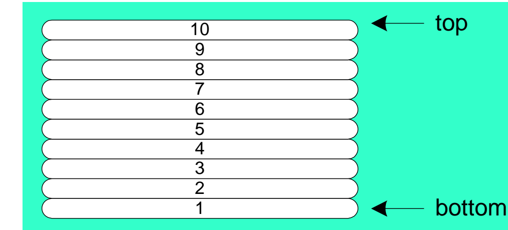
5장 프로시저(2)

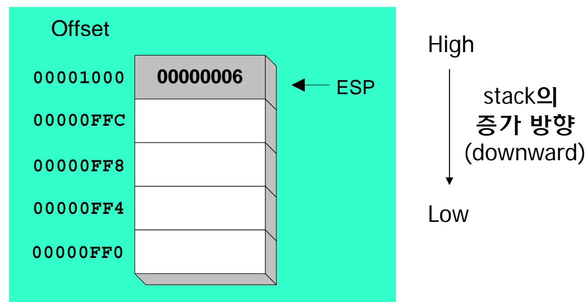
5.4 스택 연산

- 스택(Stack)
  - LIFO(Last In First Out) 구조의 자료구조
  - CPU에서 Runtime Stack을 관리하는 instruction이 제공됨
  - Runtime Stack은 procedure call, return을 하는 데 사용됨



실행시간 스택(Runtime Stack)

- Runtime Stack은 SS와 ESP register에 의해서 관리됨 (SS:ESP)
  - SS (stack segment) : stack segment descriptor register
  - ESP (stack pointer) : stack의 top 주소(offset)를 보관
    (가장 최근에 stack에 저장된 data의 위치)
  - real mode에서는 ESP 대신에 SP를 사용



High

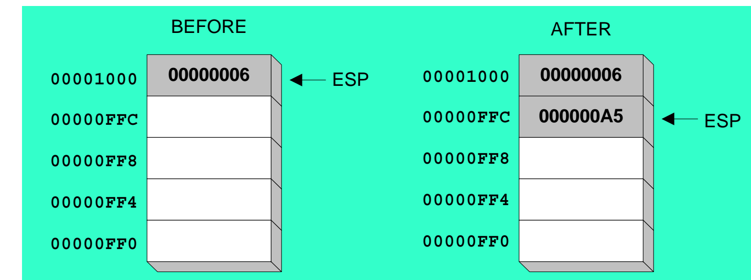
stack의
증가 방향
(downward)

Low

PUSH Operation

- 32-bit push operation
  - ESP ← ESP – 4
  - M[SS:ESP] ← 32-bit value
- 16-bit push operation
  - ESP ← ESP – 2
  - M[SS:ESP] ← 16-bit value

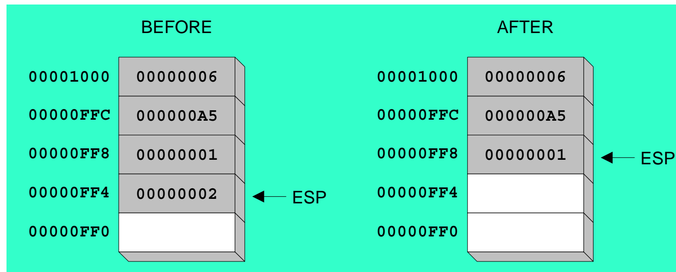
(pre-decrement addressing)



POP Operation

- 32-bit pop operation
  - r/m32 ← M[SS:ESP]
  - ESP ← ESP + 4
- 16-bit pop operation
  - r/m16 ← M[SS:ESP]
  - ESP ← ESP + 2

(post-increment addressing)



PUSH and POP 명령어

- PUSH
  - PUSH *r/m16*
  - PUSH *r/m32*
  - PUSH *imm32* (real mode에서는 PUSH *imm16*)
- POP
  - POP *r/m16*
  - POP *r/m32*

PUSH와 POP 사용하기

- push 순서의 반대 순서로 pop을 수행

```
push esi                          ; push registers
push ecx
push ebx

mov  esi,OFFSET dwordVal          ; display some memory
mov  ecx,LENGTHOF dwordVal
mov  ebx,TYPE dwordVal
call DumpMem

pop ebx                           ; restore registers
pop ecx
pop esi
```


예제: Nested Loop

- 안쪽 루프에 들어가기 전에 바깥 루프용 ECX값을 스택에 저장

```
   mov ecx,100          ; outer loop count=100
L1:                     ; begin the outer loop
   push ecx             ; save outer loop count
   mov ecx,20           ; inner loop count=20
L2:                     ; begin the inner loop
   ;
   ;
   loop L2              ; repeat the inner loop
   pop ecx              ; restore outer loop count
   loop L1              ; repeat the outer loop
```

PUSH, POP Flags

- PUSHFD
  - push EFLAGS on the stack
  - 동작: ESP ← ESP – 4;   M[SS:ESP] ← EFLAGS
- POPFD
  - pop EFLAGS off the stack
  - 동작: EFLAGS ← M[SS:ESP];   ESP ← ESP + 4
- PUSHF, POPF
  - 16-bit real mode에서 사용하며 FLAGS를 push, pop

PUSH, POP Flags (계속)

- Example

```
    cmp [esi],eax
    pushfd              ; save the flag on stack
    . . .
    popfd               ; restore the flag from stack
    loopne
```

```
    cmp [esi],eax
    pushfd              }──→ saveFlags ← EFLAGS
    pop saveFlags
    . . .                          Flag와 메모리 간의 이동
    push saveFlags      }──→ EFLAGS ← saveFlags
    popfd
    loopne
```


PUSH, POP All Registers

- PUSHAD
  - 모든 32비트 범용 레지스터 값들을 스택에 push
  - push 순서: EAX, ECX, EDX, EBX, ESP(원래값), EBP, ESI, EDI
- POPAD
  - 스택에 저장된 32비트 범용 레지스터 값들을 해당 레지스터로 pop
  - pop 순서: PUSHAD의 반대 순서
- PUSHA, POPA
  - 16-bit real mode에서 사용하여 16-bit 범용 레지스터를 push, pop함

```
    pushad              ; save GP registers
    . . .
    mov eax,…
    mov ebx,…
    . . .
    popad               ; restore GP registers
```


예제: 문자열 역순 배치

- 문자열 역순 배치
  - 문자열의 문자들을 모두 stack에 push한 다음에 pop하여 원래의 문자열에 저장함

```
.data
aName byte "Yonsei University",0
nameSize = $ - aName - 1
.code
    mov ecx,nameSize
    mov esi,0
L1: movzx eax, aName[esi]   ; zero extension
    push eax
    inc esi
    loop L1                              (continue)
```

```
    mov ecx, nameSize
    mov esi, 0
L2: pop eax
    mov aName[esi], al
    inc esi
    loop L2
```

- Q: 문자를 EAX에 넣은 다음에 push한 이유는?
  - push 명령어는 16-bit 또는 32-bit 값 만 push한다.

5.5 Procedure 정의와 사용

- PROC 디렉티브
  - procedure를 정의하는 데 사용
  - C/C++의 function의 assembly equivalent
  - 형식

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

```
sample PROC
    .
    .
    ret
sample ENDP
```


프로시저와 label

- 기본적으로 label은 같은 프로시저 내의 점프에 사용가능

```
procA proc
        jmp label
        …
label:  …
        …
procA endp
```

- 전역 label은 ::으로 표시하며 다른 프로시저로의 점프에 사용가능

```
procA  proc          procB proc
  …
  Jge label          label:: …
  …
procA endp           procB endp
```

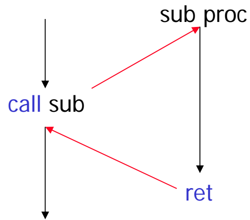

프로시저 문서화

- 프로그램을 이해하기 쉽도록 문서화

```
;---------------------------------------------------------
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, and the status flags (Carry,
;         Overflow, etc.) are changed.
; Requires: nothing
;---------------------------------------------------------
   add eax,ebx
   add eax,ecx
   ret
SumOf ENDP
```

CALL과 RET 명령어

- CALL label 또는 CALL r/m
  - call procedure
  - 동작: push stack ← EIP (next instruction의 주소);
    EIP ← label의 주소 또는 r/m
- RET
  - procedure를 call한 곳으로 return
  - 동작: EIP ← pop stack

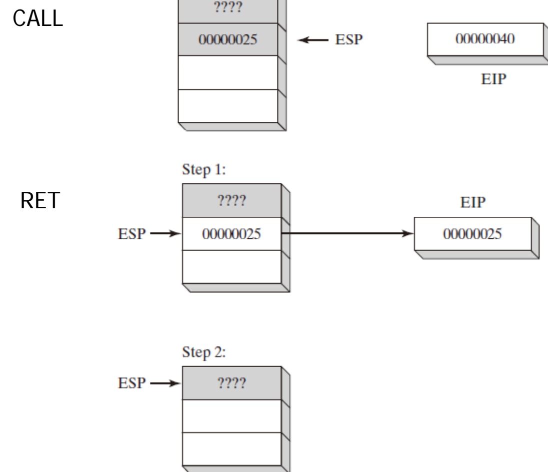


CALL, RET 예제 (1 of 2)

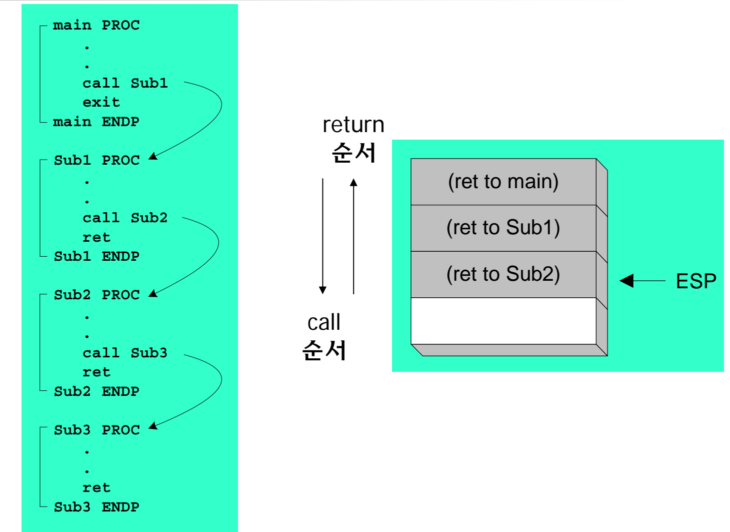
```
         main PROC
00000020   call MySub
00000025   mov eax,ebx
         .
         .
         main ENDP

         MySub PROC
00000040   mov eax,edx
         .
         .
           ret
         MySub ENDP
```

offset

CALL, RET 예제 (2 of 2)



중첩된 프로시저 호출



프로시저 인수 전달

- 특정 변수를 사용하는 것은 바람직하지 않음
- register 또는 stack을 사용하여 인수를 전달함

```
.data
theSum DWORD ?
.code
main PROC
    mov eax, 1000h    }
    mov ebx, 2000h    }  parameter 설정 (EAX, EBX, ECX)
    mov ecx, 3000h    }
    call SumOf           procedure 호출
    mov theSum, eax      결과 저장
SumOf ENDP
```


예제: 배열 합을 구하는 프로시저

- ArraySum procedure 정의

```
ArraySum PROC
; Receives: ESI = address of doubleword array
;           ECX = number of array elements.
; Returns:  EAX = sum
;-----------------------------------------------------
    mov eax,0              ; set the sum to zero
L1: add eax,[esi]          ; add each integer to sum
    add esi,4              ; point to next integer
    loop L1                ; repeat for array size
    ret
ArraySum ENDP
```


- ArraySum procedure 호출

```
.data
array  DWORD 1000h, 2000h, 3000h, 4000h, 5000h
theSum DWORD ?
.code
main PROC
    mov esi,OFFSET array
    mov ecx,LENGTHOF array
    call ArraySum
    mov theSum,eax
main ENDP
```


ArraySum procedure 정의 (다른 버전)

- 프로시저에서 값이 변하는 register를 사전에 스택에 저장
  → 프로시저 호출 후에도 register값이 보존됨

```
ArraySum PROC
; Receives: ESI = address of doubleword array
;           ECX = number of array elements.
; Returns:  EAX = sum
;-----------------------------------------------------
    push esi
    push ecx
    mov eax,0              ; set the sum to zero
L1: add eax,[esi]          ; add each integer to sum
    add esi,4              ; point to next integer
    loop L1                ; repeat for array size
    pop ecx
    pop esi
    ret
ArraySum ENDP
```

USES Operator

- USE operator
  - procedure 수행 동안 값이 수정되는 register들 중에서 값을 보존해야 하는 register들을 명시함

```
ArraySum PROC USES esi ecx
    mov eax,0              ; set the sum to zero
L1: add eax,[esi]
    add esi,4
    loop L1
    ret
ArraySum ENDP
```

  - Assembler는 이 register 값을 save 및 restore하기 위해서 procedure의 앞과 뒤에 push와 pop 명령어들 삽입

USES Operator (계속)

- 어셈블러가 생성한 코드

```
ArraySum PROC
    push esi
    push ecx
    mov eax,0
L1: add eax,[esi]
    add esi,4
    loop L1
    pop ecx
    pop esi
    ret
ArraySum ENDP
```


5.6 프로시저를 사용한 프로그램 설계

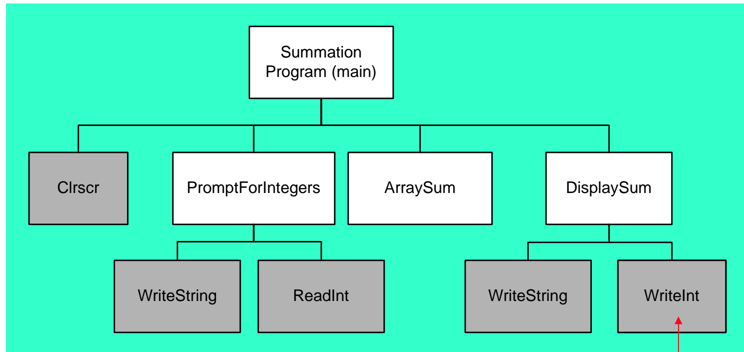
- Top-Down Design (functional decomposition)
- 예제: 정수들의 합을 계산
  - 정수 입력
    - 사용자에게 정수 입력 안내문을 제공
    - 입력한 정수들을 32비트 정수 배열에 저장
  - 정수 배열의 합을 계산
  - 합을 화면에 출력

프로시저 설계

- Specification

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

구조 챠트



- stub program (textbook 참조)
- complete program (textbook 참조)

gray indicates library procedure

Stub Program

```
TITLE Integer Summation Program       (Sum1.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
.code
main PROC
; Main program control procedure.
; Calls: Clrscr, PromptForIntegers,
;        ArraySum, DisplaySum

    exit
main ENDP

;-----------------------------------------------------
PromptForIntegers PROC
;
; Prompts the user for three integers, inserts
; them in an array.
; Receives: ESI points to an array of
;   doubleword integers, ECX = array size.
; Returns: nothing
; Calls: ReadInt, WriteString
;-----------------------------------------------------
    ret
PromptForIntegers ENDP

;-----------------------------------------------------
ArraySum PROC
;
; Calculates the sum of an array of 32-bit integers.
; Receives: ESI points to the array, ECX = array size
; Returns:  EAX = sum of the array elements
;-----------------------------------------------------
    ret
ArraySum ENDP

;-----------------------------------------------------
DisplaySum PROC
;
; Displays the sum on the screen.
; Receives: EAX = the sum
; Returns: nothing
; Calls: WriteString, WriteInt
;-----------------------------------------------------
    ret
DisplaySum ENDP
END main
```


Complete Program

```
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  PromptForIntegers
    call  ArraySum
    call  DisplaySum
    exit
main ENDP
;-----------------------------------------------------
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 intege
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

;-----------------------------------------------------
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

;-----------------------------------------------------
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
```