement is an offset to a field within the record)

```
mov  dh, [ebx+edi+20H]       ;Data segment copy.
mov  ax, [FILE+ebx+edi]      ;Constant is FILE.
mov  [LIST+ebp+esi+4], dh    ;Stack segment copy.
mov  eax, [FILE+ebx+ecx+2]   ;32-bit transfer.
```

Base Relative-Plus-Index Addressing



Arrays

```
num_zeros = 0;
num_ones = 0;
for(i = 20; i < 30; i = i + 1)
for(j = 50; j < 55; j = j + 1)
{
    if (x[i][j] == 0)
        num_zeros = num_zeros + 1;
    if (x[i][j] == 1)
        num_ones = num_ones + 1;
}
```

```
        MOV EBX, 0          ; num_zeros
        MOV ECX, 0          ; num_ones
        MOV EDX, 8000       ; 400 * 20, Initially i = 20
;
; Outer loop begins here.
;
OTL:    MOV ESI, 50         ; Let j = 50
;
; Inner loop begins here.
;
INL:    MOV EAX, [ABC + 4*ESI + EDX]
                            ; ABC = &x[0][0].
                            ; EDX = &x[i][0] - &x[0][0]
                            ; 4 * j = &x[i][j] - &x[i][0]
                            ; ESI = j
        CMP EAX, 0          ; Check for zeros
        JNE NOZ             ;
        INC EBX             ; Count zeros
NOZ:    CMP EAX, 1          ; Check for ones
        JNE NOO
        INC ECX             ; Count ones
NOO:    INC ESI             ; j = j + 1
        CMP ESI, 55         ; Check that j < 55
        JL INL              ; Inner loop ends here.
        ADD EDX, 400        ; Increase EDX by 100 * 4
        CMP EDX, 12000      ; 8000 + 10 * 100 * 4
        JL OTL              ; Outer loop ends here.
ABC:    NOP                 ; Begin array here.
```

Scaled-Index Addressing

- Effective address computed as: seg_base + base + constant*index
- Indexing two-dimensional array when the elements of the array are 2, 4, or 8 bytes in size

```
mov  eax, [ebx+4*ecx]        ;Data segment DWORD copy.
mov  [eax+2*edi-100H], cx    ;Whow !
mov  eax, [ARRAY+4*ecx]      ;Std array addressing.
```

IA-32 SW Developer's man	Lecture note	Application
displacement	Direct Displacement	- To access a statically allocated scalar operand
base	Register indirect	- Used for dynamic storage of variables and data structures
Base+displacement	Register relative	- An index into an array when the element size is not 2, 4, or 8 bytes (the displacement encodes the static offset to the beginning of the array; The base register holds the results of a calculation to determine the offset to a specific element within the array) - To access a field of a record (the base register holds the address of the beginning of the record, while the displacement is an static offset to the field) - A special case is access to parameters in a procedure activation record (the base register in this case is EBP)
(Index*scale)+displacement		- Index into a static array when the element size is 2, 4, or 8 bytes
Base+Index+Displacement	Base relative-plus-index	- A two-dimensional array (the displacement holds the address of the beginning of the array) - One of several instances of an array of records (displacement is an offset to a field within the record
	Base-plus-index	- Dynamic array ??
Base+(Index*scale)+Displacement	Scaled index	Indexing 2-dimensional array when the elements of the array are 2, 4, or 8 bytes in size