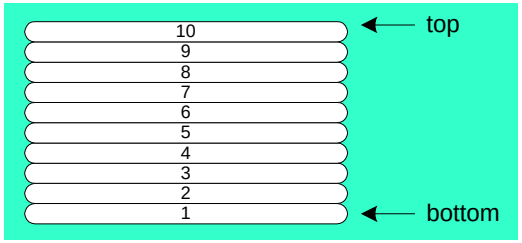
Chapter 5: Procedures (2)

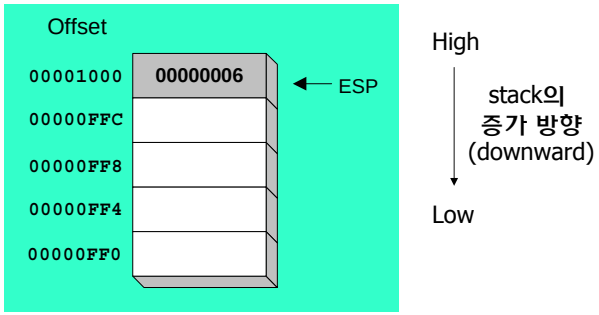
5.4 Stack Operations

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



Runtime Stack

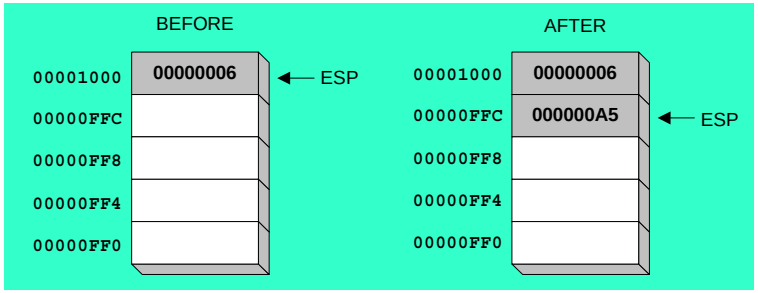
- Stack은 SS, ESP register에 의해서 관리됨
  - SS (stack segment) : stack segment descriptor register
  - ESP (stack pointer) : stack의 top 주소를 보관
    (가장 최근에 stack에 저장된 data의 위치)
  - real mode에서는 ESP 대신에 SP를 사용



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

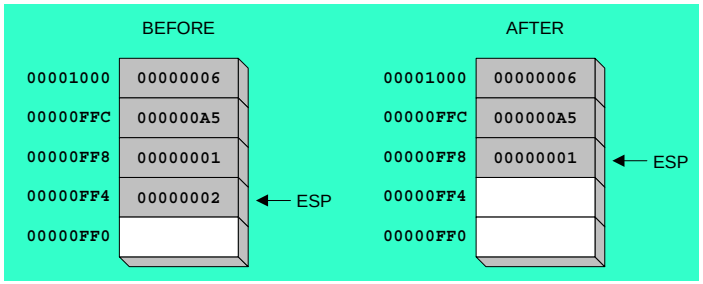
(pre-decrement addressing)



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 Instructions

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

Using PUSH and POP

- 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
```


Example: Nested Loop

- Push the outer loop counter before entering the inner loop:

```
   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
pop saveFlags         ──→ saveFlags ← EFLAGS
. . .
push saveFlags
popfd                 ──→ EFLAGS ← saveFlags
loopne
```


PUSH, POP All Registers

- PUSHAD
  - pushes 32-bit general-purpose registers on the stack
  - push 순서: EAX, ECX, EDX, EBX, ESP(원래값), EBP, ESI, EDI
- POPAD
  - pops the same registers off the stack
  - pop 순서: PUSHAD의 반대 순서
- PUSHA, POPA
  - 16-bit real mode에서 사용하여 16-bit general-purpose register를 push, pop함

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


Example: Reversing a String

- Reverse a String
  - 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 Defining and Using Procedures

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

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

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

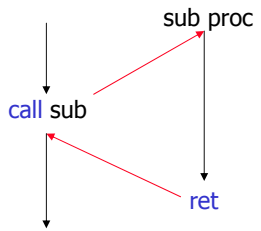

Documenting Procedures

- 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
```


CALL and RET Instructions

- 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

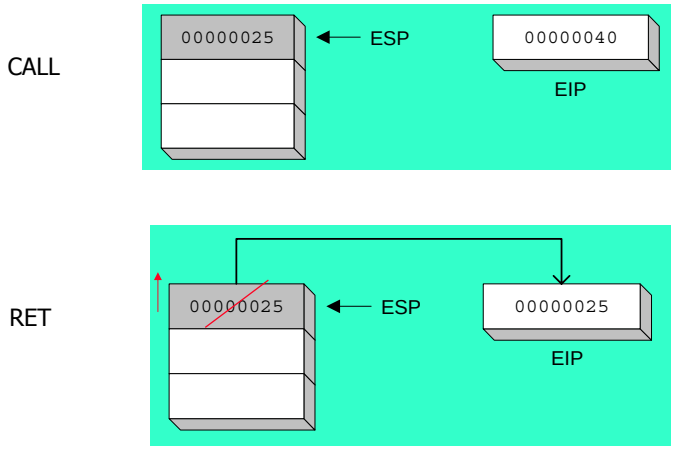


CALL, RET Example (1 of 2)

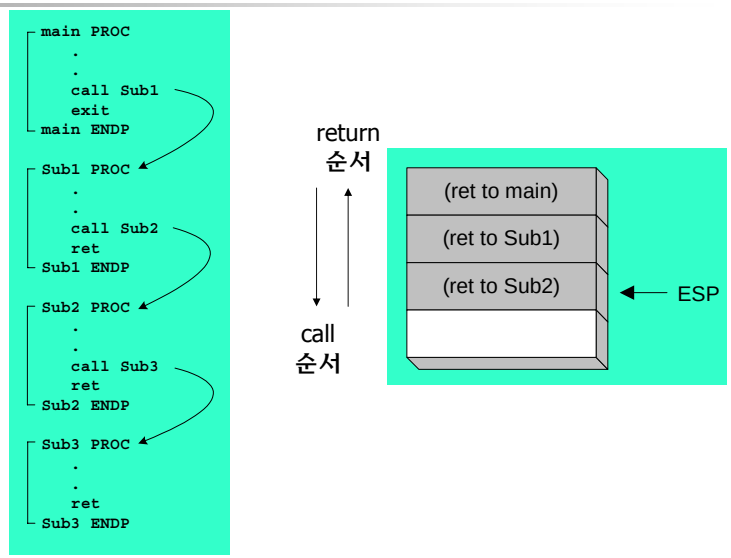
```
        main PROC
00000020  call MySub
00000025  mov eax,ebx
        .
        .
        main ENDP

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

offset

CALL-RET Example (2 of 2)



Nested Procedure Calls



Local and Global Labels

- local label
  - 같은 procedure 내에서만 사용 가능 (형식) local_label:
- global label
  - 모든 procedure에서 사용 가능 (형식) glob_label::

```
main PROC
    jmp L2                  ; error
L1::                        ; global label
    exit
main ENDP

sub2 PROC
L2:                         ; local label
    jmp L1                  ; ok
    ret
sub2 ENDP
```

OK

Error

Procedure Parameters

- Procedure로의 parameter 전달 방법
  - 특정 변수를 사용하는 것은 바람직하지 않음
  - 레지스터 또는 stack을 사용하여 parameter를 전달함

```
.data
theSum DWORD ?
.code
main PROC
    mov eax, 1000h
    mov ebx, 2000h
    mov ecx, 3000h
    call SumOf
    mov theSum, eax
SumOf ENDP
```

parameter 설정 (EAX, EBX, ECX)

procedure 호출

결과 저장

Example Procedure: Sum of Array

- 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
```


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)

- Code generated by Assembler

```
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 Program Design Using Procedures

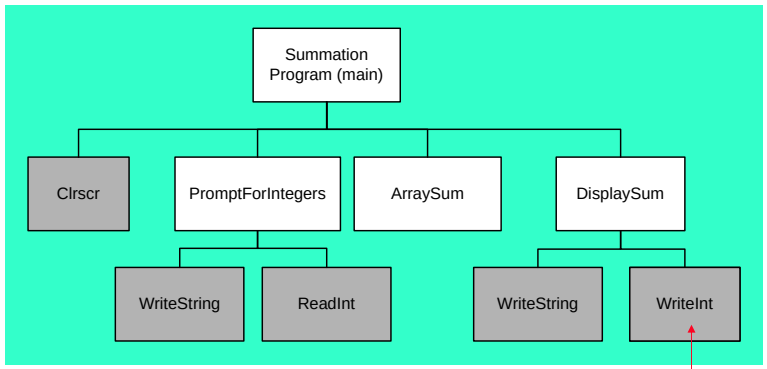
- Top-Down Design (functional decomposition)
- Example: 정수들의 합을 계산
  - 정수 입력
    - 사용자에게 정수 입력 안내문을 제공
    - 입력한 정수들을 32비트 정수 배열에 저장
  - 정수 배열의 합을 계산
  - 합을 화면에 출력

Procedure Design

- Specification

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


Structure Chart



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