Data Addressing Modes

Base-Plus-Index addressing:

Effective address computed as:

seg_base + base + index.

Base registers: Holds starting location of an array.

- **ebp** (stack)
- **ebx** (data)
- Any 32-bit register except **esp**.

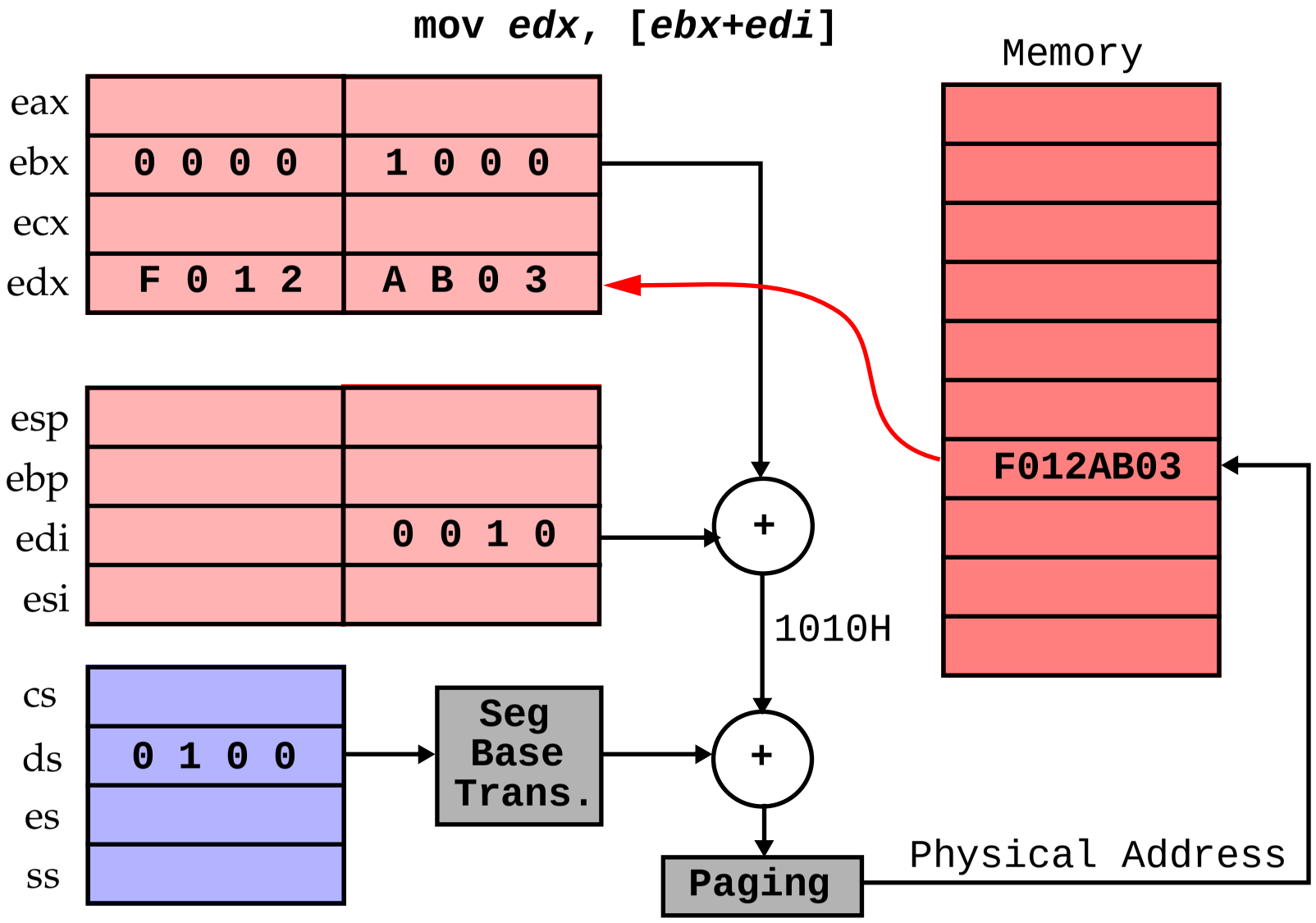
Index registers: Holds offset location.

- **edi**
- **esi**
- Any 32-bit register except **esp**.

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

Data Addressing Modes

Base-Plus-Index addressing:



Data Addressing Modes

Register Relative addressing:

Effective address computed as:

seg_base + base + constant.

```
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.
```

Same default segment rules apply with respect to **ebp**, **ebx**, **edi** and **esi**.

Displacement constant is any *32-bit* signed value.

Base Relative-Plus-Index addressing:

Effective address computed as:

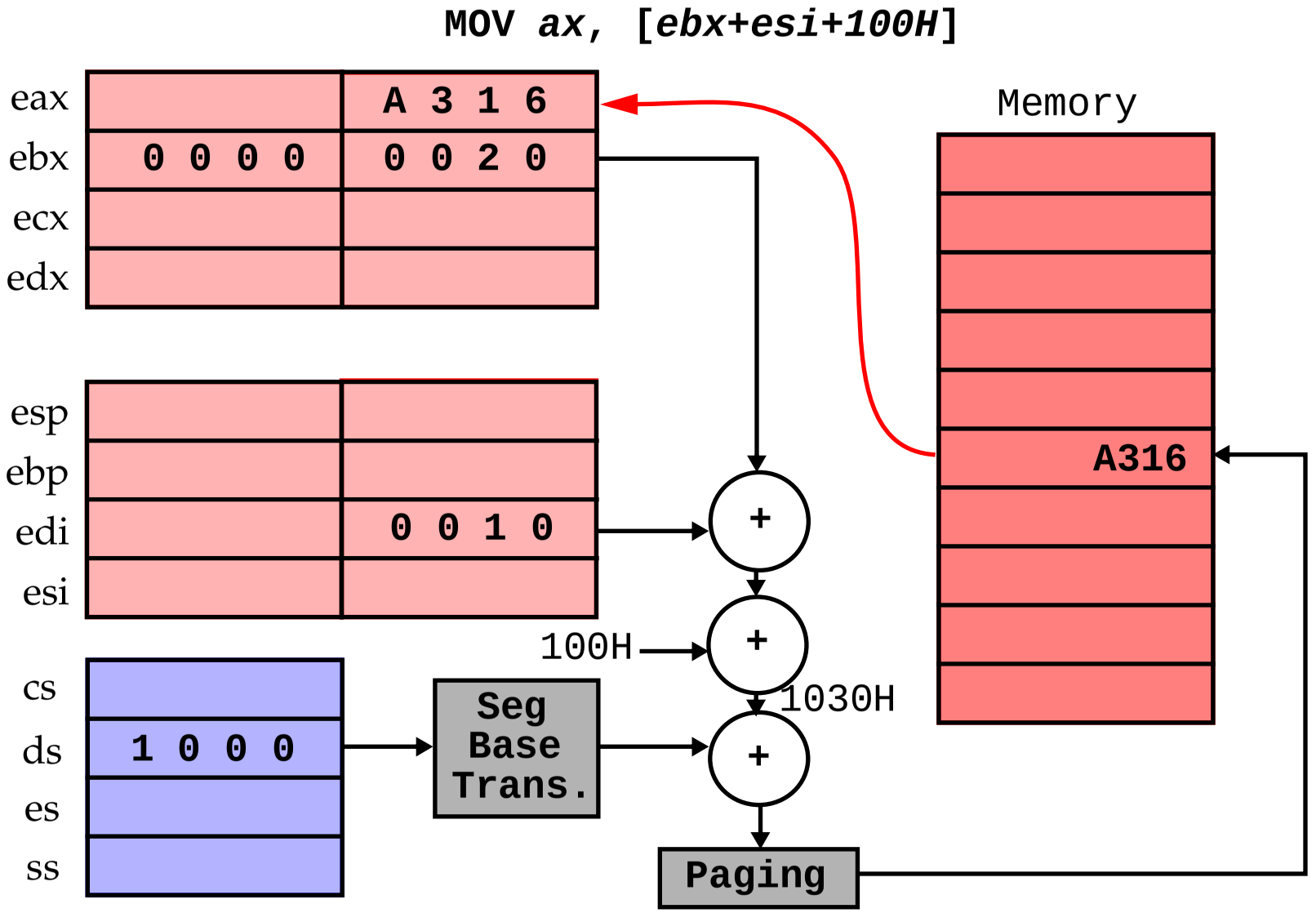
seg_base + base + index + constant.

```
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.
```

Designed to be used as a mechanism to address a two-dimensional array.

Data Addressing Modes

Base Relative-Plus-Index addressing:



Data/Code Addressing Modes

Scaled-Index addressing:

Effective address computed as:

seg_base + base + constant*index.

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

Code Memory-Addressing Modes:

Used in **jmp** and **call** instructions.

Three forms:

- Direct
- PC-Relative
- Indirect

Direct:

Absolute jump address is stored in the instruction following the opcode.

Code Addressing Modes

An *inter*segment jump:

Opcode	Offset (low)	Offset (high)	Segment(low)	Segment(high)
E A	0000	0000	00	10

This **far jmp** instruction loads **cs** with 1000H and **eip** with 00000000H.

A **far call** instruction is similar.

PC-Relative:

A displacement is added to the **EIP** register.

This constant is encoded into the instruction itself, as above.

*Intra*segment jumps:

- Short jumps use a 1-byte signed displacement.
- Near jumps use a 4-byte signed displacement.

The assembler usually computes the displacement and selects the appropriate form.

Code Addressing Modes

Indirect:

Jump location is specified by a register.

There are three forms:

- Register:

  Any register can be used: **eax**, **ebx**, **ecx**, **edx**, **esp**, **ebp**, **edi** or **esi**.

```
jmp eax                 ;Jump within the code seg.
```

- Register Indirect:

  *Intra*segment jumps can also be stored in the data segment.

```
jmp [ebx]               ;Jump address in data seg.
```

- Register Relative:

```
jmp [TABLE+ebx]         ;Jump table.
jmp [edi+2]
```

Stack Addressing Modes

The stack is used to hold temporary variables and stores return addresses for procedures.

push and **pop** instructions are used to manipulate it.

call and **ret** also refer to the stack implicitly.

Two registers maintain the stack, **esp** and **ss**.

A **LIFO** (Last-in, First-out) policy is used.

The stack grows toward lower address.

Data may be pushed from any of the registers or segment registers.

Data may be popped into any register except **cs.**

```
popfd               ;Pop doubleword for stack to EFLAG.
pushfd              ;Pushes EFLAG register.
push 1234H          ;Pushes 1234H.
push dword [ebx]    ;Pushes double word in data seg.
pushad              ;eax,ecx,edx,ebx,esp,ebp,esi,edi
pop eax             ;Pops 4 bytes.
```