

Introduction to 8086 Assembly

Lecture 16

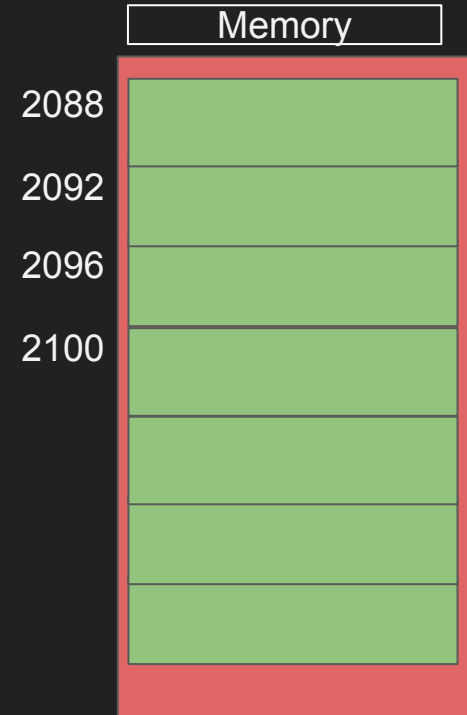
Implementing Arrays

Arrays



K. N. Toosi
University of Technology

- A list of elements all of same size (and type)
- Accessing array element
 - starting address in memory
 - element size
 - index
 - index of first element (0 or 1?)
 - no. of elements (array size)?

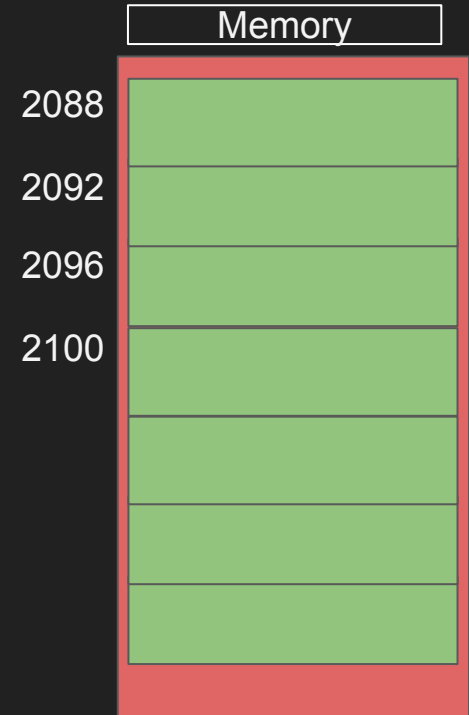


Defining arrays



K. N. Toosi
University of Technology

- Define arrays
 - In data segment (e.g. global arrays)
 - absolute address (global label)
 - In stack (e.g. local arrays)
 - relative address (relative to esp or ebp)



Global labels

```
segment .data

arr1:  db  1,3,6,10, 15, 21, 28
arr2:  dw  0, 0, 0, 0, 0, 0
arr3:  dd  10, 100, 1000, 10000, 100000
arr4:  times 64 dd 20

segment .bss

arr5:  resb 100
arr6:  resw 200
arr7:  resd 50
arr8:  resq 400
```

start address: `arr1`
element size: 1 byte
array size: 7 elements (7 bytes)

start address: `arr2`
element size: 2 bytes
array size: 6 elements (12 bytes)

start address: `arr3`
element size: 4 bytes
array size: 5 elements (20 bytes)

start address: `arr4`
element size: 4 bytes
array size: 64 elements (256 bytes)

start address: `arr8`
element size: 8 bytes
array size: 400 elements (3200 bytes)





Arrays on stack (as local variable)

func:

```
push ebp
mov  ebp, esp

;; just a single local variable (array)
sub  esp, 400
```

start address: $ebp-400$
element size: 1 byte
array size: 400 elements

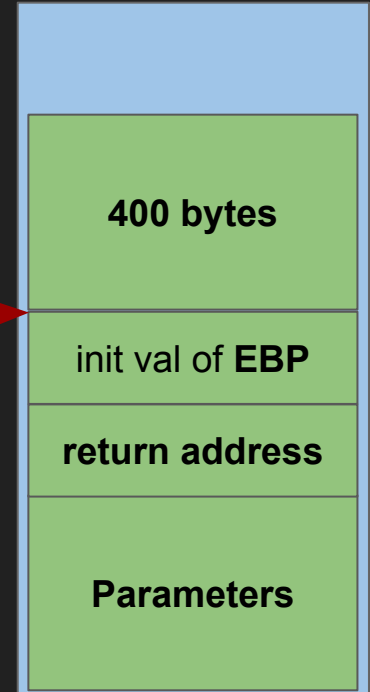
OR

start address: $ebp-400$
element size: 2 bytes
array size: 200 elements

OR

start address: $ebp-400$
element size: 4 bytes
array size: 100 elements

EBP



Access array elements



- Use indirect addressing

```
segment .data
arr1:  db  1,3,6,10, 15, 21, 28
arr2:  dw  0, 0, 0, 0, 0, 0
arr3:  dd  10, 100, 1000, 10000, 100000
arr4:  times 64 dd 20

segment .bss
arr5:  resb 100
arr6:  resw 200
arr7:  resd 50
arr8:  resq 400
```

```
mov al, [arr1+3]
```

```
mov ax, [arr2+2]
```

```
mov eax, [arr3+8]
```

```
mov eax, [arr3+3]
```

```
mov ecx, 12
```

```
mov dword [arr7+ecx], -200
```

Access array elements



```
segment .data
array1: dd 1, 2, 4, 8, 16, 32

segment .text
global asm_main
extern print_int, print_nl

asm_main:
    pusha

    mov ecx, 6 ; array size
    mov ebx, 0

loop1:
    mov eax, [array1+ebx]
    call print_int
    call print_nl

    add ebx, 4
    loop loop1

    popa
    ret
```

array2.asm

```
segment .data
array1: dd 1, 2, 4, 8, 16, 32

segment .text
global asm_main
extern print_int, print_nl

asm_main:
    pusha

    mov ecx, 6 ; array size
    mov ebx, array1

loop1:
    mov eax, [ebx]
    call print_int
    call print_nl

    add ebx, 4
    loop loop1

    popa
    ret
```

array3.asm

Exercise



- Write a function to print an array of double word integers.

```
void printArray(const int a[], int n) {  
    for (int i = 0; i < n; i++)  
        printf("%d, ", a[i]);  
    putchar('\n');  
}
```

```
%include "asm_io.inc" array4.asm  
  
segment .data  
  
array1: dd    1, 2, 4, 8, 16, 32  
  
segment .text  
        global asm_main  
  
asm_main:  
        pusha  
  
        push 6  
        push array1  
        call printArray  
  
        popa  
        ret
```


Exercise

- Write a function to print an array of double word integers.

```
void printArray(const int a[], int n) {  
    for (int i = 0; i < n; i++)  
        printf("%d, ", a[i]);  
    putchar('\n');  
}
```

```
; printArray(int ARRAY[], int SIZE)  
%define ARRAY [ebp+8]  
%define SIZE [ebp+12]
```

printArray:

```
push ebp  
mov ebp, esp
```

```
mov ebx, ARRAY  
mov ecx, SIZE
```

loop1:

```
mov eax, [ebx]  
call print_int  
mov al, ','  
call print_char  
mov al, '  
call print_char
```

```
add ebx, 4  
loop loop1
```

```
mov al, 10  
call print_char
```

```
mov esp, ebp  
pop ebp  
ret 8
```

array4.asm



Advanced indirect addressing



K. N. Toosi
University of Technology

`mov eax, [ecx]`

`mov eax, [ecx + constant]`

`mov eax, [4 * ecx + constant]`

`mov eax, [ebx + 4 * ecx + constant]`

`[mymovxyz eax, ebx, 4, ecx, immed]`

Advanced indirect addressing



K. N. Toosi
University of Technology

[**base-reg** + **scale** * **index-reg** + **constant**]

scale: **1,2,4,8**

base-reg: **EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI**

index-reg: **EAX, EBX, ECX, EDX, EBP, ESI, EDI (not ESP)**

constant: **label or immediate**

Advanced indirect addressing



```
segment .data                                     array3.asm
array1: dd 1, 2, 4, 8, 16, 32

segment .text
global asm_main
extern print_int, print_nl

asm_main:
    pusha

    mov ecx, 6 ; array size
    mov ebx, 0

loop1:
    mov eax, [ebx+array1]
    call print_int
    call print_nl

    add ebx, 4
    loop loop1

    popa
    ret
```

```
segment .data                                     array5.asm
array1: dd 1, 2, 4, 8, 16, 32

segment .text
global asm_main
extern print_int, print_nl

asm_main:
    pusha

    mov ecx, 6 ; array size
    mov ebx, 0

loop1:
    mov eax, [4*ebx+array1]
    call print_int
    call print_nl

    inc ebx
    loop loop1

    popa
    ret
```

local variables/arrays



```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

local variables/arrays



K. N. Toosi
University of Technology

```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

```
myfunc:  
    push ebp  
    mov  ebp, esp  
    sub  esp, 4+4+100*4  
  
    mov  esp, ebp  
    pop  ebp  
    ret
```

local variables/arrays



K. N. Toosi
University of Technology

```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

myfunc:

```
push ebp  
mov  ebp, esp  
sub  esp, 4+4+100*4  
                                     immediate  
  
mov  esp, ebp  
pop  ebp  
ret
```

local variables/arrays



```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

```
myfunc:                                     array6.asm  
    push ebp  
    mov  ebp, esp  
    sub  esp, 4+4+100*4  
  
    mov  ecx, 0  
beginloop:  
    cmp  ecx, 100  
    jge  endloop  
  
    mov  eax, ecx  
    mul  ecx  
  
    mov  [ebp+4*ecx-408], eax  
  
    inc  ecx  
    jmp  beginloop  
endloop:
```

EBP



a (400 bytes)

j (4 bytes)

k (4 bytes)

pushed EBP

return address

local variables/arrays

```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

```
array6.asm  
myfunc:  
    push ebp  
    mov  ebp, esp  
    sub  esp, 4+4+100*4  
  
    mov  ecx, 0  
beginloop:  
    cmp  ecx, 100  
    jge  endloop  
  
    mov  eax, ecx  
    mul  ecx  
  
    mov  [ebp+4*ecx-408], eax  
  
    inc  ecx  
    jmp  beginloop  
endloop:  
  
    mov  eax, ebp  
    sub  eax, 408  
    push 100  
    push eax  
    call printArray  
  
    mov  esp, ebp  
    pop  ebp  
    ret
```



local variables/arrays



K. N. Toosi
University of Technology

```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

endloop:

array6.asm

```
    mov    eax, ebp  
    sub    eax, 408  
    push  100  
    push  eax  
    call  printArray  
  
    mov    esp, ebp  
    pop   ebp  
    ret
```

load effective address



K. N. Toosi
University of Technology

endloop:

array6.asm

```
mov  eax, ebp
sub  eax, 408
push 100
push eax
call printArray

mov  esp, ebp
pop  ebp
ret
```

endloop:

array7.asm

```
lea  eax, [ebp-408]
push 100
push eax
call printArray

mov  esp, ebp
pop  ebp
ret
```

load effective address



K. N. Toosi
University of Technology

endloop:

array6.asm

```
mov  eax, ebp
sub  eax, 408
push 100
push eax
call printArray

mov  esp, ebp
pop  ebp
ret
```

endloop:

array7.asm

```
lea  eax, [ebp-408]
push 100
push eax
call printArray

mov  esp, ebp
pop  ebp
ret
```

address generation unit (AGU)

final program

```
void myfunc() {  
    int k;  
    int j;  
    int a[100];  
  
    for (int i = 0; i < 100; i++) {  
        a[i] = i*i;  
    }  
  
    printArray(a,100);  
}
```

```
myfunc:                                     array7.asm  
    push ebp  
    mov  ebp, esp  
    sub  esp, 4+4+100*4  
  
    mov  ecx, 0  
beginloop:  
    cmp  ecx, 100  
    jge  endloop  
  
    mov  eax, ecx  
    mul  ecx  
  
    mov  [ebp+4*ecx-408], eax  
  
    inc  ecx  
    jmp  beginloop  
endloop:  
    lea  eax, [ebp-408]  
    push 100  
    push eax  
    call printArray  
  
    mov  esp, ebp  
    pop  ebp  
    ret
```



load effective address



K. N. Toosi
University of Technology

```
mov reg, [ base-reg + scale * index-reg + constant ]
```

```
reg = *(base-reg + scale * index-reg + constant)
```

```
lea reg, [ base-reg + scale * index-reg + constant ]
```

```
reg = base-reg + scale * index-reg + constant
```

get address of local variables / arrays



- storing a pointer to a local variable
- pushing on stack for function call

```
endloop:  
  
    lea eax, [ebp-408]  
    push 100  
    push eax  
    call printArray  
  
    mov esp, ebp  
    pop ebp  
    ret
```

get address of local variables / arrays



- storing a pointer to a local variable
- pushing on stack for function call

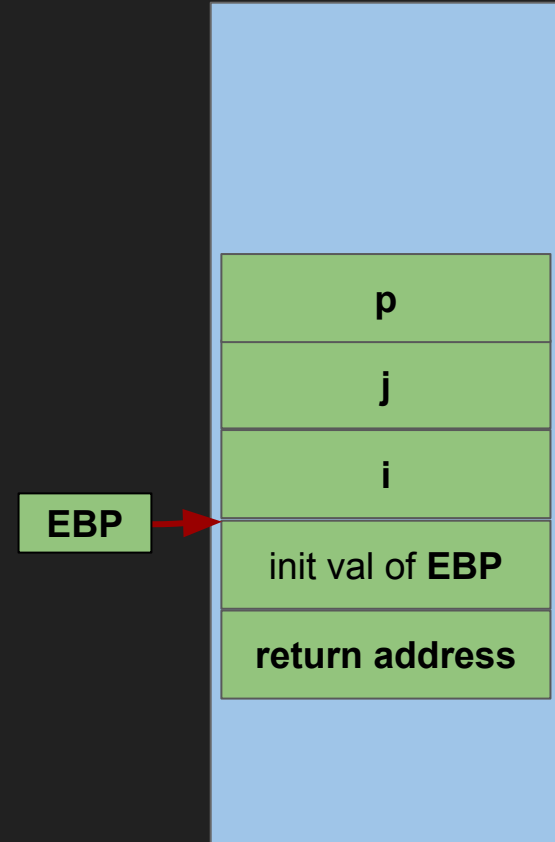
```
endloop:  
  
    lea eax, [ebp-408]  
    push 100  
    push eax  
    call printArray  
  
    mov esp, ebp  
    pop ebp  
    ret
```


get address of local variables / arrays



- storing a pointer to a local variable
- pushing on stack for function call

```
void myfunc() {  
    int i;  
    int j;  
    int *p;  
  
    p = &j;  
  
}
```



get address of local variables / arrays

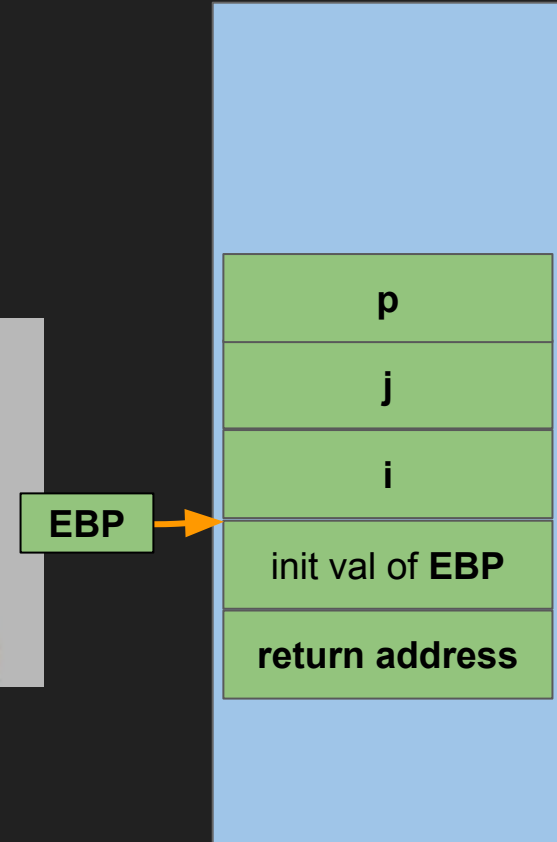


- storing a pointer to a local variable
- pushing on stack for function call

```
void myfunc() {  
    int i;  
    int j;  
    int *p;  
  
    p = &j;  
  
}
```

```
myfunc:  
    push ebp  
    mov  ebp, esp  
    sub  esp, 4+4+4  
  
    lea  eax, [ebp-8]  
    mov  [ebp-12], eax
```

assuming 32-bit addressing
(pointers are 32-bits)



fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]
```

```
EAX *= 5
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]
```

```
EAX *= 5
```

```
????
```

```
EAX *= 6
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]
```

```
EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]
```

```
EAX *= 6
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]      EAX *= 6
```

```
nasihatkon@kntu:code$ nasm -f elf lea.asm  
lea.asm:21: error: invalid effective address
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]      EAX *= 6
```

```
nasihatkon@kntu:code$ nasm -f elf lea.asm  
lea.asm:21: error: invalid effective address
```

[base-reg + scale * index-reg + constant]

scale: 1,2,4,8

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]      EAX *= 6
```

```
nasihatkon@kntu:code$ nasm -f elf lea.asm  
lea.asm:21: error: invalid effective address
```

```
lea EAX, [ EAX + 2 * EAX ]
```

```
sal EAX
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]      EAX *= 6
```

```
nasihatkon@kntu:code$ nasm -f elf lea.asm  
lea.asm:21: error: invalid effective address
```

```
lea EAX, [ EAX + 8 * EAX ]
```

```
lea EAX, [ EAX + 4 * EAX ]
```

fast computations



K. N. Toosi
University of Technology

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 5
```

```
lea EAX, [ EAX + 5 * EAX ]      EAX *= 6
```

```
nasihatkon@kntu:code$ nasm -f elf lea.asm  
lea.asm:21: error: invalid effective address
```

```
lea EAX, [ EAX + 8 * EAX ]
```

```
lea EAX, [ EAX + 4 * EAX ]      EAX *= 45
```

Arrays in inline assembly



array9.c

```
void printArray(const int a[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d, ", a[i]);
    putchar('\n');
}

int main() {
    int array[10] = {1,2,3,4,5,6,7,8,9,10};
    printArray(array,10);

    for (int i = 0; i < 10; i++) {
        asm volatile ("mov eax, [ebx+4*esi];"
                    "lea eax, [eax+8*eax];"
                    "mov [ebx+4*esi], eax"
                    :
                    : "b" (array), "S" (i)
                    : "memory", "eax");
    }

    printArray(array,10);
}
```

Arrays in inline assembly



array9.c

```
void printArray(const int a[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d, ", a[i]);
    putchar('\n');
}

int main() {
    int array[10] = {1,2,3,4,5,6,7,8,9,10};
    printArray(array,10);

    for (int i = 0; i < 10; i++) {
        asm volatile ("mov eax, [ebx+4*esi];"
                    "lea eax, [eax+8*eax];"
                    "mov [ebx+4*esi], eax"
                    :
                    : "b" (array), "S" (i)
                    : "memory", "eax");
    }

    printArray(array,10);
}
```

```
b.nasihatkon@kntu:lecture16$ gcc -m32 -masm=intel array9.c && ./a.out
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
9, 18, 27, 36, 45, 54, 63, 72, 81, 90,
```