# Indirect addressing

CSC 236

## What

- Direct addressing
    - Access memory at given location
    - Given a particular address (offset)
- Indirect addressing
    - Access memory at location *relative* to a given location
    - Resultant address is computed
        - Addition (most common)
        - Multiplication (not 8086)
    - The given location is referred to as *index* or *pointer*

# Why

- Suppose need to add list of vars

```
      .data
a    dw   ?
b    dw   ?
c    dw   ?
…
z    dw   ?
```

- Poor solution

```
mov   ax,0
add   ax,[a]
add   ax,[b]
add   ax,[c]
...
add   ax,[z]
```

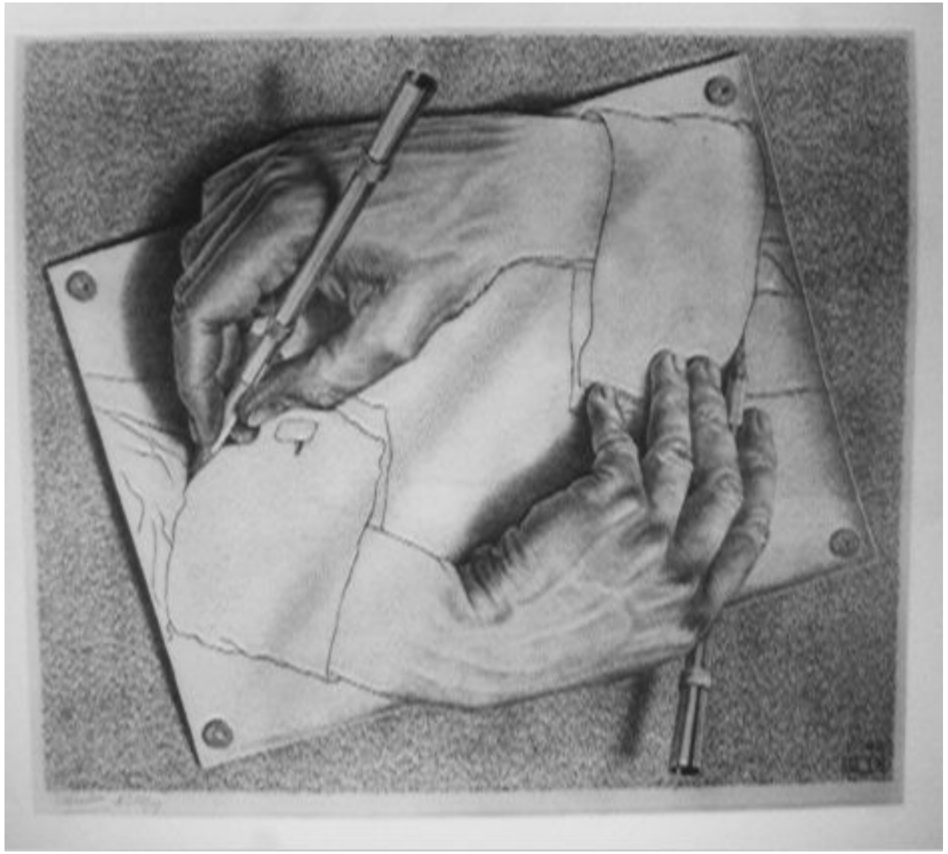
**Intractable as list grows in size;
not feasible for dynamic lists.**

# Self-modifying code

- In ASM a name is a location
  - `add   ax,[a]`     (source)
  - Becomes
  - `add   ax,0F9A`   (machine code)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| … | A0 | 05 | 9A | 0F | | | |

- Self-modifying code
  - Knows location of address
  - Increments the address



## Most basic form

- Declare a list

```
list dw 10 dup (0)
```

- Fill in values (not shown)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

## Most basic form

- Declare a list

```
list dw 10 dup (0)
```

- Fill in values (not shown)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00E4 | 10FF | CA56 | 9872 | 4C6F | 1234 | B00D | A5E7 | 66F4 | 0AA9 |

## Most basic form

- Declare a list

```
list dw 10 dup (0)
```

- Fill in values (not shown)
- Sum values

```
for i in 0..9: sum += list[i]
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00E4 | 10FF | CA56 | 9872 | 4C6F | 1234 | B00D | A5E7 | 66F4 | 0AA9 |

## Most basic form

- Declare a list

```
list dw 10 dup (0)
```

- Fill in values (not shown)
- Sum values

```
for i in 0..9: sum += list[i]
```

- Code it
- Index, i
  - cx
- Sum
  - ax
- Pointer
  - si

| | |
|---|---|
| Stack pointer | **SP** |
| Base pointer | **BP** |
| Source index | **SI** |
| **Destination idx** | **DI** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00E4 | 10FF | CA56 | 9872 | 4C6F | 1234 | B00D | A5E7 | 66F4 | 0AA9 |

## Most basic form

- Declare a list

```
list dw 10 dup (0)
```

- Fill in values (not shown)
- Sum values

```
for i in 0..9: sum += list[i]
```

- Code it

```
          mov    ax,0            ; sum = 0
          mov    cx,0            ; i = 0
          mov    si,offset list  ; si is index
calc:
          add    ax,[si]         ; add nxt val
          add    si,2            ; advance ptr
          inc    cx              ; i=i+1
          cmp    cx,10           ; is i < 10
          jb     calc            ; yes: repeat
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00E4 | 10FF | CA56 | 9872 | 4C6F | 1234 | B00D | A5E7 | 66F4 | 0AA9 |

# For C programmers

| C | Assembler |
|---|---|
| `int *si`<br>***Define*** **si as a pointer to integers** | `si` |
| `si = &list`<br>***Set*** **si to point to a list** | `mov si,offset list` |
| `sum = sum + *si`<br>***Add*** **the value pointed by si to sum** | `add ax,[si]` |

## Loop

```
label:  …
        …
        …
        …
```

## Loop instruction

```
        mov    cx,10
label:  …
        …
        …
        …
        loop   label
```

- Reduce the cx register by 1
- If cx != zero then jump to label
- Condition code is not modified

- Initialize cx with
  - Unsigned count
  - Greater than zero
- If cx == 0 before loop instruction
  - `0000 - 1 = FFFF`
  - Jump taken
  - 65,536 times

## Using loop

```
     mov  ax, 0           ; sum = 0
     mov  cx, 0           ; i = 0
     mov  si, offset list ; si index
calc:
     add  ax,[si]         ; add nxtl
     add  si,2            ; adv ptr
     inc  cx              ; i=i+1
     cmp  cx,10           ; i < 10?
     jne  calc            ; jmp
```

```
     mov  ax, 0           ; sum = 0
     mov  cx, 10          ; count = 10
     mov  si, offset list ; si is index
calc:
     add  ax,[si]         ; add nxt val
     add  si,2            ; advance ptr
     loop calc            ; reduce count
                          ; & repeat
```

## A problem

```
count   dw    6     ;count
list    dw    -3, 7, 100, -83, 0, 5 ;list

      cmp   [count], 0      ;is count zero
      je    fin             ;yes, done
      mov   si,offset list  ;point tolist
      mov   cx,[count]      ;set loop count
test: cmp   [si],0          ;is entry  0
      jge   next            ;yes, skip
      mov   [si],0          ;no, set to 0
next: add   si,2            ;advance pointer
      loop  test            ; loop
fin:
```

- What is the problem?

# A problem

```
count   dw    6    ;count
list    dw    -3, 7, 100, -83, 0, 5 ;list

      cmp   [count], 0      ;is count zero
      je    fin             ;yes, done
      mov   si,offset list  ;point tolist
      mov   cx,[count]      ;set loop count
test: cmp   [si],0          ;is entry  0
      jge   next            ;yes, skip
      mov   [si],0          ;no, set to 0
next: add   si,2            ;advance pointer
      loop  test            ; loop
fin:
```

- What is the problem?
  - `cmp    [si],0`

**Memory reference**

**Immediate constant**

## A problem

```
count    dw    6     ;count
list     dw    -3, 7, 100, -83, 0, 5 ;list

      cmp   [count], 0.      ;is count zero
      je    fin              ;yes, done
      mov   si,offset list   ;point tolist
      mov   cx,[count].      ;set loop count
test: cmp   [si],0           ;is entry  0
      jge   next             ;yes, skip
      mov   [si],0           ;no, set to 0
next: add   si,2             ;advance pointer
      loop  test             ; loop
fin:
```

- What is the problem?
    - `cmp    [si],0`

**Memory reference**

**Immediate constant**

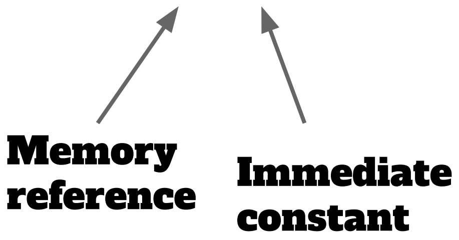
- What is size of [si]?
    - Byte?
    - Word?

# A problem

```
count    dw     6     ;count
list     dw     -3, 7, 100, -83, 0, 5 ;list

      cmp   [count], 0      ;is count zero
      je    fin             ;yes, done
      mov   si,offset list  ;point tolist
      mov   cx,[count]      ;set loop count
test: cmp   [si],0          ;is entry  0
      jge   next            ;yes, skip
      mov   [si],0          ;no, set to 0
next: add   si,2            ;advance pointer
      loop  test            ; loop
fin:
```

- What is the problem?
  - `cmp    [si],0`



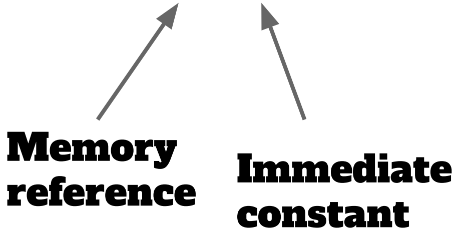
- What is size of [si]?
  - Byte?
  - Word?
- Tell hardware
  - Override

## A problem

```
count    dw    6     ;count
list     dw    -3, 7, 100, -83, 0, 5 ;list

      cmp   [count], 0       ;is count zero
      je    fin              ;yes, done
      mov   si,offset list   ;point tolist
      mov   cx,[count]       ;set loop count
test: cmp   word ptr [si],0 ;is entry  0
      jge   next             ;yes, skip
      mov   word ptr [si],0 ;no, set to 0
next: add   si,2             ;advance pointer
      loop  test             ; loop
fin:
```

- What is the problem?
  - `cmp    [si],0`



- What is size of [si]?
  - Byte?
  - Word?
- Tell hardware
  - Override
  - If size cannot be determined

# A problem

```
count    dw    6    ;count
list     dw    -3, 7, 100, -83, 0, 5 ;list

      cmp   word ptr [count], 0
      je    fin                ;yes, done
      mov   si,offset list     ;point tolist
      mov   cx,[count]         ;set loop count
test: cmp   word ptr [si],0    ;is entry  0
      jge   next               ;yes, skip
      mov   word ptr [si],0    ;no, set to 0
next: add   si,2               ;advance pointer
      loop  test               ; loop
fin:
```

- What is the problem?
  - `cmp    [si],0`

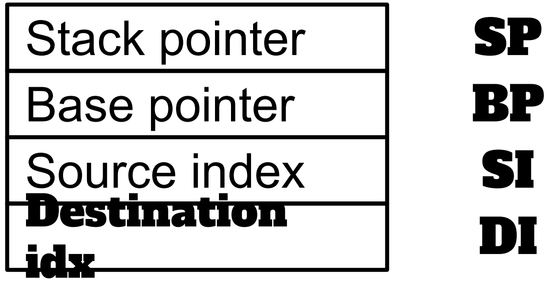
**Memory reference**

**Immediate constant**

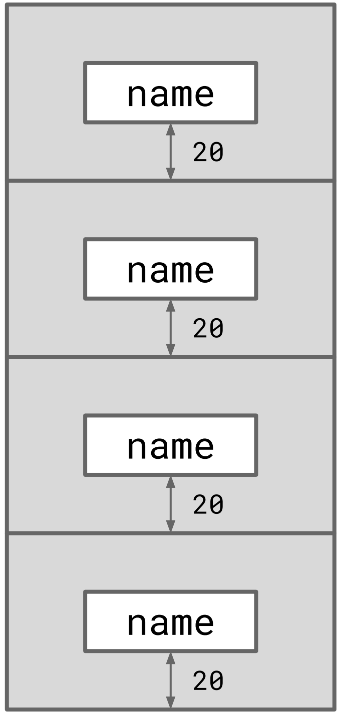
- What is size of [si]?
  - Byte?
  - Word?
- Tell hardware
  - Override
  - If size cannot be determined

# Pointer registers

- Data segment
  - [ si ]
  - [ di ]
  - [ bx ]
- Stack segment
  - [ bp ]

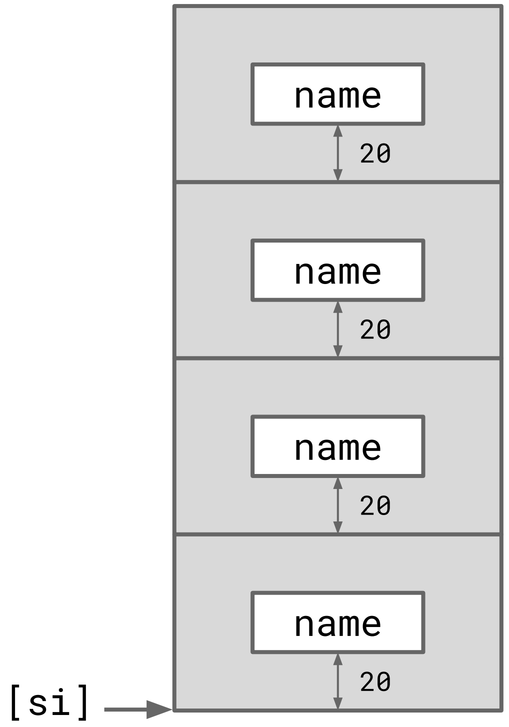


## Indirect addressing



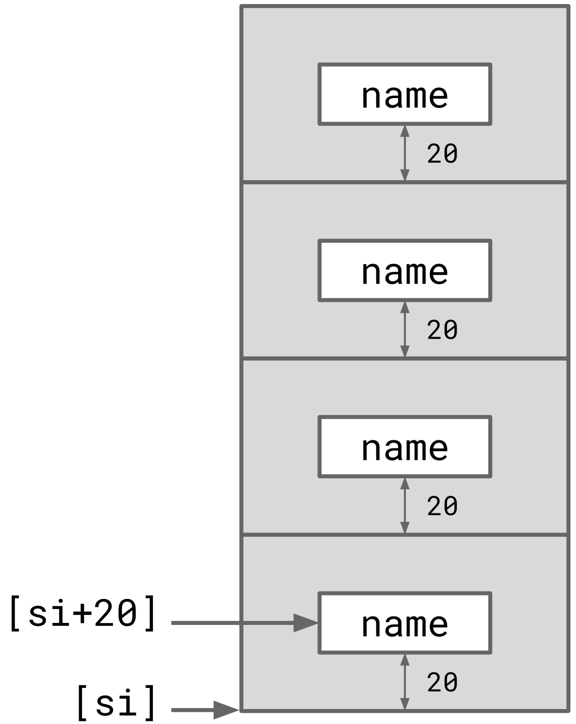
- Employee record
  - Multiple fields
  - e.g., name
- Many employees
  - Array (list)
  - Of records

# Indirect addressing



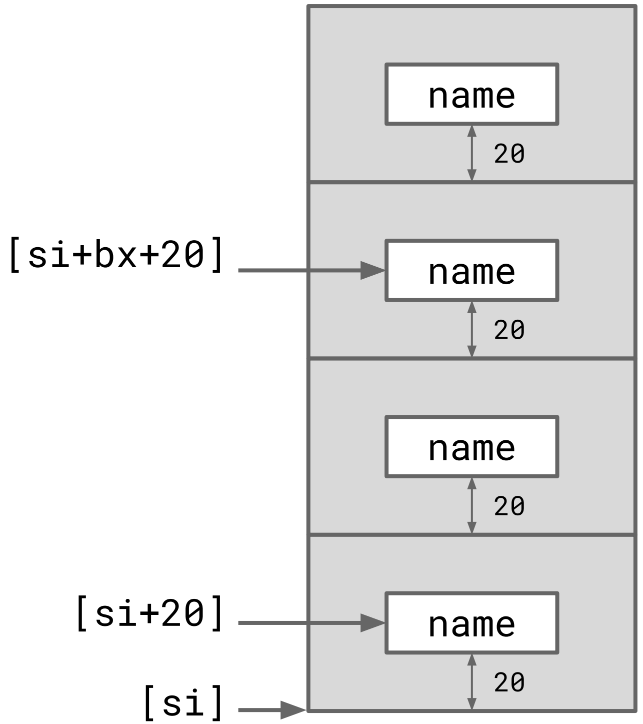
- Array starts at base
- Set register to point to base
    - `mov si,offset base`
- All fields and record offsets from base

## Indirect addressing

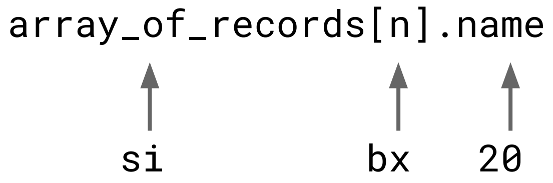


- name of first record is
  - si + 20

# Indirect addressing



- name of nth record is
  - n * sizeof(struct), ie 40
  - Zero indexed
  - `mov bx,40`
  - `mul bx,[n]`



| Components | Format |
|---|---|
| base or index | [bx] [bp] [si] [di] |
| base or index with displacement | [bx ± n] [bp ± n] [si ± n] [di ± n] |
| base and index | [bx + si] [bx + di] [bp + si] [bp + di] |
| base and index with displacement | [bx + si ± n] [bx + di ± n] [bp + si ± n] [bp + di ± n] |

**Only 16 valid combinations**

**Complete discussion see pg 10-7**

| Components | Format |
|---|---|
| **base or index** | [bx] [bp] [si] [di] |
| **base or index with displacement** | [bx ± n] [bp ± n]<br>[si ± n] [di ± n] |
| **base and index** | [bx + si] [bx + di]<br>[bp + si] [bp + di] |
| **base and index with displacement** | [bx + si ± n]<br>[bx + di ± n]<br>[bp + si ± n]<br>[bp + di ± n] |

**Only 16 valid combinations**

**Complete discussion see pg 10-7**

## Single pointer register

| Components | Format |
|---|---|
| **base or index** | `[bx] [bp] [si] [di]` |
| **base or index with displacement** | `[bx ± n] [bp ± n]`<br>`[si ± n] [di ± n]` |
| **base and index** | `[bx + si] [bx + di]`<br>`[bp + si] [bp + di]` |
| **base and index with displacement** | `[bx + si ± n]`<br>`[bx + di ± n]`<br>`[bp + si ± n]`<br>`[bp + di ± n]` |

**Only 16 valid combinations**

**Complete discussion see pg 10-7**

## Single pointer register plus constant offset

| Components | Format |
|---|---|
| base or index | [bx] [bp] [si] [di] |
| base or index with displacement | [bx ± n] [bp ± n] [si ± n] [di ± n] |
| base and index | [bx + si] [bx + di] [bp + si] [bp + di] |
| base and index with displacement | [bx + si ± n] [bx + di ± n] [bp + si ± n] [bp + di ± n] |

**Only 16 valid combinations**

**Only these 4 pairs {bx,bp}x{si,di}**

**Complete discussion see pg 10-7**

## Two pointer registers

| Components | Format |
|---|---|
| **base or index** | [bx] [bp] [si] [di] |
| **base or index with displacement** | [bx ± n] [bp ± n] [si ± n] [di ± n] |
| **base and index** | [bx + si] [bx + di] [bp + si] [bp + di] |
| **base and index with displacement** | [bx + si ± n] [bx + di ± n] [bp + si ± n] [bp + di ± n] |

**Only 16 valid combinations**

**Complete discussion see pg 10-7**

## Two pointer registers plus constant offset

| Components | Format |
|---|---|
| **base or index** | [bx] [bp] [si] [di] |
| **base or index with displacement** | [bx ± n] [bp ± n]<br>[si ± n] [di ± n] |
| **base and index** | [bx + si] [bx + di]<br>[bp + si] [bp + di] |
| **base and index with displacement** | [bx + si ± n]<br>[bx + di ± n]<br>[bp + si ± n]<br>[bp + di ± n] |

**Only 16 valid combinations**

- **disp is immediate**
- **disp is not variable**
- **bp accesses stack seg**

**Complete discussion see pg 10-7**

## What is the fastest way to solve a problem?

- Lookup the answer
    - square = SquareTable[n]
- Not always the faster
    - If calculation is simple
    - Can be less expensive than the math for indexing into an array.

| index | value |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |
| … | $…^2$ |

## Lookup answer

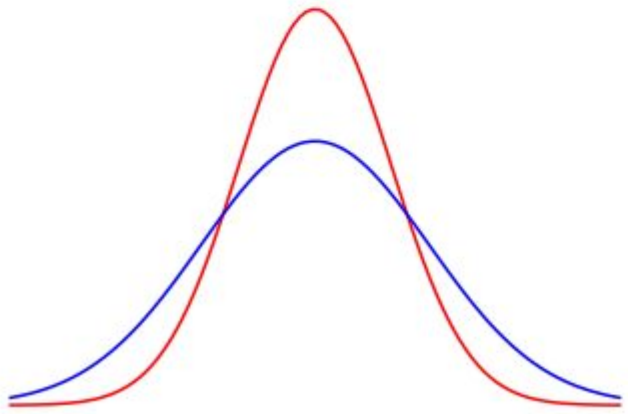
- Consider a BW picture
- Millions of pixels
  - Picture elements
  - Hold grayscale value
- Suppose byte - unsigned
  - 256 levels from black to white

- Increase brightness
  - Increase value of each pixel
  - Add a constant to each pixel
- Straightforward solution
  - $pixel_i = min(255, pixel_i+20)$
  - efficient

## Lookup answer

- Consider a BW picture
- Millions of pixels
  - Picture elements
  - Hold grayscale value
- Suppose byte
  - 256 levels from black to white

- Increase contrast
  - Make white, whiter; black, blacker
  - Increase “spread” of the pixels



## Lookup answer

- Consider a BW picture
- Millions of pixels
  - Picture elements
  - Hold grayscale value
- Suppose byte
  - 256 levels from black to white

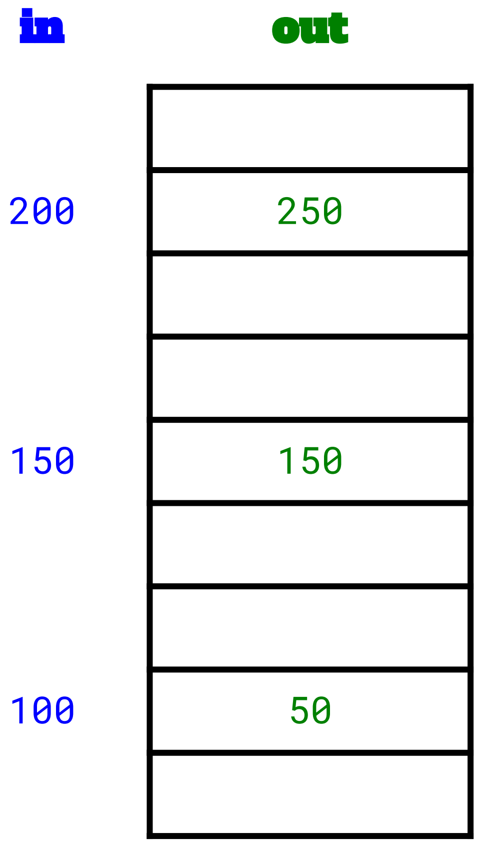
- Increase contrast
  - Make white, whiter; black, blacker
  - Increase “spread” of the pixels
- Formula

$$new = (old - oldlow) \times \frac{newhi - newlow}{oldhi - oldlow} + newlow$$

- Calculate on each pixel
  - The middle term is same
  - Subtract, multiply, add
- Slow, inefficient

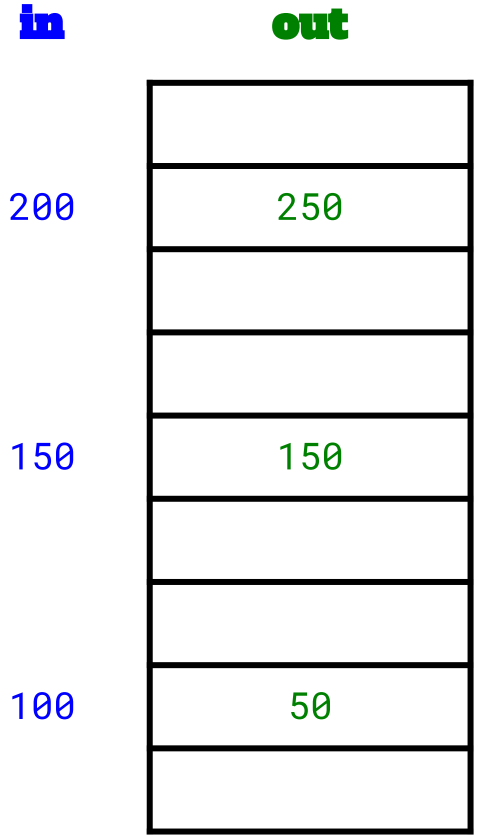
## Lookup answer

- 64,000 x 64,000 image
  - That's 4 billion pixels
- Only 256 different intensity values
- Calculate a table
  - Once for each possible value
  - Just 256 complex calculations
- Lookup for each pixel
- Generalization of xlat



## Lookup answer

```
mov   si,offset table *
mov   bl,[inpix]
mov   bl,[si+bl]
```

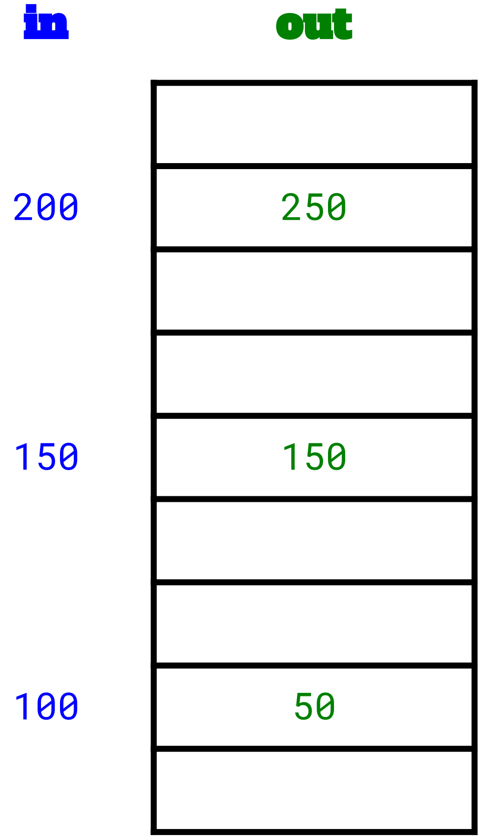


*** simplified code**

## Lookup answer

```
mov   si,offset table *
mov   bl,[inpix]
mov   bl,[si+bl]
```

Oops. This doesn't work.

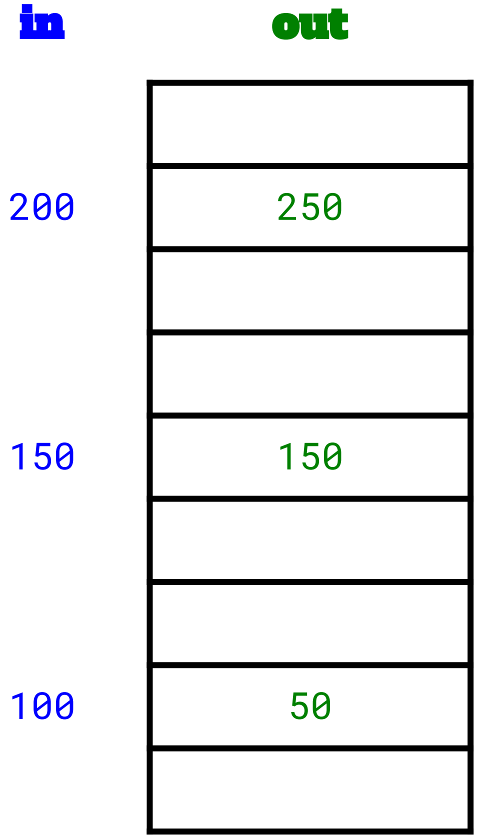


* simplified code

# Lookup answer

```
mov   bx, 0
mov   si,offset table *
mov   bl,[inpix]
mov   bl,[si+bx]
```

That's better.



*** simplified code**

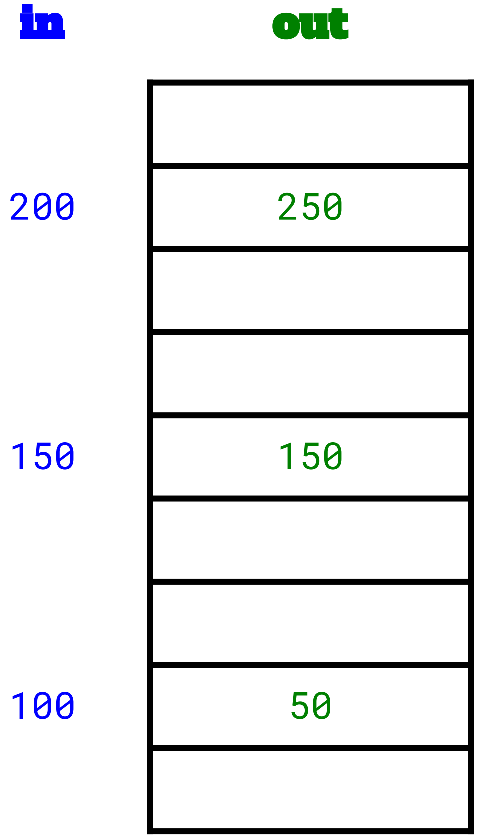
## Lookup answer

```
mov   si,offset table *
mov   bl,[inpix]
mov   bl,[si+bx]
```

***Peterson's Law:***

**Nothing is so complicated it cannot be solved with another level of indirection**

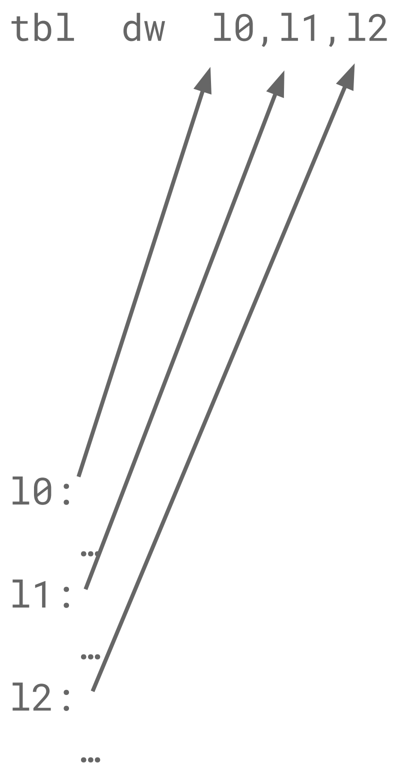
*** simplified code**



## Jump tables

- Based on some
    - input, or index, or key
    - directly jump to code
    - without any compares
- Example
    - Read a key: ‘0’, ‘1’, ‘2’
    - Jump to corresponding routine

## Jump tables



```
tbl  dw  l0,l1,l2



l0:
    …
l1:
    …
l2:
    …
```

- Based on some
  - input, or index, or key
  - directly jump to code
  - without any compares
- Example
  - Read a key: ‘0’, ‘1’, ‘2’
  - Jump to corresponding routine

## Jump tables

```
tbl  dw  l0,l1,l2

     mov  ah,8   ;code to read char
     int  21h    ;read '2' ax=08 32
     and  0003h  ;ax=00 02
     add  ax,ax  ;ax=00 04
     mov  bx,ax  ;move to index
     jmp  [tbl+bx]

l0:
   …
l1:
   …
l2:
   …
```

- Based on some
  - input, or index, or key
  - directly jump to code
  - without any compares
- Example
  - Read a key: ‘0’, ‘1’, ‘2’
  - Jump to corresponding routine

# Jump tables

```
tbl  dw  l0,l1,l2

     mov  ah,8   ;code to read char
     int  21h    ;read ’2’ ax=08 32
     and  0003h  ;ax=00 02
     add  ax,ax  ;ax=00 04
     mov  bx,ax  ;move to index
     jmp  [tbl+bx]

l0:
   …
l1:
   …
l2:
   …
```

- Wait -- You cannot use variables
- How does this work?
- tbl
  - is not a variable
  - It is a location in memory
  - Constant at compile-time
  - Similar to immediate value

## Indirect with Variables

```
list   db   10 dup(?)    ; list
n      dw    0           ; index into list


mov    al,0              ; sum = 0
mov    si,offset list    ; points to list
add    al,[si + n]       ; add next item
```

- Does this work?

## Indirect with Variables

```
list   db   10 dup(?)    ; list
n      dw    0           ; index into list


mov    al,0              ; sum = 0
mov    si,offset list    ; points to list
~~add    al,[si + n]       ; add next item~~
mov    bx,[n]            ; put index in reg
add    al,[si + bx]      ; add next item
```

- Does not work
  - No memory reference
  - ‘n’ is address