CS 271
Computer Architecture &
Assembly Language

Lecture 14
Array
Random Number
*Local Variables
2/17/22, Thursday
Odds and Ends

• Program 5 Clarifications

• Due Sunday 2/20 11:59 pm:
  • Weekly Summary 7
Lecture Topics:

• Introduction to Arrays
• Arrays as Reference Parameters
• Display an Array Sequentially
• “Random” Numbers
Recall: Introduction to Arrays
Recall: Array References in MASM

• Several methods for accessing specific array elements
  • Indexed
  • Register indirect
  • Base-indexed
Recall: Indexed Addressing

- Array name, with “distance” to element in a register
  - Used for global array references (not used in Program #5)
- Examples:

```
mov     edi, 0                      ; high-level notation
mov     list[edi], eax            ; is list[0]
add     edi, 4                     ; * see note below
mov     list[edi], ebx            ; list[1]
```

- This means “add the value in [] to address of list”
- *Note: add 4 because these array elements are DWORD
  - If BYTE, add 1
  - If WORD, add 2
  - If QWORD, add 8
  - Etc.
Recall: Register Indirect Addressing

- Actual address of array element in register
  - Used for referencing array elements in procedures

- Examples:
  - In calling procedure...
    ```
    push OFFSET list
    ```
  - In called procedure... (example only)
    ```
    push ebp
    mov esi, [ebp+8] ; get address of list into esi
    mov eax, [esi]  ; get list[0] into eax
    add esi, 4
    add eax, [esi]  ; add list[1] to eax
    add esi, 16
    mov [esi], eax  ; send result to list[5]
    ```
Recall: Base-indexed Addressing

- Starting address in one register, offset in another; add and access memory
  - Used for referencing array elements in procedures

- Examples:
  - In calling procedure ...
    ```
    push OFFSET list
    ```
  - In called procedure ... (example only)

```plaintext
... ; set up stack frame
mov edx, [ebp+8] ; get address of list into edx
mov ecx, 20
mov eax, [edx+ecx] ; get list[5] into eax
mov ebx, 4
add eax, [edx+ebx] ; add list[1] to eax
mov [edx+ecx], eax ; send result to list[5]
```
Passing Arrays by Reference

• Never pass an array by value!!!

• Suppose that an *ArrayFill* procedure fills an array with 32-bit integers

• The calling program passed the address of the array, along with *count* of the number of array elements:

```
COUNT = 100
.data
list DWORD COUNT DUP(?)
.code
...
push OFFSET list
push COUNT
call ArrayFill
```

\[ \text{ArrayFill (list, count)} \]
Passing Arrays by Reference

- **ArrayFill** can reference an array without knowing the array’s name:

  ```asm
  ArrayFill PROC
  push ebp
  -> mov ebp,esp
  mov edi,[ebp+12] ;@list in edi
  mov ecx,[ebp+8] ;value of count in ecx
  ; ... etc.
  ```

- **edi** points to the beginning of the array, so it’s easy to use a loop to access each array element.

- Style note: We use **edi** because the array is the “destination”
Passing Arrays by Reference

• This *ArrayFill* uses *register indirect* addressing:

```assembly
ArrayFill PROC
    push ebp
    mov ebp, esp
    mov edi, [ebp+12] ; @list in edi
    mov ecx, [ebp+8] ; value of count in ecx
more:
    ; .
    ; Code to generate a random number in eax
    ; goes here.
    mov [edi], eax
    add edi, 4
    loop more
    pop ebp
    ret 8
ArrayFill ENDP
```
Passing Arrays by Reference

- This *ArrayFill* uses base-indexed addressing, saves registers:

```assembly
ArrayFill    PROC
    pushad                   ; save all registers
    mov    ebp, esp
    mov    edx, [ebp+40]     ; @list in edx
    mov    ebx, 0            ; "index" in ebx
    mov    ecx, [ebp+36]     ; value of count in ecx
    more:
    ;
    ; Code to generate a random number in eax
    ; goes here.
    mov    [edx+ebx], eax
    add    ebx, 4
    loop   more

    popad                   ; restore all registers
    ret 8                   
ArrayFill    ENDP
```
Lecture Topics:

- Introduction to Arrays
- Arrays as Reference Parameters
- Display an Array Sequentially
- “Random” Numbers
Setup in Calling Procedure

.data
list DWORD 100 DUP(?)
count DWORD 0

.code
;...
; code to initialize list and count
;...
;set up parameters and call display
push OFFSET list ;@list
push count ; number of elements
call display
;...
display PROC
    push ebp
    mov ebp,esp
    mov esi,[ebp+12] ;@list
    mov ecx,[ebp+8] ;ecx is loop control
more:
    mov eax,[esi] ;get current element
    call WriteDec
    call Crlf
    add esi,4 ;next element
    loop more
endMore:
    pop ebp
    ret 8
display ENDP
display PROC
push ebp
mov ebp,esp
mov esi, [ebp+12] ; @list
mov ecx, [ebp+8] ; ecx is loop control
mov edx, 0 ; edx is element "pointer"
more:
    mov eax, [esi+edx] ; get current element
call WriteDec
call Crlf
add edx, 4 ; next element
loop more
endMore:
    pop ebp
    ret 8
display ENDP
Random Numbers

- Irving library has random integer generator
  - “pseudo-random” numbers
- Randomize procedure
  - Initialize sequence based on system clock (random seed)
  - Call once at the beginning of the program
  - Without Randomize, program gets the same sequence every time it is executed
Limiting Random Values

• **RandomRange** procedure
  • Accepts N>0 in eax
  • Returns random integer in [0 ... N-1] in eax

• To generate a random number in [lo ... hi]:
  • Find number of integer possible in [lo ... hi]: \( \text{range} = \text{hi} - \text{lo} + 1 \)
  • Put range in eax, and call RandomRange
  • Result in eax is in [0 ... range -1]
  • Add lo to eax.
RandomRange Example

- Get a random integer in range \([18 \ldots 31]\)

\[
\begin{align*}
call & \quad \text{Randomize} \\
mov & \quad \text{eax}, \text{hi} \quad ; 31 \\
sub & \quad \text{eax}, \text{lo} \quad ; 31-18 = 13 \\
inc & \quad \text{eax} \quad ; 14 \\
call & \quad \text{RandomRange} \quad ; \text{eax in } [0..13] \\
add & \quad \text{eax}, \text{lo} \quad ; \text{eax in } [18..31]
\end{align*}
\]
*Additional Topics:

• Local Variables in Assembly
• LEA instruction

*will NOT be tested!
Local Variables

• Local Variables: created, used, and destroyed within a single subroutine (function, control structure, or loops).

• Local Variables are allocated on the runtime stack, below EBP

• Cannot be assigned default values at assembly time, but can be initialized at runtime
Local Variable Example

• In HLL:
  ```c
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

• In Assembly
  ```assembly
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```
Local Variable Visualization

- In HLL:
  ```
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

- In Assembly
  ```assembly
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  
  Func ENDP
  ```
Local Variable Visualization

- In HLL:
  ```c
  void func() {
      int x = 10;
      int y = 20;
  }
  ```

- In Assembly
  ```asm
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```

System Stack

<table>
<thead>
<tr>
<th>ESP</th>
<th>old EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ESP]</td>
<td>old EBP</td>
</tr>
<tr>
<td>[ESP + 4]</td>
<td>return @</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>ESP</th>
<th>[ESP]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ESP]</td>
<td>old EBP</td>
</tr>
<tr>
<td>[ESP + 4]</td>
<td>return @</td>
</tr>
</tbody>
</table>
```
Local Variable Visualization

• In HLL:
  ```c
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

• In Assembly
  ```assembly
  func  PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func  ENDP
  ```

System Stack

<table>
<thead>
<tr>
<th>[EBP]</th>
<th>old EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Local Variable Visualization

- In HLL:
  ```
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

- In Assembly:
  ```
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```

### System Stack

<table>
<thead>
<tr>
<th>ESP</th>
<th>[EBP - 8]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[EBP - 4]</td>
</tr>
<tr>
<td>EBP</td>
<td>[EBP]</td>
</tr>
<tr>
<td></td>
<td>old EBP</td>
</tr>
<tr>
<td></td>
<td>[EBP + 4]</td>
</tr>
<tr>
<td></td>
<td>return @</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Local Variable Visualization

• In HLL:

```c
void func() {
    int x = 10;
    int y = 20;
}
```

• In Assembly

```assembly
func PROC
    push ebp
    mov ebp, esp
    sub esp, 8
    mov DWORD PTR [ebp – 4], 10
    mov DWORD PTR [ebp – 8], 20
    mov esp, ebp
    pop ebp
ret
Func ENDP
```

System Stack

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP</td>
<td>EBP</td>
</tr>
<tr>
<td>[EBP – 8]</td>
<td>[EBP]</td>
</tr>
<tr>
<td>[EBP – 4]</td>
<td>old EBP</td>
</tr>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
</tr>
</tbody>
</table>

... ...
Local Variable Visualization

- In HLL:
  ```c
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

- In Assembly
  ```assembly
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp – 4], 10
  mov DWORD PTR [ebp – 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```

---

### System Stack

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EBP – 8]</td>
<td>20</td>
</tr>
<tr>
<td>[EBP – 4]</td>
<td>10</td>
</tr>
<tr>
<td>[EBP]</td>
<td>old EBP</td>
</tr>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Local Variable Visualization

- In HLL:
  ```
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

- In Assembly
  ```
  func PROC
    push ebp
    mov ebp, esp
    sub esp, 8
    mov DWORD PTR [ebp - 4], 10
    mov DWORD PTR [ebp - 8], 20
    mov esp, ebp
    pop ebp
    ret
  Func ENDP
  ```

System Stack

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EBP - 8]</td>
<td>20</td>
</tr>
<tr>
<td>[EBP - 4]</td>
<td>10</td>
</tr>
<tr>
<td>[EBP]</td>
<td>old EBP</td>
</tr>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Local Variable Visualization

• In HLL:

```c
void func() {
    int x = 10;
    int y = 20;
}
```

• In Assembly

```assembly
func PROC
    push ebp
    mov ebp, esp
    sub esp, 8
    mov DWORD PTR [ebp - 4], 10
    mov DWORD PTR [ebp - 8], 20
    mov esp, ebp
    pop ebp
    ret
Func ENDP
```

System Stack

<table>
<thead>
<tr>
<th>[ESP]</th>
<th>return @</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ESP</th>
<th>[ESP]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>old EBP</td>
<td></td>
</tr>
</tbody>
</table>

EBP: 30
Local Variable Visualization

- In HLL:
  ```
  void func() {
      int x = 10;
      int y = 20;
  }
  ```

- In Assembly
  ```
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```

System Stack

<table>
<thead>
<tr>
<th>ESP</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EBP
**Local Variable Example**

- **In HLL:**
  ```c
  void func() {
    int x = 10;
    int y = 20;
  }
  ```

- **In Assembly**
  ```assembly
  func PROC
  push ebp
  mov ebp, esp
  sub esp, 8
  mov DWORD PTR [ebp - 4], 10
  mov DWORD PTR [ebp - 8], 20
  mov esp, ebp
  pop ebp
  ret
  Func ENDP
  ```

What if this step is omitted?

*Ebp, return to invalid address!*
*Additional Topics:

• Local Variables in Assembly

• LEA instruction

*will NOT be tested!
LEA: Load Effective Address

• LEA: returns the address of an **indirect operand** (offset calculated during runtime)
LEA Example

• In HLL:
void create_arr(){
    char arr[30];
    for (int i = 0; i < 30; i++)
        arr[i] = '*' ;
}

• In Assembly
create_arr PROC
    push ebp
    mov ebp, esp
    sub esp, 32
    lea esi, [ebp-30]
    mov ecx, 30
L1:
    mov BYTE PTR [esi], '*'
    inc esi
    loop L1
    add esp, 32
    pop ebp
    ret
create_arr ENDP
LEA Visualization

• In Assembly

create_arr PROC
push ebp
mov ebp, esp
sub esp, 32
lea esi, [ebp-30]
mov ecx, 30
L1:
    mov BYTE PTR [esi], ‘*’
    inc esi
    loop L1
add esp, 32
pop ebp
ret
create_arr ENDP

• In HLL:

```c
void create_arr()
{
    char arr[30];
    for (int i = 0; i < 30; i++)
        arr[i] = ‘*’;
}
```

System Stack

<table>
<thead>
<tr>
<th>[EBP]</th>
<th>old EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
</tr>
</tbody>
</table>

ESP, EBP
LEA Visualization

- **In Assembly**
  ```assembly
  create_arr PROC
  push ebp
  mov ebp, esp
  sub esp, 32
  lea esi, [ebp-30]
  mov ecx, 30
  L1:
  mov BYTE PTR [esi], '*'
  inc esi
  loop L1
  add esp, 32
  pop ebp
  ret
  create_arr ENDP
  ```

- **In HLL:**
  ```c
  void create_arr(){
    char arr[30];
    for (int i = 0; i < 30; i++)
      arr[i] = '*';
  }
  ```

**System Stack**

<table>
<thead>
<tr>
<th>EBP</th>
<th>[EBP]</th>
<th>old EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[EBP - 32]</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>[EBP + 4]</td>
<td>return @</td>
<td>...</td>
</tr>
</tbody>
</table>

**Why 32 instead of 30?**
LEA Visualization

- In Assembly

```assembly
create_arr PROC
    push ebp
    mov ebp, esp
    sub esp, 32
    lea esi, [ebp - 30]
    mov ecx, 30
L1:
    mov BYTE PTR [esi], '*'
    inc esi
    loop L1
    add esp, 32
    pop ebp
ret
create_arr ENDP
```

- In HLL:

```c
void create_arr()
{
    char arr[30];
    for (int i = 0; i < 30; i++)
        arr[i] = '*';
}
```

Can we do: 
```
mov esi, OFFSET [ebp-30]
```
**LEA Visualization**

- **In Assembly**

  ```assembly
  create_arr PROC
  push ebp
  mov ebp, esp
  sub esp, 32
  lea esi, [ebp-30]
  mov ecx, 30
  L1:
  mov BYTE PTR [esi], '*'
  inc esi
  loop L1
  add esp, 32
  pop ebp
  ret
  create_arr ENDP
  ```

- **In HLL**

  ```c
  void create_arr()
  {
  char arr[30];
  for (int i = 0; i < 30; i++)
  {
      arr[i] = '*';
  }
  }
  ```

- **System Stack**

  ```
  System Stack
  EBP ESP [EBP - 32]
  .... ....
  [EBP] old EBP
  [EBP + 4] return @
  .... ...
  ```

  Can we do: `mov esi, ebp-30`?
struct Point {
    int xcoord;  // x
    int ycoord;  // y
};

int y = points[i].ycoord;
int *p = &points[i].ycoord;

lea esi, [ebx + 8 * eax + 4] ; addr in esi