PROCEDURES IN ASSEMBLY LANGUAGE

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:**

```
global procedureName
```

- **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:**

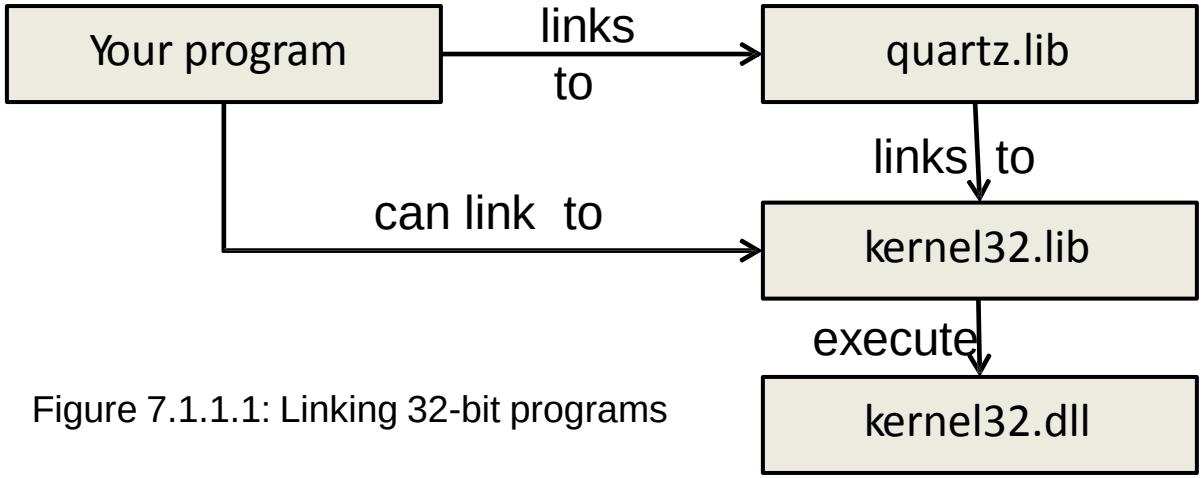


Figure 7.1.1.1: Linking 32-bit programs

7.2. STACK OPERATIONS - 1

7.2.1. INTRODUCTION

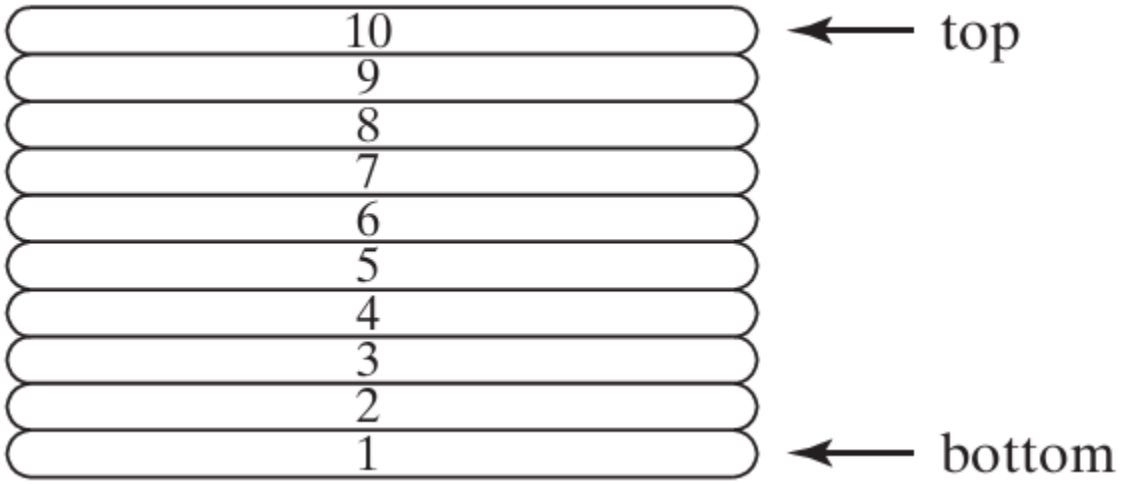
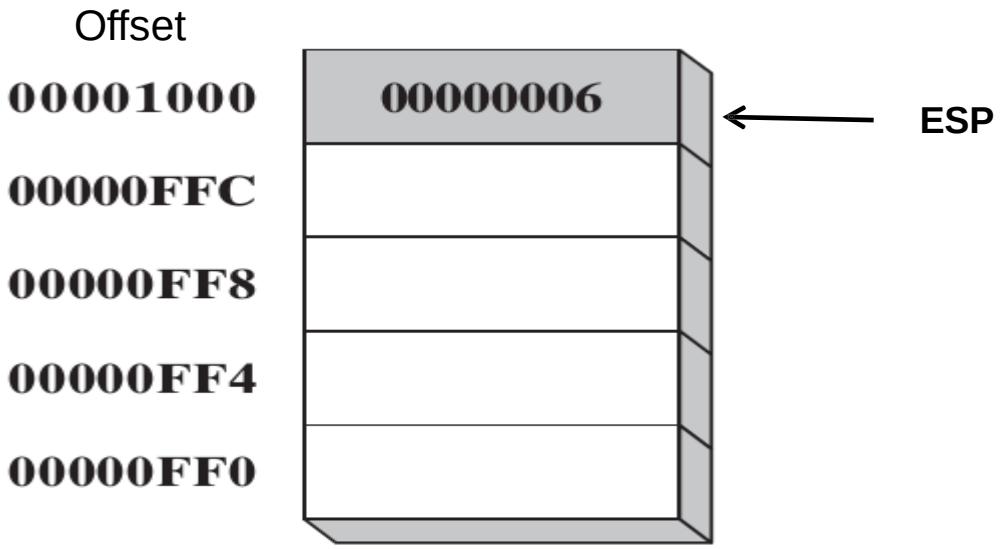


Figure 7.2.1.1: Stack of Plates

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.***



7.2. STACK OPERATIONS - 3

7.2.3. PUSH AND POP INSTRUCTIONS

- ***PUSH INSTRUCTION***

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

7.2. STACK OPERATIONS - 4

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***PUSH INSTRUCTION***

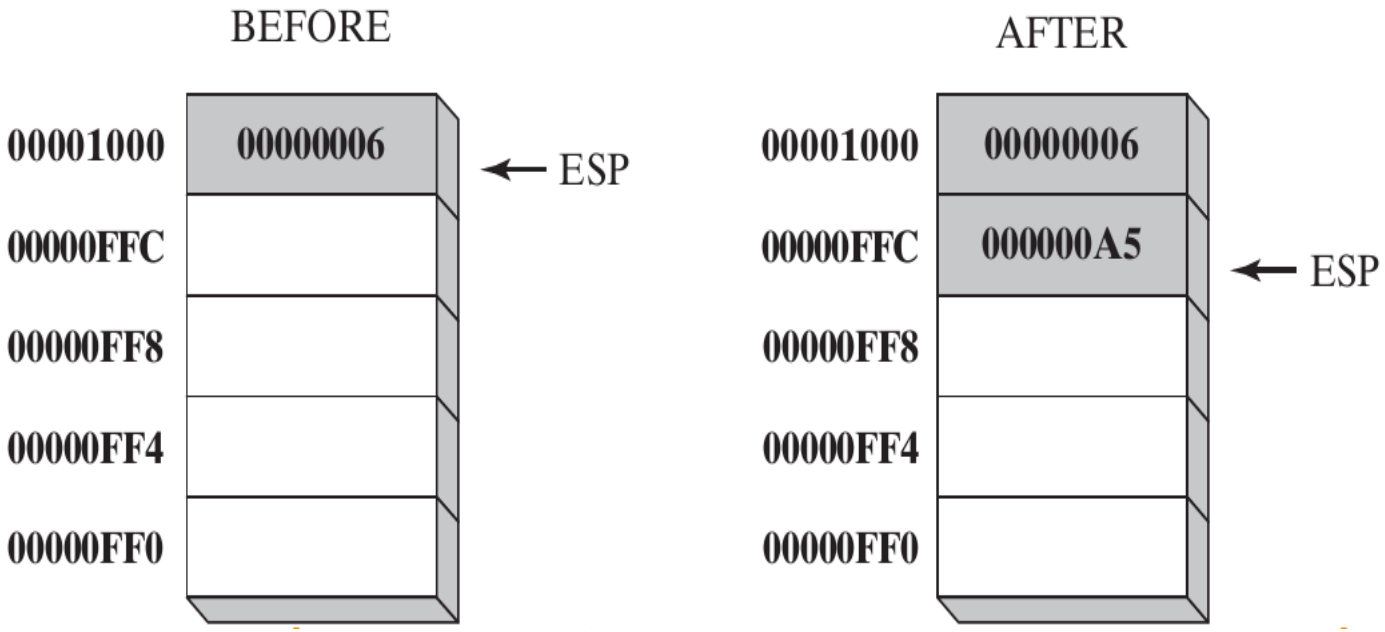


Figure 7.2.3.1: Pushing Integers on the Stack

7.2. STACK OPERATIONS - 5

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***POP INSTRUCTION***

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

7.2. STACK OPERATIONS - 6

7.2.3. PUSH AND POP INSTRUCTIONS - contd

- ***POP INSTRUCTION***

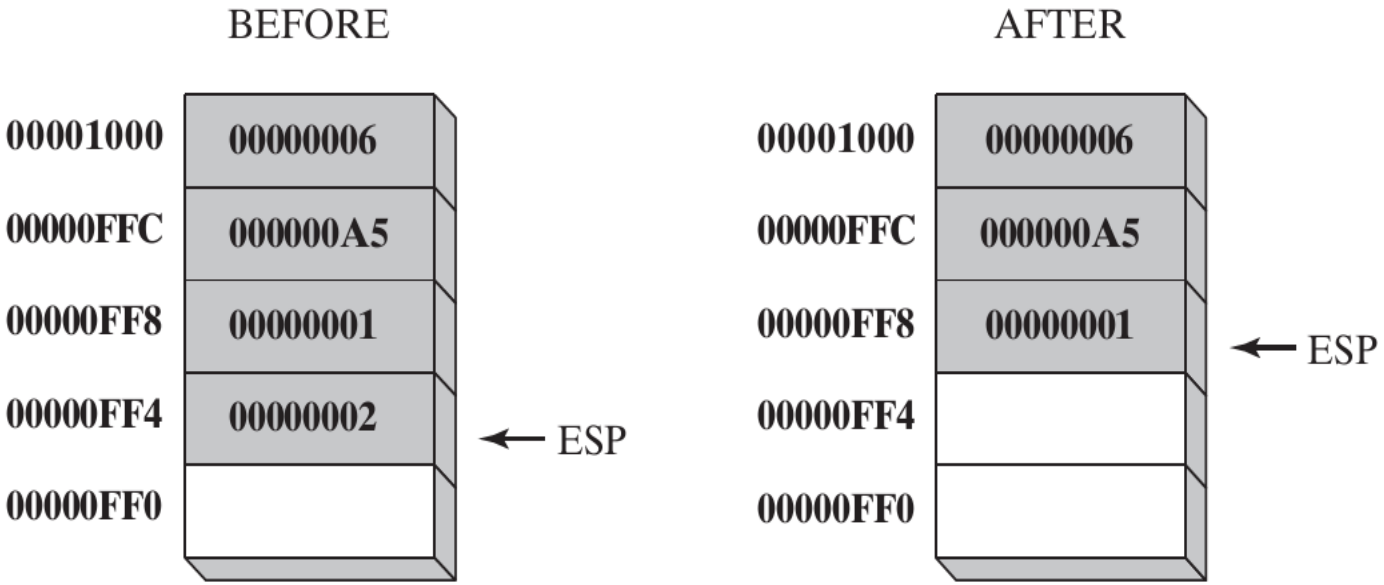


Figure 7.2.3.2: Popping a Value from the Runtime Stack

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
```

7.2. STACK OPERATIONS - 8

7.2.3. PUSH AND POP INSTRUCTIONS - contd

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

```
.bss
saveFlags DW
.text
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
```

7.2. STACK OPERATIONS - 9

7.2.3. PUSH AND POP INSTRUCTIONS - contd

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

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

7.2. STACK OPERATIONS - 9

7.2.3. PUSH AND POP INSTRUCTIONS - contd

```
readValue:
        pushad      ;save general-purpose registers
        .
        .
        mov eax,return_value
        .
        .
        popad       ;overwrites EAX!
        ret
```

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:
   .
   .
   ret
```

- **Sample Procedure:**

```
sampleProC
   .
   .
   ret          ;forces the CPU to return to the caller
```

7.3. DEFINING AND USING PROCEDURES - 3

7.3.2. PROC DIRECTIVE

- **Labels in Procedures:**

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

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

```
sumOf:
    add     eax,ebx     add
    eax,ecx ret
    ret
```

7.3. DEFINING AND USING PROCEDURES - 4

7.3.2. PROC DIRECTIVE

- **Documentation Procedures:**

```
;-----------------------------------------------------
sumof:
;
;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
```

7.3. DEFINING AND USING PROCEDURES - 5

7.3.2. CALL and RET Instructions

- **Call and Return Example**

```
            main:
00000020        call mySub
00000025        mov eax,ebx
                …
                ret
00000040    mySub:
               mov eax, edx
               .
               .
               ret
```

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Call and Return Example**

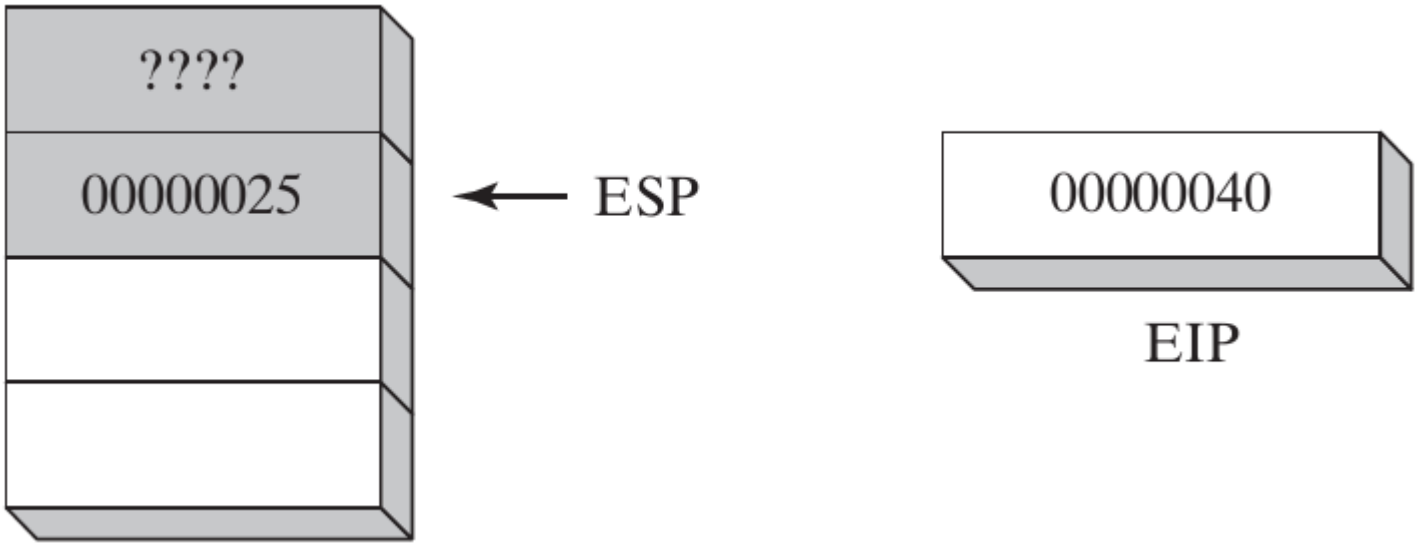


Figure 7.3.2.1: Executing a CALL Instruction

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Call and Return Example**

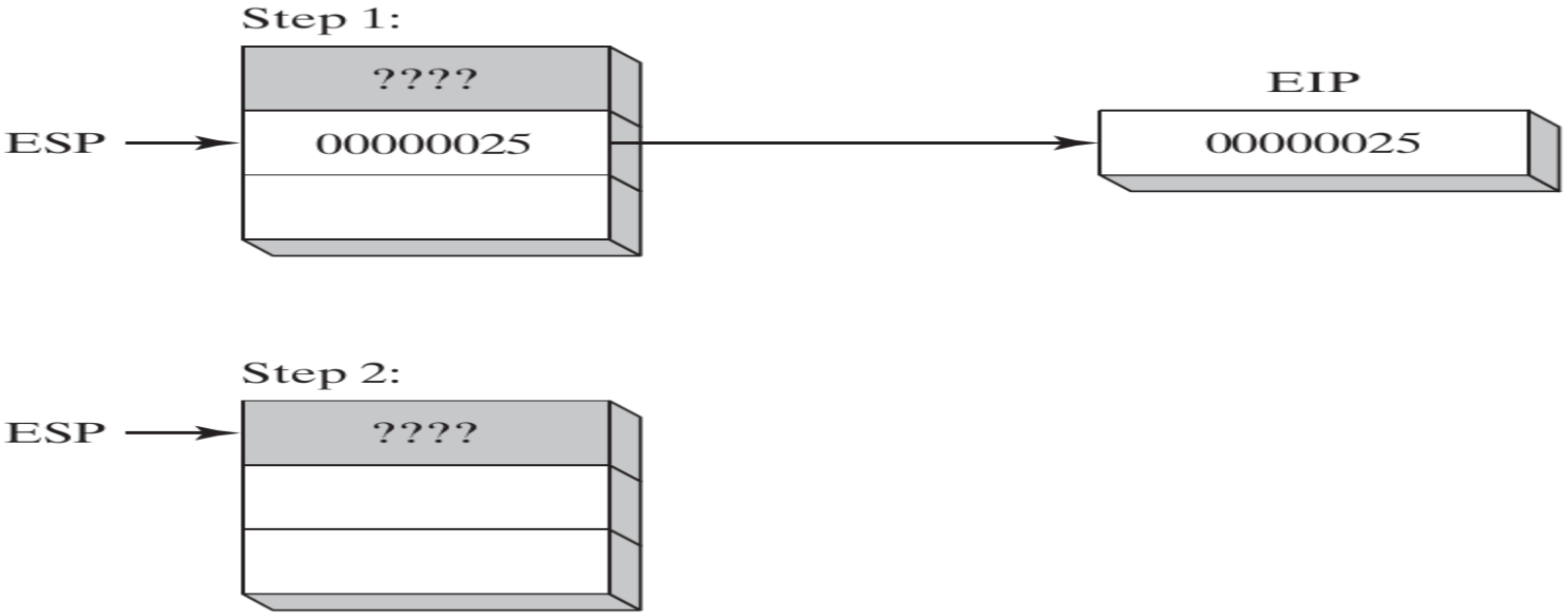


Figure 7.3.2.2: Executing the RET Instruction

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Nested Procedure Calls**

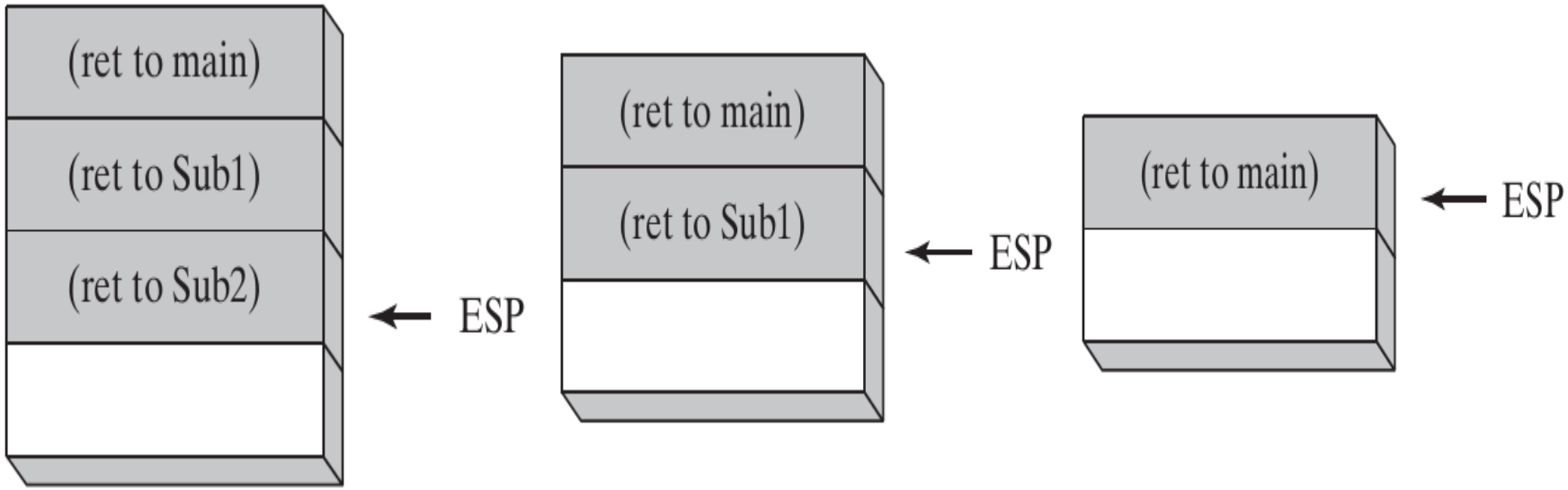


Figure 7.3.2.3: Example of usage of ESP in Nested Procedure Calls

7.3. DEFINING AND USING PROCEDURES - 6

7.3.2. CALL and RET Instructions

- **Passing Register Arguments to Procedures**

```
.bss
theSum: DW
.text
main:
   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:
;
;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,DW                              ;point to next integer
    loop L1                                 ;repeat for array size
    pop ecx                                 ;restore ECX, ESI
    pop esi                                 ;sum is in eax
   ret
```

7.3. DEFINING AND USING PROCEDURES - 6

7.3.3. Example: Summing an Integer Array

- **Calling ArraySum**

```
.data
array dw 10000h,20000h,30000h,40000h,50000h
arrLen equ $-array
.bss
theSum dw
.text
main:
    mov esi,OFFSET array ; ESI points to array
    mov ecx,arrLen         ;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

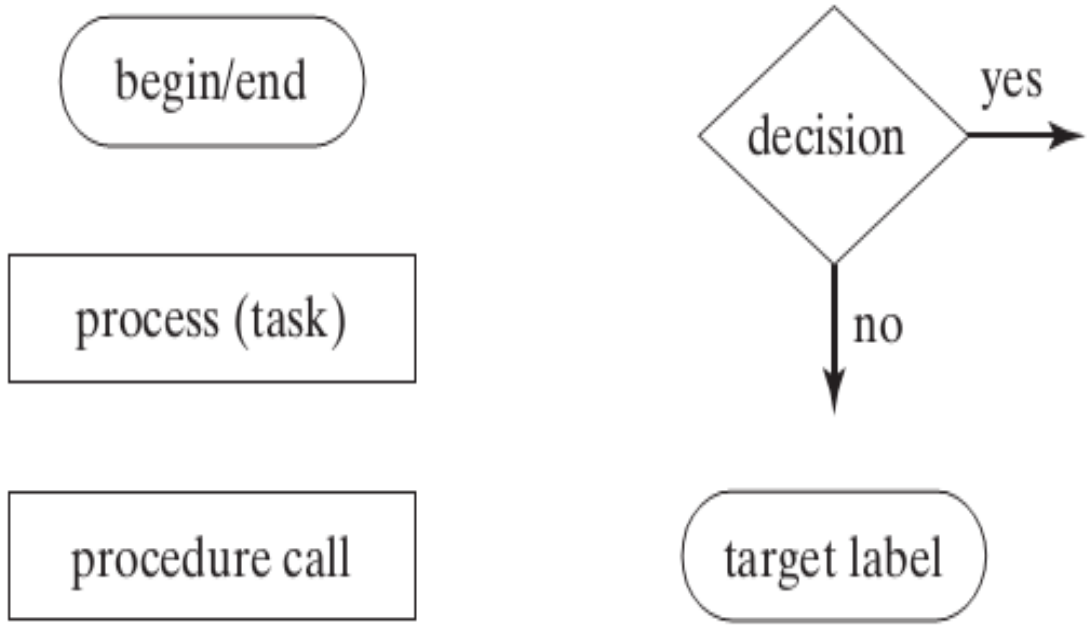
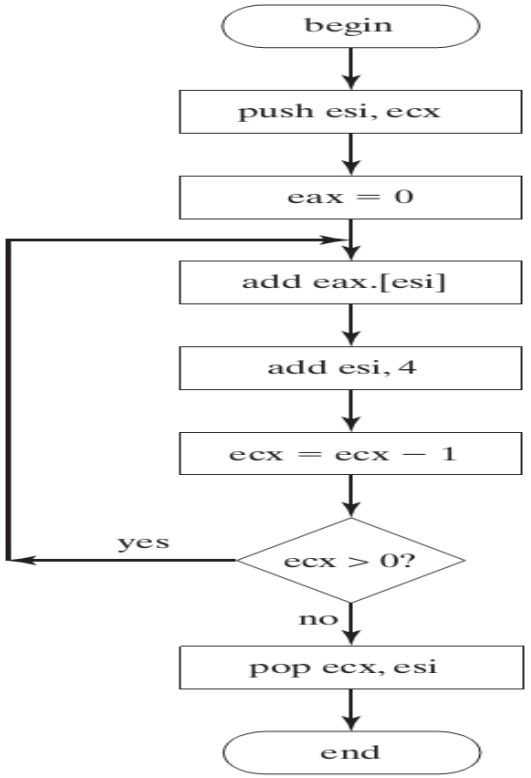


Figure 7.3.4.1: Basic Flowchart Shapes

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:
    mov eax,0              ;set the sum to zero
L1:
    add eax,[esi]          ;add each integer to sum
    add esi,dw             ;point to next integer
    loop L1                ;repeat for array size
    ret                    ;sum is in EAX
```

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

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 “Along32.inc"
INTEGER_COUNT EQU 3
.data
str1 db "Enter a signed integer: ",0
str2 db "The sum of the integers is: ",0
.bss
array TIMES DW INTEGER_COUNT
.text
main:
     call clrscr
     mov esi, array
     mov ecx,INTEGER_COUNT
     call arraySum
     call displaySum
    ret
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 5

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------------
promptForIntegers: ;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,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,DW                      ;next integer
    loop L1
    ret
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 6

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------
arraySum: ;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,DW                   ;point to next integer
   loop L1                      ;repeat for array size
   ret                          ;sum is in EAX
```

7.4. PROGRAM DESIGN AND USING PROCEDURES - 7

7.4.3. INTEGER SUMMATION IMPLEMENTATION

```
;-----------------------------------------------------
displaySum:  ;edx
;
; Displays the sum on the screen
; Receives: EAX = the sum
; Returns: nothing
;-----------------------------------------------------
    mov edx,str2                    ;"The sum of the..."
    call writeString
    call writeInt                   ;display EAX
    call crlf
    ret
```