Chapter 8: Advanced Procedures

Chapter Overview

- Local Variables
- Stack Parameters
- Stack Frames
- Recursion
- Creating Multimodule Programs

8.2 Local Variables

- Static global variable
  - all variables declared in the data segments
  - lifetime: program이 수행되는 동안 사용 (static)
  - scope: program 전체에서 사용 가능(global)

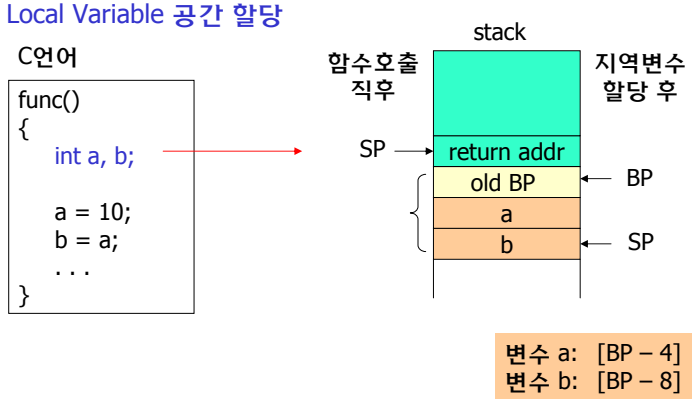
  (cf) C언어의 global variable

- Local variables
  - is created, used, and destroyed within a single procedure
  - lifetime/scope: procedure가 호출되어 수행되는 동안 procedure 내에서 사용
  - 기억장소를 다른 용도로 재사용할 수 있어서 효율적임

  (cf) C언어의 함수 내에서 선언된 local variable

Local Variable의 구현

- Local Variable은 stack을 사용하여 구현함
  - EBP(BP)에 local variable의 기준 위치를 저장함
  - EBP(BP)를 frame pointer라고도 부름
- Local Variable 공간 할당



변수 a: [BP – 4]
변수 b: [BP – 8]

Local Variable을 사용한 Procedure

- C언어

```
func()
{
    int a, b;

    a = 10;
    b = a;
    . . .
}
```

변수 a: [ebp – 4]
변수 b: [ebp – 8]

Assembly언어

```
func proc
    push ebp              ; ebp 보관
    mov  ebp, esp         ; ebp ← esp
    sub  esp, 8           ; esp ← esp–8

    mov  dword ptr [ebp-4], 10   ; a ← 10
    mov  eax, [ebp-4]
    mov  [ebp-8], eax     ; b ← a
    . . .
    mov esp, ebp          ; esp ← ebp
    pop ebp               ; ebp 복원
    ret
func endp
```


LOCAL directive

- LOCAL varlist
  - PROC directive 바로 다음에 위치하여 local variable들을 선언
  - variable은 type과 함께 선언

```
MySub PROC
    LOCAL var1:BYTE, var2:WORD, var3:SDWORD
```

- Example: array, pointer

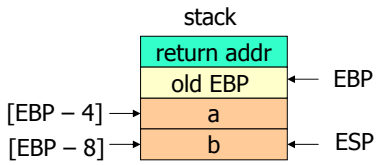
```
LOCAL flagVals[20]:BYTE     ; array of bytes

LOCAL pArray:PTR WORD       ; pointer to an array
```

- LOCAL directive의 구현
  - Assembler는 local variable를 stack을 사용하여 구현함

MASM-Generated Code

```
BubbleSort PROC
    LOCAL temp:DWORD, SwapFlag:BYTE
    . . .
    ret
BubbleSort ENDP
```

```
BubbleSort PROC
    push ebp
    mov  ebp,esp
    sub  esp,8
    . . .
    mov  esp,ebp
    pop  ebp
    ret
BubbleSort ENDP
```



Example: SumOf

```
SumOf PROC
    LOCAL tempSum:DWORD

    mov  tempSum,eax
    add  tempSum,ebx
    add  tempSum,ecx          ; tempSum = eax + ebx + ecx
    mov  eax,tempSum
    ret
BubbleSort ENDP
```

LOCAL directive를 사용하여 local variable을 선언하면 local variable을 [EBP – 4]와 같은 표기 대신에 tempSum과 같은 표기를 사용할 수 있어서 편리함

Reserving Stack Space

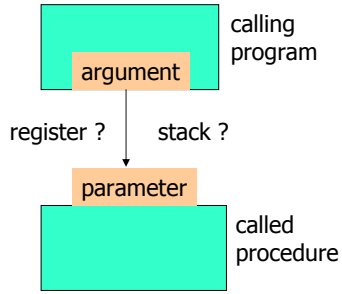
- Stack을 사용하기 위해서는 stack을 위한 공간을 할당해야 함
- STACK directive
  - stack segment의 크기를 지정

```
.stack 4096                    ; 4KB
```

  - irvine32.inc에 포함되어 있음 (4KB)

8.3 Stack Parameters

- Argument와 parameter
  - argument: calling program이 procedure에 전달하는 값
  - parameter: called procedure가 받는 값
- Procedure Parameter
  - register parameter
  - stack parameter



Register vs. Stack Parameters

- Register parameters
  - 각 parameter를 위한 register를 지정해야 함
  - parameter로 사용되는 register의 기존 내용은 호출이전에 저장해야 함.
- Stack parameters
  - 위와 같은 필요성이 없음

Register parameter사용 예

```
pushad
mov esi,OFFSET array
mov ecx,LENGTHOF array
mov ebx,TYPE array
call DumpMem
popad
```

Stack parameter사용 예

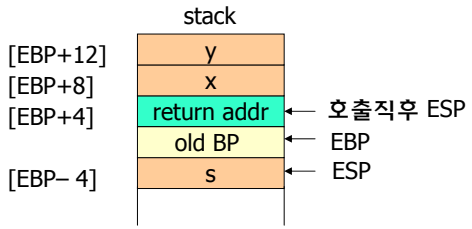
```
push OFFSET array
push LENGTHOF array
push TYPE array
call DumpMem
```

esi, ecx, ebx를 parameter용으로 사용

Stack Parameter 구현

C언어

```
func(int x, int y)
{
    int s

    s = x + y;
        . . .
}
```



parameter x : [EBP + 8]
parameter y : [EBP + 12]

Example

```
.data
sum DWORD ?
.code
    push 6                  ; second argument
    push 5                  ; first argument
    call AddTwo             ; EAX = sum
    add esp, 8
    mov  sum,eax            ; save the sum
```

```
AddTwo PROC
    push ebp
    mov ebp,esp             ; base of stack frame
    mov eax,[ebp + 12]      ; second argument (6)
    add eax,[ebp + 8]       ; first argument (5)
    . . .
    mov esp,ebp
    pop bp
    ret
AddTwo ENDP                 ; EAX contains the sum
```


RET Instruction

- Stack에 저장되어 있는 return address로 jump
  - EIP (IP) ← pop stack
- 형식
  - RET ; return
  - RET n ; return & stack pointer에 n을 더함

```
.data
sum DWORD ?
.code
    push 6
    push 5
    call AddTwo
    add esp, 8
    mov  sum,eax
```

```
AddTwo PROC
    push ebp
    mov ebp,esp
    mov eax,[ebp + 12]
    add eax,[ebp + 8]
    . . .
    mov esp,ebp
    pop bp
    ret 8
AddTwo ENDP
```


INVOKE Directive

- INVOKE procedureName [, argumentList]
  - CALL instruction이 여러 개의 argument를 사용할 때에 push와 call의 대신에서 사용할 수 있는 directive

```
push 6
push 5
call AddTwo
. . .
```

→

```
INVOKE AddTwo, 5, 6
. . .
```

- Argument의 유형
  - 상수, 상수식 (ex) 100, OFFSET myList, TYPE array
  - register (ex) eax, bl
  - variable (ex) myList, array
  - address 식 (ex) [myList+2], [ebx+esi]
  - ADDR name (ex) ADDR myList

ADDR Operator

- ADDR variable
  - variable에 대한 주소를 return함
  - INVOKE의 argument로 사용함
  - memory model에 따라서 주소의 크기가 다름
- memory model과 ADDR 값
  - Small model: 16-bit offset (near pointer)
  - Large model: 32-bit segment/offset (far pointer)
  - Flat model: 32-bit offset
- Example:

```
.data
myWord WORD ?
.code
INVOKE mySub, ADDR myWord
```

PROC Directive

- label PROC [, paramlist]
  . . .
  label ENDP
  - label: procedure 이름
  - paramlist: procedure의 parameter list (선택사항)
- paramList
  - 형식: paramName: type, paramName: type, . . .
  - type은 표준 MASM type (BYTE, SBYTE, WORD 등) 또는
    이 type에 대한 pointer(PTR BYTE, PTR WORD 등) type을 사용함

Example: AddTwo Procedure

- procedure AddTwo
  - parameters: var1, var2 (stack parameter)
  - result: eax ← var1 + var2

```
AddTwo PROC, val1:DWORD, val2:DWORD
   mov eax,val1
   add eax,val2
   ret
AddTwo ENDP
```

PROC directive를 사용하여 parameter list를 선언하면
parameter를 [EBP+12] 와 같은 표기 대신에
val1, val2와 같은 표기를 사용할 수 있어서 편리함

PROC Examples

- procedure FillArray
  - parameter: BYTE 배열의 주소, BYTE배열을 채울 값, 배열크기

```
FillArray PROC,
    pArray:PTR BYTE,
    fillVal:BYTE,
    arraySize:DWORD

    mov ecx,arraySize
    mov esi,pArray
    mov al,fillVal
L1: mov [esi],al
    inc esi
    loop L1
    ret
FillArray ENDP
```

parameter

PROC Examples

```
Swap PROC,
   pValX:PTR DWORD,
   pValY:PTR DWORD
   . . .
Swap ENDP
```

```
ReadFile PROC,
   pBuffer:PTR BYTE
   LOCAL fileHandle:DWORD
   . . .
ReadFile ENDP
```

parameter

local variable

PROTO Directive

- label PROTO paramList
  - procedure의 prototype 선언
  - procedure가 정의되기 전에 사용되는 경우에 필요함
    (C/C++의 prototype 선언과 같은 용도)
  - INVOKE directive를 사용하여 호출되는 procedure는 반드시 prototype을 가져야 함

```
MySub PROTO ...      ; procedure prototype

.code
   INVOKE MySub      ; procedure call
   . . .

MySub PROC ...       ; procedure implementation
   . . .
MySub ENDP
```


Example: ArraySum

- Prototype, Invocation

```
ArraySum PROTO,ptrArray:PTR DWORD,szArray:DWORD
.data
array DWORD 10000h, 20000h, 30000h, 40000h, 50000h
Sum   DWORD ?
.code
main PROC
   INVOKE ArraySum, ADDR array, LENGTHOF array
   mov Sum, eax
   . . .
main ENDP
```


Example: ArraySum(계속)

- Implementation

```
ArraySum PROC,ptrArray:PTR DWORD,szArray:DWORD
   mov esi,ptrArray
   mov ecx,szArray
   cmp ecx,0
   je L2
   mov eax, 0
L1: add eax,[esi]
   add esi, 4
   loop L1
L2: ret
```


Passing by Value or by Reference

- Passing by Value
  - 32-bit 값을 stack에 push (16-bit mode에서는 16-bit값 push 가능)

```
.data
myData DWORD 10000h
.code
main PROC
   INVOKE Sub1,myData
```

MASM →

```
push myData
call Sub1
```

- Passing by Reference
  - address를 stack에 push

```
.data
myData WORD 1000h
.code
main PROC
   INVOKE Sub1,ADDR myData
```

→

```
push OFFSET myData
call Sub1
```

Parameter Classifications

parameter 종류	passing by value	passing by reference
input	O	O (변수 값을 수정하지 않음)
output	X	O (변수 값을 사용하지 않음)
input-output	X	O (변수 값을 사용하며 수정도 함)

Example: Exchanging Two Integers

- Swap procedure
  - 두 32비트 정수변수의 값을 교환

```
Swap PROC USES eax esi edi,
   pValX:PTR DWORD,        ; pointer to first integer
   pValY:PTR DWORD         ; pointer to second integer

   mov esi,pValX           ; get pointers
   mov edi,pValY
   mov eax,[esi]           ; get first integer
   xchg eax,[edi]          ; exchange with second
   mov [esi],eax           ; replace first integer
   ret
Swap ENDP
```

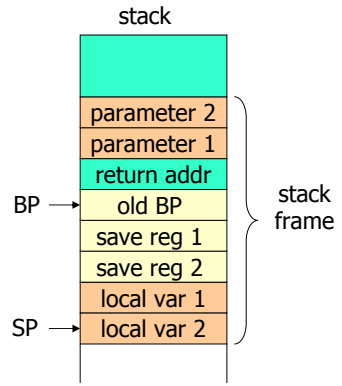
pValX, pValY는 input-output parameter임

Trouble-Shooting Tips

- Save and restore registers
  - procedure에서 값이 수정될 때에 (예외: return값 저장용 register)
- Wrong operand size
  - operand size에 따라서 배열 원소의 주소 계산에 주의해야 함
    [DArray + 1] (x) [DArray + 4] (o)
- Wrong type of pointer
  - assembler는 PTR BYTE와 PTR DWORD등을 서로 구분하지 못하므로 다른 type에 대한 pointer를 인수로 전달하지 않아야 함
- Passing wrong immediate values
  - reference parameter로 상수를 전달하지 않아야 함

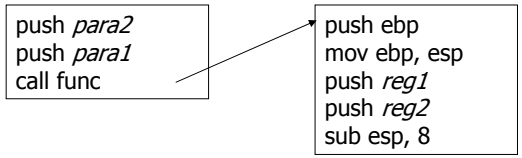
8.4 Stack Frames

- Stack Frame
  - activation record라고도 함
  - procedure의 다음 사항을 저장한 stack 부분
    - return address
    - passed parameter
    - saved register
    - local variable



Construction of Stack Frame

- Stack Frame 구성 과정
  - push arguments on the stack
  - call procedure
  - push EBP on the stack and set EBP to ESP
  - push modified registers on the stack
  - a constant is subtracted from ESP to make room on the stack for local variable



Memory Models

- Memory Model
  - code와 data segment의 개수와 크기를 결정함
- Real mode의 memory model

tiny	1 segment (code, data 모두 포함), COM 프로그램
small	1 code segment, 1 data segment
medium	multiple code segments, 1 data segment
compact	1 code segment, multiple data segments
large	multiple code, data segments
huge	single data segment보다 큰 data 사용 가능

- Protected mode의 memory model
  - flat model : 32-bit offset 사용 (최대 4GB)
    single segment에 모든 data와 code가 포함됨

.MODEL Directive

- .MODEL *model* [*, modeloptions*]
  - program's memory model과 model options 지정
  - *modeloptions* 은 language specifier를 포함
- Language Specifiers
  - procedure naming scheme, parameter passing conventions 지정

specifier	argument push order	clean up argument stack
C	reverse order	calling program
pascal	forward order	called procedure
stdcall	reverse order	called procedure

  - irvine32.inc에서는 stdcall을 사용

ret 8 add sp, 8

LEA Instruction

- LEA reg, mem (load effective address)
  - 동작: reg ← mem의 offset
- LEA instruction과 OFFSET operator
  - OFFSET operator는 direct operand에 대한 offset만 사용가능
  - LEA instruction은 direct와 indirect operand모두 사용 가능
    (local variable, stack parameter 모두 indirect operand임)
- LEA instruction은 local variable, stack parameter의 주소를 필요로 할 때 사용해야 함

```
CopyString PROC, count:DWORD
   LOCAL temp[20]:BYTE

   mov edi,OFFSET count       ; invalid operand
   mov esi,OFFSET temp        ; invalid operand
   lea edi,count              ; ok
   lea esi,temp               ; ok
```

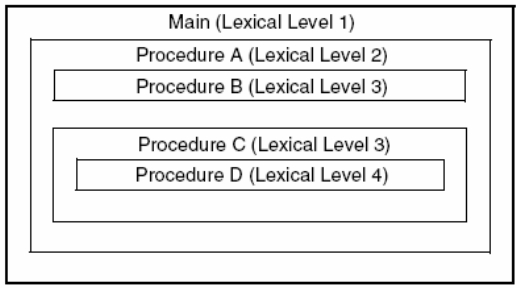
ENTER and LEAVE Instructions

- ENTER localbytes, nestinglevel
  - procedure을 위한 stack frame을 만듬
    - localbytes: local variable이 사용하는 stack공간의 크기 (byte)
    - nestinglevel: 일부고급언어에서 사용하는 nesting level
      (assembler, FORTRAN, C 등에서는 사용하지 않음, 0)
- LEAVE
  - stack frame을 없앰, ENTER의 반대 동작

```
MySub PROC, arg:BYTE
    enter 8,0
    . . .
    leave
    ret 4
MySub ENDP
```

```
MySub PROC, arg:BYTE
    push ebp
    mov ebp,esp
    sub esp, 8
    . . .
    mov esp,ebp
    pop ebp
    ret 4
MySub ENDP
```

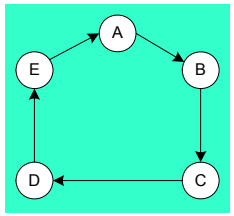

Nesting Level과 Block Structured Language



- procedure A: Main의 stack frame 사용가능
- procedure B: Main, A의 stack frame 사용가능
- procedure C: Main, A의 stack frame 사용가능
- procedure D: Main, A, C의 stack frame 사용가능

8.5 Recursion

- What is Recursion?
  - procedure가 직접, 또는 간접적으로 자신을 호출하는 것
- Call graph
  - recursion은 cycle을 형성함



Recursively Calculating a Sum

- CalcSum procedure: 정수의 합을 재귀적으로 계산
  - Receives: ECX = count. Returns: EAX = sum)

```
CalcSum PROC
    cmp ecx,0              ; check counter value
    jz L2                  ; quit if zero
    add eax,ecx            ; otherwise, add to sum
    dec ecx                ; decrement counter
    call CalcSum           ; recursive call
L2: ret
CalcSum ENDP
```

Stack frame:

Pushed On Stack	ECX	EAX
L1	5	0
L2	4	5
L2	3	9
L2	2	12
L2	1	14
L2	0	15

Calculating a Factorial

```
int factorial(int n)
{
    if(n == 0)
      return 1;
    else
      return n * factorial(n-1);
}
```

call path ⟶

factorial(5) → factorial(4) → factorial(3) → factorial(2) → factorial(1) → factorial(0)

5*4! ← 4*3! ← 3*2! ← 2*1! ← 1*0! ← 1

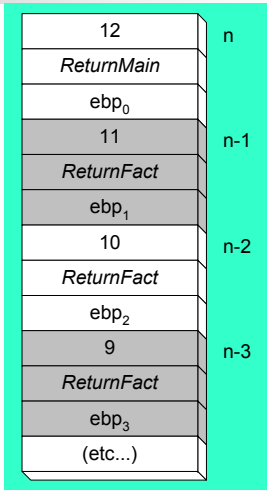
⟵ return path

Calculating a Factorial

```
Factorial PROC
    push ebp
    mov  ebp,esp
    mov  eax,[ebp+8]          ; get n
    cmp  eax,0                ; n < 0?
    ja   L1                   ; yes: continue
    mov  eax,1                ; no: return 1
    jmp  L2

L1: dec  eax
    push eax                  ; Factorial(n-1)
    call Factorial

; Instructions from this point on execute when each
; recursive call returns.
ReturnFact:
    mov  ebx,[ebp+8]          ; get n
    mul  ebx                  ; eax = eax * ebx

L2: mov  esp,ebp              ; 없어도 됨
    pop  ebp                  ; return EAX
    ret  4                    ; clean up stack
Factorial ENDP
```


Calculating a Factorial

- 12! 계산할 때의 stack
  - 각 recursive call은 12byte의 stack공간을 사용



8.6 Creating Multimodule Programs

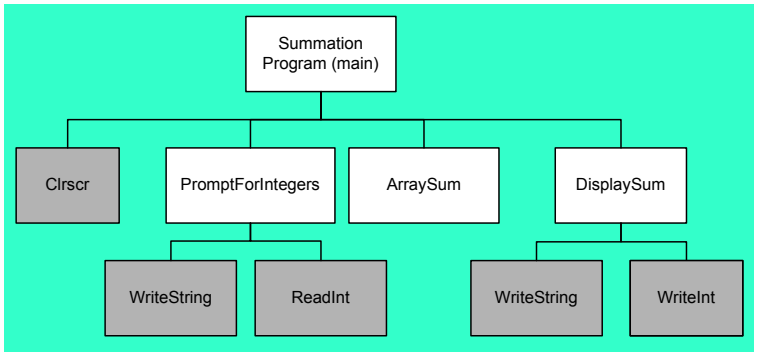
- A multimodule program
  - source코드가 여러 개의 파일(module)로 나누어서 작성된 프로그램
- 각 module은 분리된 OBJ file로 어셈블 또는 컴파일됨
- 모든 OBJ file들은 LINK utility를 사용하여 하나의 EXE file로 link됨 (static linking)
- 장점
  - 커다란 프로그램의 작성, 유지보수, 디버깅 용이

Multimodule Program의 구성

- Multimodule program의 구성
  - main module 작성 – main procedure 포함
  - procedure 또는 related procedure의 집합을 별도의 source code module로 작성
  - PROTO directive를 사용한 procedure의 prototype 선언을 포함하는 include file 작성
  - include file을 INCLUDE directive를 사용하여 포함

Example: ArraySum Program

- ArraySum program의 구조도



INCLUDE File

- Include file "sum.inc"

```
INCLUDE Irvine32.inc

PromptForIntegers PROTO,
    ptrPrompt:PTR BYTE,         ; prompt string
    ptrArray:PTR DWORD,         ; points to the array
    arraySize:DWORD             ; size of the array

ArraySum PROTO,
    ptrArray:PTR DWORD,         ; points to the array
    count:DWORD                 ; size of the array

DisplaySum PROTO,
    ptrPrompt:PTR BYTE,         ; prompt string
    theSum:DWORD                ; sum of the array
```


Individual Modules

- Main
- PromptForIntegers
- ArraySum
- DisplaySum

(textbook의 source file 참조)

- Assemble and Link
  - Custom batch file 사용 (textbook참조)
  - Make32는 single module program을 위한 batch file임