PROCEDURES

MODULE OUTLINE

1. LINKING TO AN EXTERNAL LIBRARY
2. STACK OPERATIONS
3. DEFINING AND USING PROCEDURES
4. PROGRAM DESIGN AND USING PROCEDURES

7.1. LINKING TO AN EXTERNAL LIBRARY- 1

7.1.1. INTRODUCTION

- **Link Library is a file containing procedures (subroutines) that have been assembled in machine code.**
- **General Syntax of Procedure prototype:**

```
ProcedureName PROTO
```

- **Calling a Procedure:**

```
call ProcedureName
```

7.1. LINKING TO AN EXTERNAL LIBRARY- 2

7.1.1. INTRODUCTION

- **Linker Command Options:**

```
link hello.obj myApi.lib kernel.lib
```

- **Linking 32-bit Programs:**

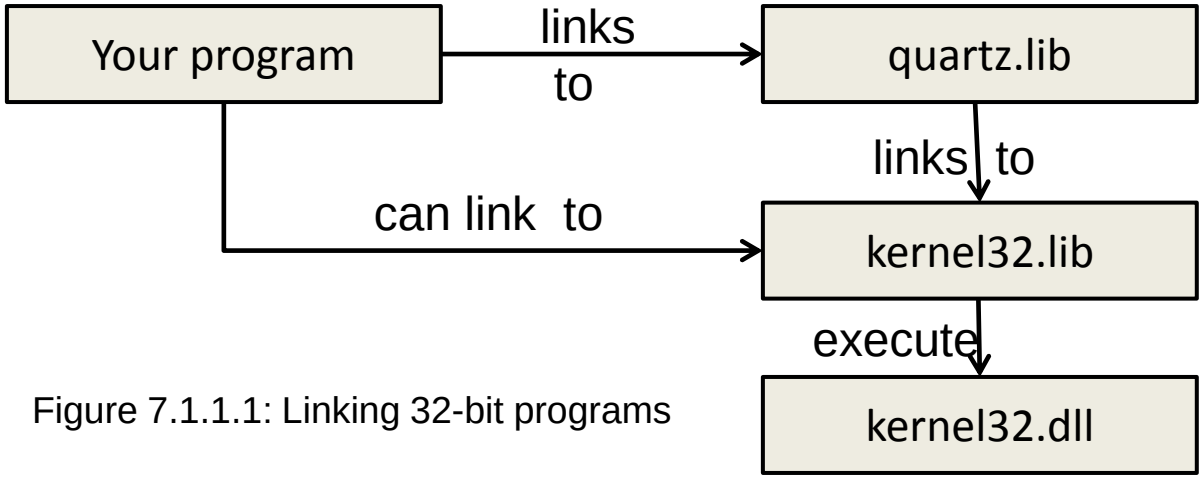


Figure 7.1.1.1: Linking 32-bit programs

7.2. STACK OPERATIONS - 1

7.2.1. INTRODUCTION

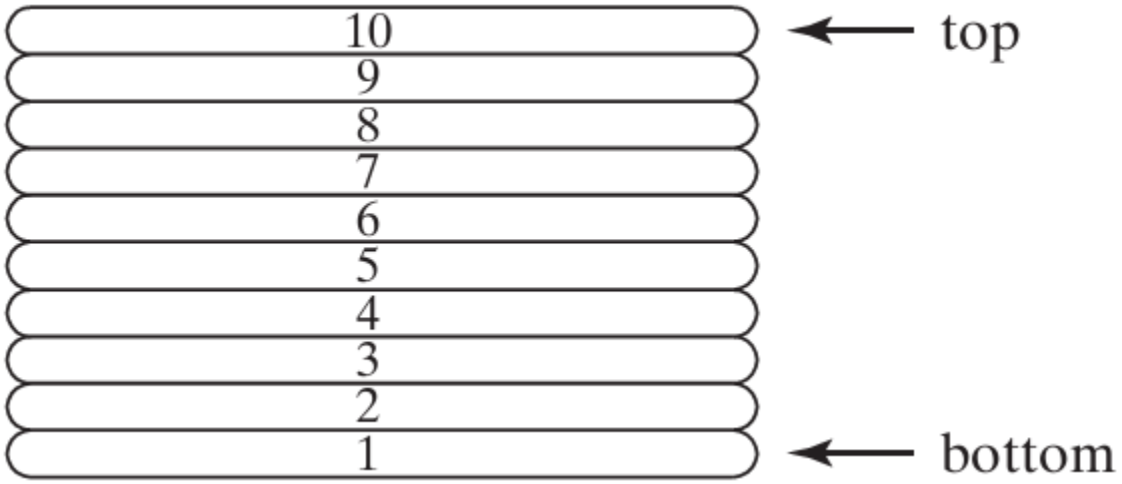


Figure 7.2.1.1: Stack of Plates

7.2. STACK OPERATIONS - 2

7.2.2. RUNTIME STACK

- ***Memory array managed directly by the CPU, using the ESP register***
- ***ESP is modified by instructions: CALL, RET, PUSH and POP.***

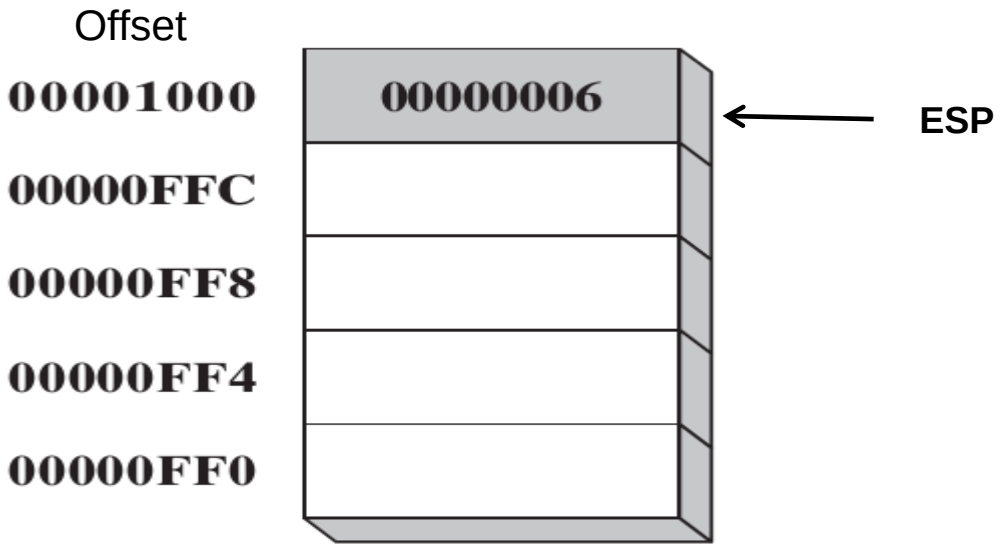


Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 3

7.2.3. PUSH AND POP INSTRUCTIONS

- ***PUSH INSTRUCTION***

```
PUSH reg/mem16
PUSH reg/mem32
PUSH imm32
```

Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 4

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***PUSH INSTRUCTION***

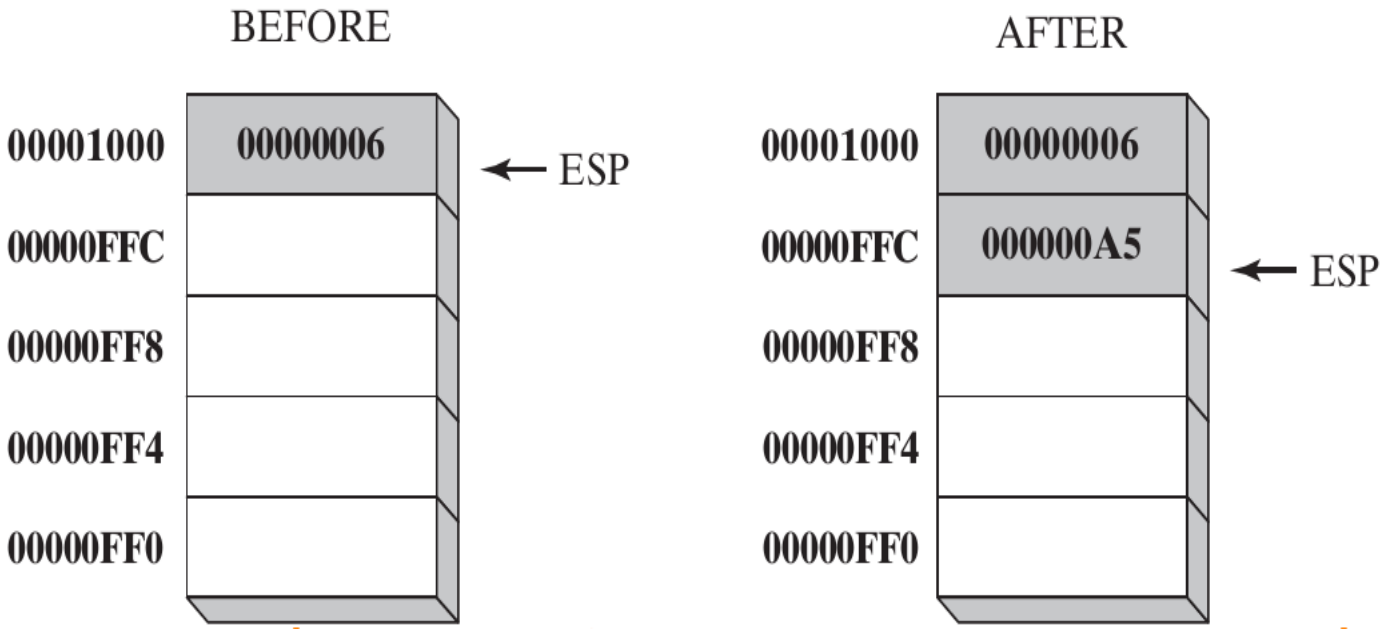


Figure 7.2.3.1: Pushing Integers on the Stack

7.2. STACK OPERATIONS - 5

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***POP INSTRUCTION***

```
POP reg/mem16
POP reg/mem32
```

Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 6

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***POP INSTRUCTION***

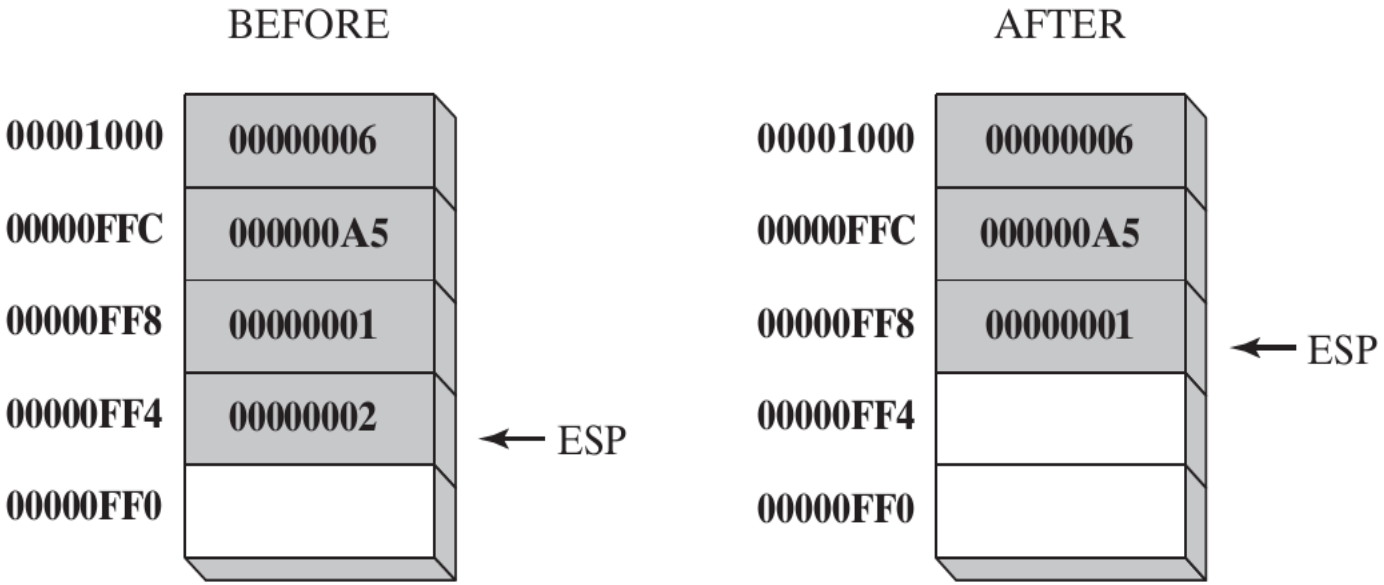


Figure 7.2.3.2: Popping a Value from the Runtime Stack

7.2. STACK OPERATIONS - 7

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***PUSHFD AND POPFD INSTRUCTIONS: 32-BIT EFLAGS***

```
pushfd
popfd

pushfd                          ;save the flags
;
; any sequence of statements here...
;
popfd                           ;restore the flags
```

- ***PUSHF AND POPF INSTRUCTIONS: 16-BIT EFLAGS***

```
pushf
popf
```

Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 8

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***LESS ERROR-PRONE WAY:***

```
.data
saveFlags DWORD ?
.code
pushfd                  ;push flags on stack
pop saveFlags           ;copy into a variable
```

- ***RESTORE THE FLAGS FROM THE SAME VARIABLE***

```
push saveFlags          ;push saved flag values
popfd                   ;copy into the flags
```

Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 9

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***PUSHAD, PUSHA, POPAD, and POPA:***

```
MySub PROC
     pushad          ;save general-purpose registers
     .
     .
     mov eax,...
     mov edx,...
     mov ecx,...
     .
     .
     popad           ;restore general-purpose registers
     ret
MySub
```

Figure 7.2.2.1: A Stack Containing a Single Value

7.2. STACK OPERATIONS - 9

7.2.3. PUSH AND POP INSTRUCTIONS - contd

```
ReadValue PROC
        pushad      ;save general-purpose registers
        .
        .
        mov eax,return_value
        .
        .
        popad       ;overwrites EAX!
        ret
ReadValue ENDP
```

Figure 7.2.2.1: A Stack Containing a Single value

7.3. DEFINING AND USING PROCEDURES - 1

7.3.1. INTRODUCTION

- **Recall:**
  `[label:] mnemonic [operands][ ; comment ]`
- **Generally:**

```
mnemonic
mnemonic [destination]
mnemonic [destination],[source]
mnemonic [destination],[source-1],[source-2]
```

7.3. DEFINING AND USING PROCEDURES - 2

7.3.2. PROC DIRECTIVE

- **Defining a Procedure:**

```
main PROC
.
.
main ENDP
```

- **Sample Procedure:**

```
sample PROC
.
.
ret             ;forces the CPU to return to the caller
sample ENDP
```

7.3. DEFINING AND USING PROCEDURES - 3

7.3.2. PROC DIRECTIVE

- **Labels in Procedures:**

```
jmp Destination
.
.
Destination:
```

- **Example: Sum of Three Integers:**

```
SumOf PROC
    add eax,ebx
    add eax,ecx
    ret
SumOf ENDP
```

7.3. DEFINING AND USING PROCEDURES - 4

7.3.2. PROC DIRECTIVE

- **Documentation Procedures:**

```
;-----------------------------------------------------
Sumof PROC
;
;Calculates and returns the sum of three 32-bit integers.
;Receives: EAX, EBX, ECX, the three integers. May be
;signed or unsigned.
;Returns: EAX = sum
;-----------------------------------------------------
    add eax,ebx
    add eax,ecx
    ret
SumOf ENDP
```

7.3. DEFINING AND USING PROCEDURES - 5

7.3.2. CALL and RET Instructions

- **Call and Return Example**

```
            main PROC
00000020       call MySub
00000025       mov eax,ebx
            MySub PROC
00000040       mov eax,edx
               .
               .
               ret
            MySub ENDP
```

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Call and Return Example**

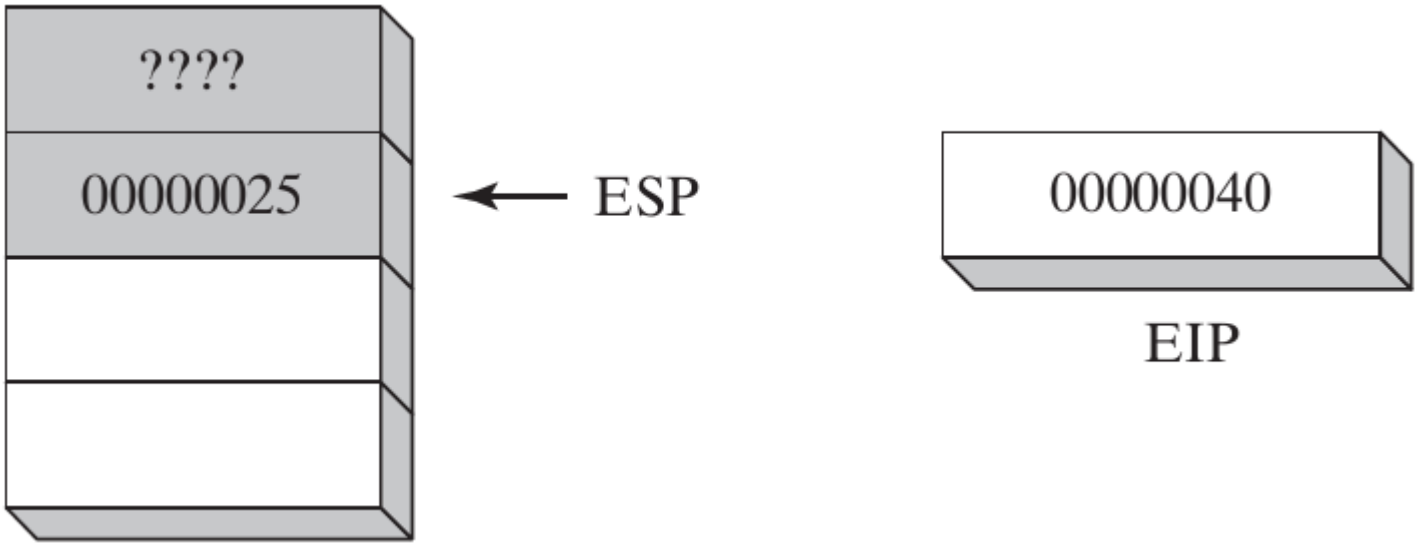


Figure 7.3.2.1: Executing a CALL Instruction

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Call and Return Example**

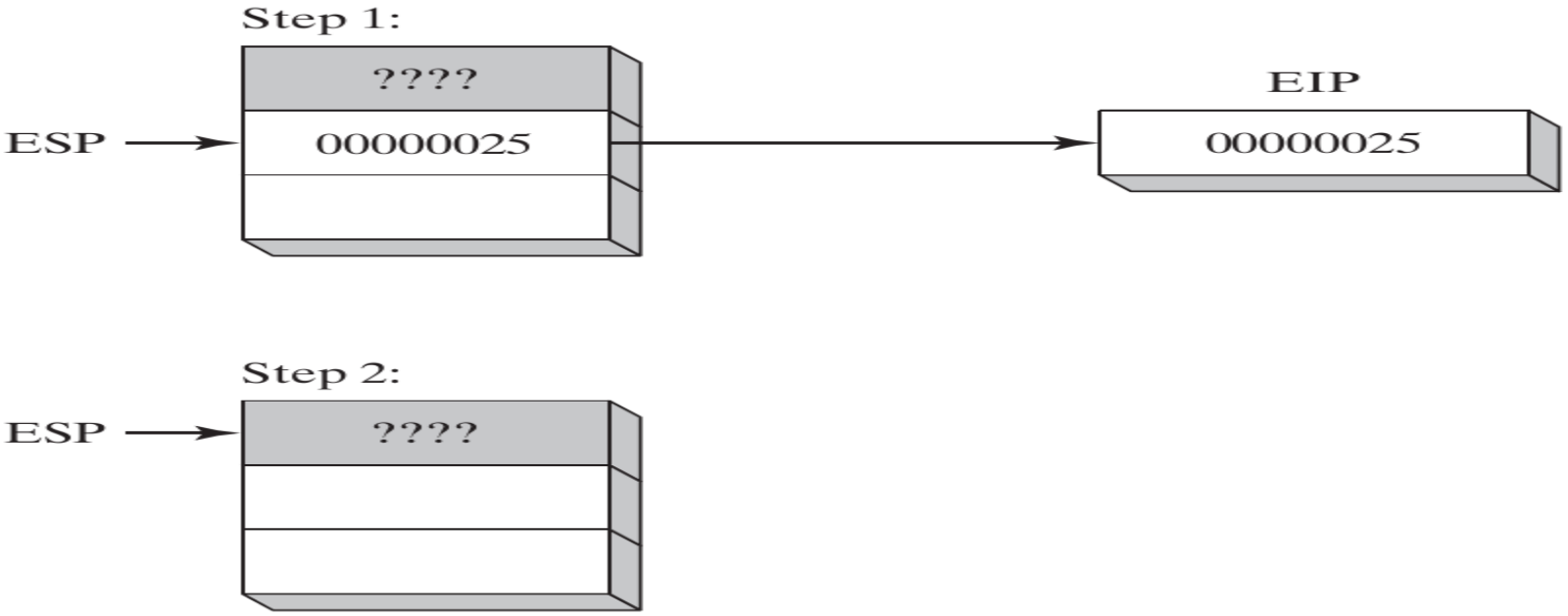


Figure 7.3.2.2: Executing the RET Instruction

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Nested Procedure Calls**

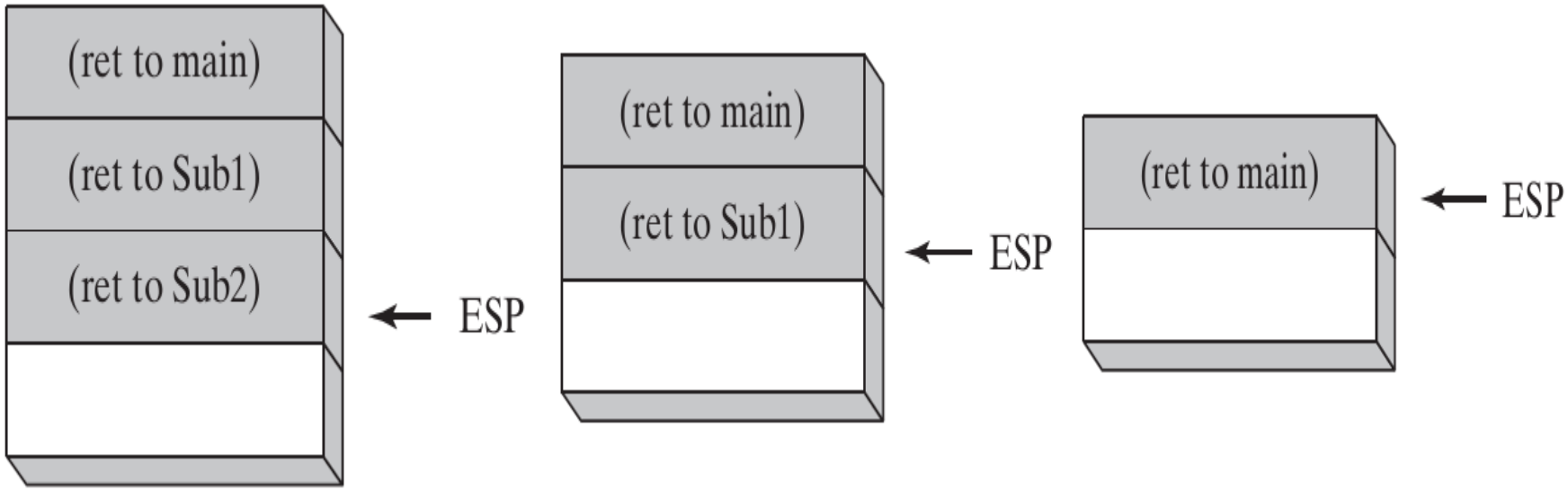


Figure 7.3.2.3: Example of usage of ESP in Nested Procedure Calls

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Passing Register Arguments to Procedures**

```
.data
theSum DWORD ?
.code
main PROC
   mov eax,10000h ; argument
   mov ebx,20000h ; argument
   mov ecx,30000h ; argument
   call Sumof  ; EAX = (EAX + EBX + ECX)
   mov theSum,eax ; save the sum
```

7.3. DEFINING AND USING PROCEDURES - 6

7.3.3. Example: Summing an Integer Array

- **Passing Register Arguments to Procedures**

```
;-----------------------------------------------------
ArraySum PROC
;
;Calculates the sum of an array of 32-bit integers.
;Receives: ESI = the array offset
;ECX = number of elements in the array
;Returns: EAX = sum of the array elements
;-----------------------------------------------------
    push esi                                ;save ESI, ECX
    push ecx
    mov eax,0                               ;set the sum to zero
L1: add eax,[esi]                           ;add each integer to sum
    add esi,TYPE DWORD                      ;point to next integer
    loop L1                                 ;repeat for array size
    pop ecx                                 ;restore ECX, ESI
   pop esi
   ret ;sum is in EAX
ArraySum ENDP
```

7.3. DEFINING AND USING PROCEDURES - 6

7.3.3. Example: Summing an Integer Array

- **Calling ArraySum**

```
.data
array DWORD 10000h,20000h,30000h,40000h,50000h
theSum DWORD ?
.code
main PROC
   mov esi,OFFSET array ; ESI points to array
   mov ecx,LENGTHOF array ; ECX = array count
   call ArraySum ; calculate the sum
   mov theSum,eax ; returned in EAX
```

7.3. DEFINING AND USING PROCEDURES - 7

7.3.4. FLOWCHARTS

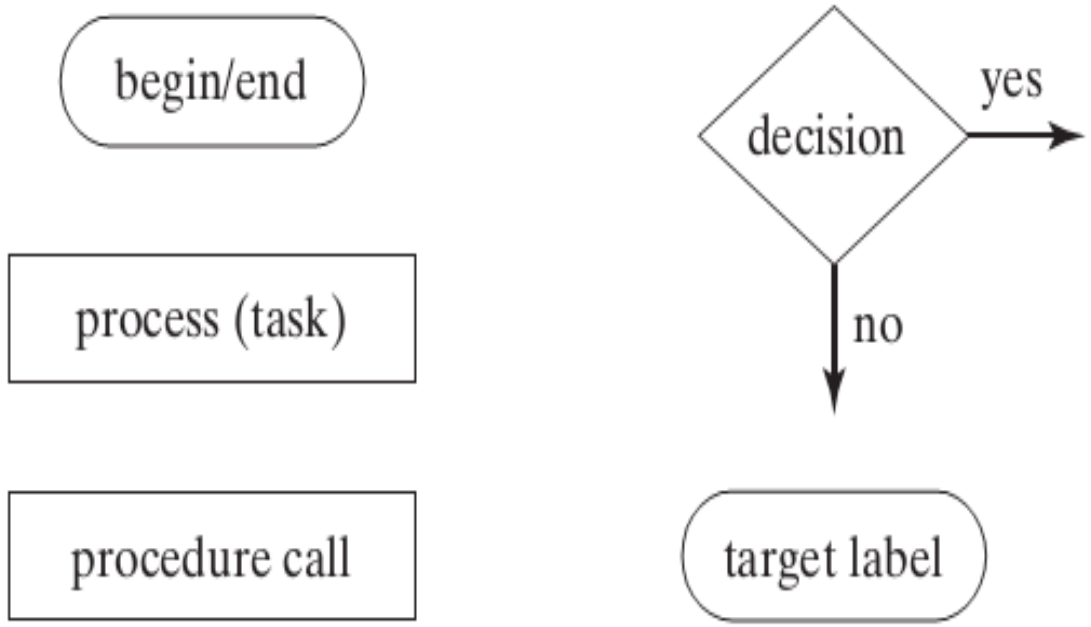
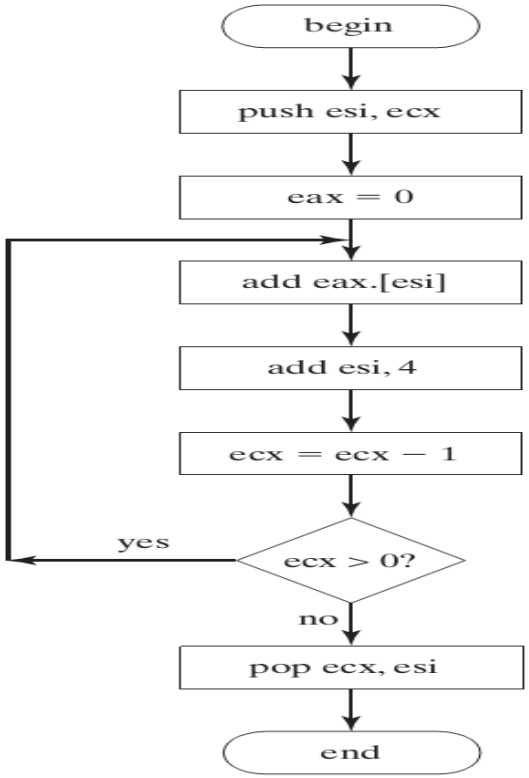


Figure 7.3.4.1: Basic Flowchart Shapes

7.3. DEFINING AND USING PROCEDURES - 8

7.3.4. FLOWCHARTS




```
    push esi
    push ecx
    mov eax,0
AS1:
    add eax,[esi]
    add esi,4
    loop AS1
    pop ecx
    pop esi
```

Figure 7.3.4.2: Flowchart for the ArraySumProcedure

7.3. DEFINING AND USING PROCEDURES - 9

7.3.5. SAVING AND RESTORING REGISTERS

- **USES Operator: coupled with PROC directive, lets you list the names of all registers modified within a procedure.**

```
ArraySum PROC USES esi ecx
    mov eax,0              ;set the sum to zero
L1:
    add eax,[esi]          ;add each integer to sum
    add esi,TYPE DWORD     ;point to next integer
    loop L1                ;repeat for array size
    ret                    ;sum is in EAX
ArraySum ENDP
```

7.3. DEFINING AND USING PROCEDURES - 10

7.3.5. SAVING AND RESTORING REGISTERS

- **The corresponding code generated by the assembler shows the effect of USES.**

```
ArraySum PROC
    push esi
    push ecx
    mov eax,0              ;set the sum to zero
L1:
    add eax,[esi]          ;add each integer to sum
    add esi,TYPE DWORD     ;point to next integer
    loop L1                ;repeat for array size
    pop ecx
    pop esi
    ret
ArraySum ENDP
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 1

7.4.1. INTRODUCTION

- **The Principle of "Functional Decomposition" or "Top-Down" design is based on:**
  - **Large problem should be divided into small tasks.**
  - **Procedures, tested separately – Program maintenance easier**
  - **This design exposes procedures relationships.**
  - **Design before coding individual procedures**

7.4. PROGRAM DESIGN AND USING PROCEDURES - 2

7.4.2. INTEGER SUMMATION PROGRAM DESIGN

- **Problem: Write a program that prompts the user for three 32-bit integers, stores them in an array, calculates the sum of the array, and displays the sum on the screen.**
- **Design ideas in pseudocode**

```
Integer Summation Program
    Prompt user for three integers
    Calculate the sum of the array
    Display the sum
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 3

7.4.2. INTEGER SUMMATION PROGRAM DESIGN

- **Assign a procedure to each task:**

```
Main
    PromptForIntegers
    ArraySum
    DisplaySum
```

- **Further refinement:**

```
Main
    Clrscr                          ;clear screen
    PromptForIntegers
        WriteString                 ;display string
        ReadInt                     ;input integer
    ArraySum                        ;sum the integers
    DisplaySum
        WriteString                 ;display string
        WriteInt                    ;display integer
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 4

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
TITLE Integer Summation Program (Sum2.asm)
; This program prompts the user for three integers,
; stores them in an array, calculates the sum of the
; array, and displays the sum.
INCLUDE Irvine32.inc
INTEGER_COUNT = 3
.data
str1 BYTE "Enter a signed integer: ",0
str2 BYTE "The sum of the integers is: ",0
array DWORD INTEGER_COUNT DUP(?)
.code
main PROC
     call Clrscr
     mov esi,OFFSET array
     mov ecx,INTEGER_COUNT
     call ArraySum
     call DisplaySum
     exit
main ENDP
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 5

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------------
PromptForIntegers PROC USES ecx edx esi
;
; Prompts the user for an arbitrary number of integers
; and inserts the integers into an array.
; Receives: ESI points to the array, ECX = array size
; Returns: nothing
;-----------------------------------------------------
    mov edx,OFFSET str1           ;"Enter a signed integer"
L1: call WriteString              ;display string
    call ReadInt                  ;read integer into EAX
    call Crlf                     ;go to next output line
    mov [esi],eax                 ;store in array
    add esi,TYPE DWORD            ;next integer
    loop L1
    ret
PromptForIntegers ENDP
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 6

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------
ArraySum PROC USES esi ecx
;
; Calculates the sum of an array of 32-bit integers.
; Receives: ESI points to the array, ECX = number
; of array elements
; Returns: EAX = sum of the array elements
;-----------------------------------------------
   mov eax,0                    ;set the sum to zero
L1: add eax,[esi]               ;add each integer to sum
   add esi,TYPE DWORD           ;point to next integer
   loop L1                      ;repeat for array size
   ret                          ;sum is in EAX
ArraySum ENDP
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 7

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------------
DisplaySum PROC USES edx
;
; Displays the sum on the screen
; Receives: EAX = the sum
; Returns: nothing
;-----------------------------------------------------
    mov edx,OFFSET str2          ;"The sum of the..."
    call WriteString
    call WriteInt                ;display EAX
    call Crlf
    ret
DisplaySum ENDP
END main
```