

## ELEC3730 Embedded Systems Lecture 11: Mixing C and Assembly

- Register Usage
- Calling Functions
- Passing/Retrieving Parameters
- Stack Management

---

---

---

---

---

---

---

---

### C - Register Usage Conventions

Register(s)	Usage in C functions
EAX	Functions return all pointers and integer values up to 32-bits in this register.
EDX and EAX	Functions return 64-bit values (long long ints) in this register pair. (Note: EDX holds bits 63-32, EAX holds bits 31-0).
EBP	Used to access: (1) The arguments that were passed to a function when it was called, and (2) any automatic variables allocated by the function.
EBX, ESI, EDI, EBP, DS, ES, and SS.	These registers must be preserved by functions written in assembly language. Any of these registers that the function modifies should be pushed on entry to the function and popped on exit.
EAX, ECX, EDX, FS and GS	"Scratch" registers. These registers may be used without preserving their current content.
DS, ES, SS, EBP, and ESP	Used to reference data. If modified by a function, the current contents of these registers must be preserved on entry and restored on <i>return</i> .

---

---

---

---

---

---

---

---

### C Objects (Variables) - Compiled Assembly

C	Intel Assembly
<pre>static int x=1; static int y=2; x=y;</pre>	<pre>_x  DWORD 1 _y  DWORD 2 MOV EAX, DWORD [_y] MOV DWORD [_x], EAX</pre>
<pre>static int a[100]; static int k=5; a[k]=5;</pre>	<pre>_a  DWORD 100 DUP(0) _y  DWORD 5 MOV EAX, [_k] MOV DWORD [_a+4*EAX], 5</pre>
<pre>static int x; static int *p; p=&amp;x; *p=0;</pre>	<pre>_x  DWORD 0 _p  DWORD 0 MOV DWORD [_p], _x MOV EAX, [_p] MOV DWORD [EAX], 0</pre>

---

---

---

---

---

---

---

---

### No Parameters and No Return Value.

- CALL instruction used by caller to invoke the function
  - Pushes the return address onto the stack.
- RET instruction used in function to return to caller.
  - Pops the return address off the stack.

C prototype:	void Disable_Ints(void) ;
Example usage:	Disable_Ints() ;
Generated code:	CALL _Disable_Ints
NASM source code for the function:	<pre> _Disable_Ints:     CLI      ; Disables interrupt system     RET     ; Return from function         </pre>

---

---

---

---

---

---

---

---

---

---

### No Parameters and 8-bit Return Value.

C prototype:	BYTE8 LPT1_Status(void) ;
Example usage:	status = LPT1_Status() ;
Generated code:	CALL _LPT1_Status ; returns status in EAX MOV [ _status],AL
NASM source code for the function:	<pre> _LPT1_Status:     MOV     DX,03BDh    ; Load DX w/hex I/O adr     XOR     EAX,EAX     ; Pre-Zero rest of EAX     IN     AL,DX       ; Get status byte from port.     RET     ; Return from function.         </pre>

---

---

---

---

---

---

---

---

---

---

### Passing Parameters to a C Function

- Parameters are pushed onto stack prior to CALL.
  - gcc pushes parameters in reverse order.
  - 8/16-bit parameters are extended to 32-bits
- Caller removes parameters after function returns.

Function call w/parameters:	Byte2Port(0x3BC, data) ;
Code generated by the compiler:	<pre> PUSH DWORD [_data] ; Push 2nd param MOV  EAX,03BCh    ; Push 1st param PUSH EAX CALL _Byte2Port   ; Call the function. ADD  ESP,8        ; Remove params         </pre>

---

---

---

---

---

---

---

---

---

---



## Retrieving Parameters

```

_Byte2Port:
    MOV    DX,[ESP+4]    ; Copy 1st parameter to DX (the I/O port adrs).
    MOV    AL,[ESP+8]    ; Copy 2nd parameter to AL (discard bits 31-8).
    OUT    DX,AL        ; Write the data to the I/O port.
    RET                    ; Return to caller.
    
```

Better way - EBP is associated with Stack Segment ESS in forming physical addresses - specifically designed for retrieving parameters off the stack.

```

_Byte2Port:
    PUSH   EBP          ; Preserve current contents of BP on stack
    MOV    EBP,ESP      ; Establish a reference point in the stack
    MOV    DX,[EBP+8]   ; Copy 1st parameter to DX (the I/O port address)
    MOV    AL,[EBP+12]  ; Copy 2nd parameter to AL (discard bits 15-8)
    OUT    DX,AL        ; Write the data to the I/O port
    POP    EBP          ; Restore old contents of BP from stack
    RET                    ; Return to caller
    
```

## Temporary Variables

- Use automatic allocation:
  - Temporaries rarely need persistence;
  - Allocate temporaries on the stack;
  - Guarantees that function is reentrant.
- Only available space is *beyond* top of stack.
  - Must be allocated before it can be used (stack pointer must be adjusted and later restored).
- Example - swap two integers:

Function definition	Function invocation
<pre> void Swap(int *p1, int *p2) {     int temp = *p1 ;     *p1 = *p2 ;     *p2 = temp ; }                 </pre>	<pre> int x = 4 ; int y = 7 ; ... Swap(&amp;x, &amp;y) ; ...                 </pre>

## Temporary Variables - Swap Example

```

_Swap:  PUSH   EBP          ; Preserve original EBP contents
        MOV    EBP,ESP      ; Establish stack frame reference in EBP
        SUB    ESP,4        ; Allocate temporary in automatic memory
        . . .
    
```

Address	Content	Description
	***	Stack space currently in use by calling context.
[EBP+12]	p2	Function parameters pushed on the stack by the caller.
[EBP+8]	p1	
[EBP+4]	Return address	Return address pushed by the CALL and popped by the RET.
[EBP]	original EBP	Original EBP preserved by PUSH EBP and restored by POP EBP.
[EBP-4]	temp	Temporary int with automatic memory allocation. ( <i>Top of stack</i> )
	***	Unused stack space (Interrupts push return address here)

```

        MOV    ESP,EBP      ; Release the temporary automatic int
        POP    EBP          ; Restore original EBP
        RET                    ; Return from this function
    
```

### Temporary Variables - Swap Example (Continued)

```
_Swap: PUSH EBP           ; Preserve original EBP contents
      MOV EBP,ESP       ; Establish stack frame reference in EBP
      SUB ESP,4         ; Allocate a temporary in automatic memory
      MOV ECX,[EBP+8]   ; temp = *p1: (1) Get 1st parameter (p1)
      MOV EAX,[ECX]     ; (2) Use it to get *p1 into EAX
      MOV [EBP-4],EAX   ; (3) Then store EAX into temp.
      MOV ECX,[EBP+12] ; *p1 = *p2: (1) Get 2nd parameter (p2)
      MOV EAX,[ECX]     ; (2) Use it to get *p2 into EAX
      MOV ECX,[EBP+8]   ; (3) Get 1st parameter (p1) again
      MOV [ECX],EAX    ; (4) Use it to store EAX into *p1
      MOV EAX,[EBP-4]   ; *p2 = temp: (1) Get the temp into EAX
      MOV ECX,[EBP+12] ; (2) Get 2nd parameter (p2) again
      MOV [ECX],EAX    ; (3) Use it to store EAX into *p2
      MOV ESP,EBP      ; Release the temporary int
      POP EBP          ; Restore original EBP
      RET              ; Return from this function
```

### Swap Example (Continued) - Use of Enter and Leave Instructions

```
_Swap: ; PUSH EBP           ; Preserve original EBP contents
      ; MOV EBP,ESP       ; Establish stack frame reference in EBP
      ; SUB ESP,4         ; Allocate a temporary in automatic memory
      ENTER 4,0          ; Push EBP, ESP->EBP, ESP<-ESP-4
      MOV ECX,[EBP+8]   ; temp = *p1: (1) Get 1st parameter (p1)
      MOV EAX,[ECX]     ; (2) Use it to get *p1 into EAX
      MOV [EBP-4],EAX   ; (3) Then store EAX into temp.
      MOV ECX,[EBP+12] ; *p1 = *p2: (1) Get 2nd parameter (p2)
      MOV EAX,[ECX]     ; (2) Use it to get *p2 into EAX
      MOV ECX,[EBP+8]   ; (3) Get 1st parameter (p1) again
      MOV [ECX],EAX    ; (4) Use it to store EAX into *p1
      MOV EAX,[EBP-4]   ; *p2 = temp: (1) Get the temp into EAX
      MOV ECX,[EBP+12] ; (2) Get 2nd parameter (p2) again
      MOV [ECX],EAX    ; (3) Use it to store EAX into *p2
      LEAVE             ; EBP->ESP, POP Stack into EBP
      ; MOV ESP,EBP      ; Release the temporary int
      ; POP EBP          ; Restore original EBP
      RET              ; Return from this function
```