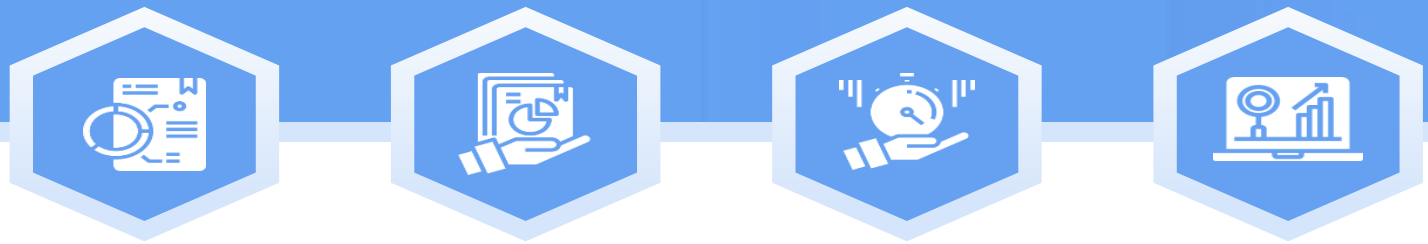
시스템 보안

어셈블리어 실습 - 2



KOREA UNIVERSITY
SEJONG CAMPUS

어셈블리어 실습

- **실습 – C 함수 호출 (파일 경로 체크) (1/4)**

  - SASM에서 아래 코드를 입력하여 결과를 확인, access.asm

```
%include "io.inc"

extern _access

section .data
    buffer times 0xff db 0x0
    buffer_len dd 0xff
```

255byte 크기의 변수 선언 : byte buffer[255]

_access() 함수를 이용하여 파일의 존재 여부를 확인

외부 함수이기 때문에 extern 선언

C

```
int _access(
   const char *path,
   int mode
);
```

mode 값	파일 검사
00	존재만
02	쓰기 전용
04	읽기 전용
06	읽기 및 쓰기

- 리턴 값이 0이면 성공 -1이면 실패

※ _access() 함수는 범용 CRT 함수로 아래 경로의 라이브러리 파일이 있음

C:₩Program Files (x86)₩SASM₩MinGW₩lib₩libmsvcrt.a

https://learn.microsoft.com/ko-kr/cpp/windows/universal-crt-deployment?view=msvc-170

어셈블리어 실습

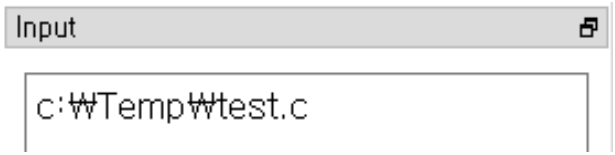
- **실습 – C 함수 호출 (파일 경로 체크) (2/4)**

```
section .text
global CMAIN
CMAIN:
    push ebp
    mov ebp, esp

    mov ecx, [buffer_len]
    GET_STRING buffer, ecx

    call removeLF

    PRINT_STRING buffer
    NEWLINE

    push 0x0
    push buffer
    call _access
    add esp, 8

    PRINT_HEX 4, eax
    NEWLINE


    mov esp, ebp
    pop ebp

    xor eax, eax
    ret
```

- 문자열 입력 받기

SASM에서 실행 시에는 미리 입력 창에 문자열을 입력해 놓아야 함

입력 창에 경로를 입력하면서 줄 바꿈을 하면 LF(Line Feed) 문자도 버퍼에 저장되어 별도 처리 필요



- 스택 프레임 생성 및 해제를 위한 프롤로그와 에필로그 코드

어셈블리어 실습

- **실습 – C 함수 호출 (파일 경로 체크) (3/4)**

```
section .text
global CMAIN
CMAIN:
    push ebp
    mov ebp, esp

    mov ecx, [buffer_len]
    GET_STRING buffer, ecx

    call removeLF

    PRINT_STRING buffer
    NEWLINE

    push 0x0
    push buffer
    call _access
    add esp, 8

    PRINT_HEX 4, eax
    NEWLINE

    mov esp, ebp
    pop ebp

    xor eax, eax
    ret
```

```
; removeLF 프로시져는 파라메터나 지역변수 사용 등과 같은 스택 관련 작업이 없기 때문에
; 프롤로그 및 에필로그 코드는 없어도 무방
removeLF:
    mov ecx, [buffer_len]
    mov ebx, buffer
    check_loop:
        cmp byte [ebx], 0x0a        ; LF(Line Feed, 0x0a) 문자 여부 비교
        jnz loop_next               ; LF 문자가 아닌 경우(ZF != 0) loop_next로 이동
        mov byte [ebx], 0x0         ; LF(Line Feed) 문자를 NULL 문자로 치환
        jmp end
    loop_next:
        inc ebx
        loop check_loop
    end:
        ret
```

- 입력 시 줄 바꿈을 하면 LF(Line Feed) 문자도 버퍼에 저장되기 때문에 제거 필요
- 문자열 버퍼 크기 만큼 순회 하면서 LF 문자 = `0x0a`를 만나면 NULL 문자 = `0x0`으로 변경 후 종료

어셈블리어 실습

- **실습 – C 함수 호출 (파일 경로 체크) (4/4)**

```
section .text
global CMAIN
CMAIN:
    push ebp
    mov ebp, esp

    mov ecx, [buffer_len]
    GET_STRING buffer, ecx

    call removeLF

    PRINT_STRING buffer
    NEWLINE

    push 0x0
    push buffer
    call _access
    add esp, 8

    PRINT_HEX 4, eax
    NEWLINE


    mov esp, ebp
    pop ebp

    xor eax, eax
    ret
```

Input

```
c:₩temp₩fs.txt
```

```
push 0x0        ②
push buffer     ①
call _access
add esp, 8
```

스택 정리

Output

```
c:₩temp₩fs.txt
0
```

```
int _access(
    const char *path, ①
    int mode ②
);
```

mode 값	파일 검사
00	존재만

※ 리턴 값 :
파일에 지정된 모드가 있으면 0을 반환
명명된 파일이 없거나 지정된 모드가 없는 경우 -1을 반환

어셈블리어 실습

- ## 실습 – C 함수 호출 (printf) (1/2)

  - SASM에서 아래 코드를 입력하여 결과를 확인, printf.asm

```
%include "io.inc"

; 외부 함수 선언
extern printf

; 프롤로그(prologue) 매크로 정의
; 새로운 스택 프레임을 설정
%macro PROLOGUE 0
    push ebp
    mov ebp,esp
%endmacro

; 에필로그(epilogue) 매크로 정의
; 현재의 스택 프레임을 해제
%macro EPILOGUE 0
    mov esp, ebp
    pop ebp
%endmacro

section .data
    msg db 'Hello World!', 0x0d, 0x00
```

printf() 함수를 이용하여 문자열 출력

외부 함수이기 때문에 extern 선언

```
C

int printf(
   const char *format [,
   argument]...
);
```

```
C

printf("Line one\n\t\tLine two\n");
```

프롤로그, 에필로그 코드를 매크로로 정의

- %define, %macro 기능은 어셈블리 언어의 기능이 아닌 NASM 어셈블러에서 제공하는 기능으로 특정 작업을 수행하기 위해서 미리 정의한 명령어 모음
- %define은 한 줄만 작성 가능하지만, %macro는 여러 줄로 코드를 작성 가능

출력할 문자열 설정

※ printf() 함수는 범용 CRT 함수로 아래 경로의 라이브러리 파일이 있음

C:₩Program Files (x86)₩SASM₩MinGW₩lib₩libmsvcrt.a

https://learn.microsoft.com/ko-kr/cpp/windows/universal-crt-deployment?view=msvc-170

어셈블리어 실습

- **실습 – C 함수 호출 (printf) (2/2)**

```
section .text
global CMAIN
CMAIN:
    call print_msg          ; print_msg 프로시저 호출
    call print_msg_macro    ; print_msg_macro 프로시저 호출
    xor eax, eax
    ret
```

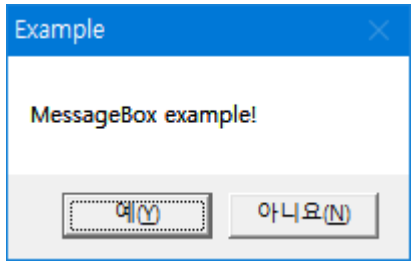
```
; 매크로 사용 안하는 코드
print_msg:
    ; 프롤로그
    push ebp
    mov ebp,esp

    push msg
    call printf    ; printf C 함수 호출
    add esp, 4     ; C 함수 호출이기 때문에 스택 정리

    ; 에필로그
    mov esp, ebp
    pop ebp

    ret
```

```
; 매크로 사용 코드
print_msg_macro:
    ; 프롤로그 매크로
    PROLOGUE

    push msg
    call printf    ; printf C 함수 호출
    add esp, 4     ; C 함수 호출이기 때문에 스택 정리

    ; 에필로그 매크로
    EPILOGUE

    ret
```

어셈블리어 실습

- **실습 – Windows API 호출(스택을 이용하여 함수 인자 전달) (1/3)**

  - SASM에서 아래 코드를 입력하여 결과를 확인, messagebox.asm




```
C++

int MessageBoxA(
  [in, optional] HWND   hWnd,
  [in, optional] LPCSTR lpText,
  [in, optional] LPCSTR lpCaption,
  [in]           UINT   uType
);
```

- MessageBoxA() 함수 정의

```
%include "io.inc"

; 함수 선언 시 '@4' 같은 문자가 붙는 이유는 라이브러리 내에 그 이름으로 선언되어 있기 때문
extern printf             ; printf 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)
extern MessageBoxA@16     ; MessageBoxA 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)
extern ExitProcess@4      ; ExitProcess 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)
```

※ 외부 함수이기 때문에 extern 선언

printf() 함수는 C 함수(범용 CRT 함수)

※ MessageBoxA()와 ExitProcess() 함수는 Windows API(이전에는 Win32 API로 부름)로 함수를 제공하는 DLL에 따라서 참조하는 라이브러리가 다름

MessageBoxA() 함수는 user32.dll, ExitProcess() 함수는 kernel32.dll에서 함수를 제공하며

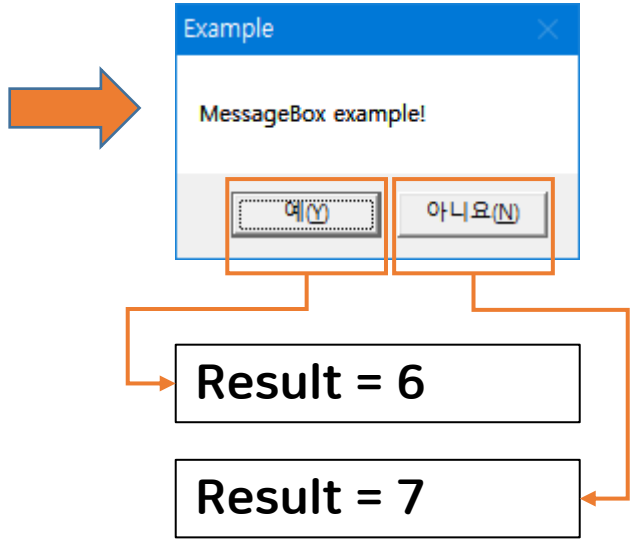
C:₩Program Files (x86)₩SASM₩MinGW₩lib 경로에 libuser32.a(user32.dll), libkernel32.a(kernel32.dll) 라이브러리에 정의되어 있음

주의 : 해당 라이브러리 파일에서 API를 검색했을 때 '@숫자'가 API 뒤에 붙어 있는 경우, 이런 경우 해당 숫자를 포함해서 선언해야 함

어셈블리어 실습

- **실습 – Windows API 호출(스택을 이용하여 함수 인자 전달) (2/3)**

```
%include "io.inc"

; 함수 선언 시 '@4' 같은 문자가 붙는 이유는 라이브러리 내에 그 이름으로 선언되어 있기 때문
extern printf             ; printf 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)
extern MessageBoxA@16     ; MessageBoxA 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)
extern ExitProcess@4      ; ExitProcess 함수 선언(외부 라이브러리를 이용하기 때문에 선언만)

section .data
    format db 'Result = %d', 0x0d, 0x00         ; printf 함수에 사용할 포맷 문자열
    caption db 'Example', 0x00                  ; MessageBox의 캡션 문자열
    text db 'MessageBox example!', 0x0d, 0x00   ; MessageBox에 표시할 문자열

section .text
global CMAIN
CMAIN:
    push dword 0x4              ; MessageBoxA의 네 번째(MessageBox 타입) 파라미터 설정
    push dword caption          ; MessageBoxA의 세 번째(caption) 파라미터 설정
    push dword text             ; MessageBoxA의 두 번째(text) 파라미터 설정
    push dword 0x0              ; MessageBoxA의 첫 번째(hWnd) 파라미터 설정 - 자기 자신은 0
    call MessageBoxA@16         ; MessageBoxA 함수(Windows API) 호출

    push eax                    ; printf의 두 번째(정수값) 파라미터 설정
    push format                 ; printf의 첫 번째(출력 포맷) 파라미터 설정
    call printf                 ; printf 함수 호출 - printf("%d", eax)
    add esp, 8                  ; C 함수이기 때문에 _cdecl 호출방식을 사용하고, 따라서 스택 정리 필요
                                ; 스택에 8byte 크기 만큼 push 했기 때문에 +8을 해서 스택 정리

    push 0                      ; 종료 코드 설정
    call ExitProcess@4          ; ExitProcess(프로세스 종료 함수 - Windows API) 호출

    ret
```



어셈블리어 실습

- **실습 – Windows API 호출(스택을 이용하여 함수 인자 전달) (3/3)**

  - SASM에서 아래 코드를 입력하여 결과를 확인, messagebox_proc.asm

```
%include "io.inc"

; 프롤로그(prologue) 매크로 정의
; 새로운 스택 프레임을 설정
%macro PROLOGUE 0
    push ebp
    mov ebp,esp
%endmacro

; 에필로그(epilogue) 매크로 정의
; 현재의 스택 프레임을 해제
%macro EPILOGUE 0
    mov esp, ebp
    pop ebp
%endmacro

extern printf
extern MessageBoxA@16
extern ExitProcess@4

section .data
    format db 'Result = %d', 0x0d, 0x00
    caption db 'Example', 0x00
    text db 'MessageBox example!', 0x0d, 0x00
```

```
section .text
global CMAIN
CMAIN:
    call message_box

    call print_result

    push 0
    call ExitProcess@4

    xor eax, eax
    ret
```

프로시저를 이용한 호출

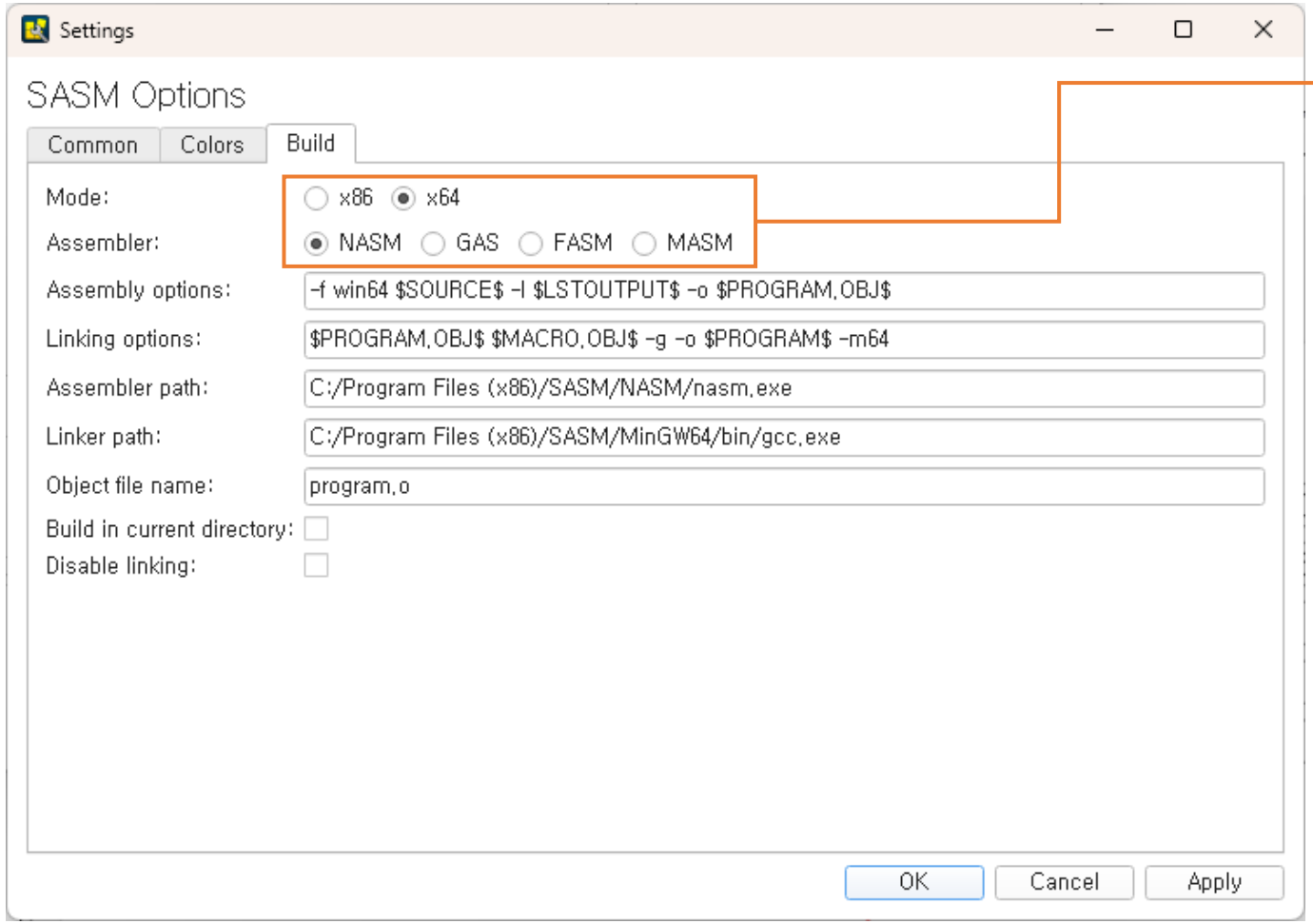
```
; message_box 프로시져
message_box:
    PROLOGUE

    push dword 0x4
    push dword caption
    push dword text
    push dword 0x0
    call MessageBoxA@16

    EPILOGUE
    ret
```

```
; print_result 프로시져
print_result:
    PROLOGUE

    push eax
    push format
    call printf
    add esp, 8

    EPILOGUE
    ret
```

어셈블리어 실습

- **실습 – 준비(64bit)**

  - SASM Options



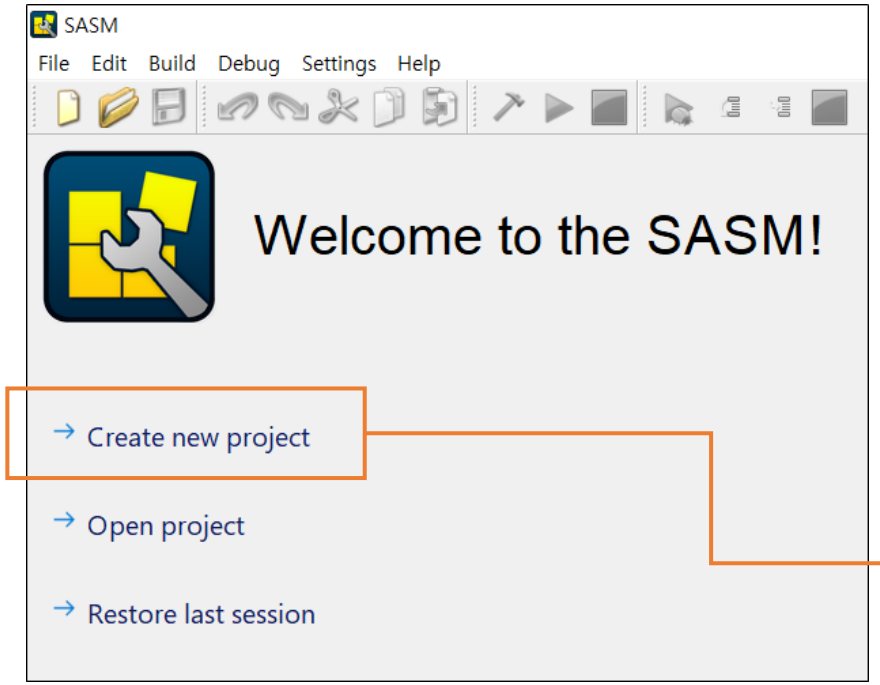
※ NASM(Netwide Assembler)

Mode는 x64(64bit)로 설정

어셈블리어 실습

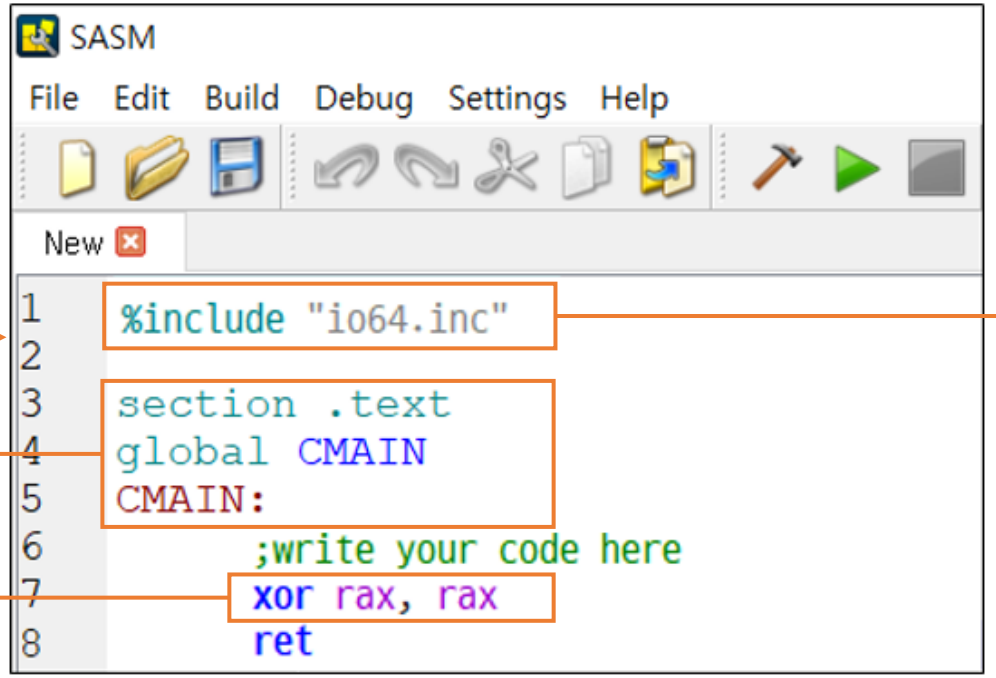
- **실습 – 준비(64bit)**

  - SASM Builtin functions & macro



※ "io.inc" macro library for NASM

64bit에서는 io64.inc



※ .text 섹션에 C에서의 main() 함수인 CMAIN 선언

※ 레지스터가 32bit eax -> 64bit rax로 변경

어셈블리어 실습

- **실습 – 메모리 저장 순서 확인(64bit)**

  - SASM에서 아래 코드를 입력하여 결과를 확인, memory_littel_endian_x64.asm

```
%include "io64.inc"

section .data               ; 데이터(초기화 된 데이터) 영역임을 선언
    var1 dw 0x1234
    var2 dd 0x55555555
    var3 dq 0x6666777788889999


section .text               ; 코드 영역임을 선언
global CMAIN
CMAIN:
1   PRINT_HEX 1, [var1]     ; 1바이트 출력
    NEWLINE
2   PRINT_HEX 2, [var1]     ; 2바이트 출력
    NEWLINE
3   PRINT_HEX 4, [var1]     ; 4바이트 출력 (변수 경계를 넘어서 출력됨)
    NEWLINE
    mov rax, 0x1234567890abcdef
    mov [var2], rax         ; rax에 저장된 값으로 var2 변수 값 변경
4   PRINT_HEX 4, [var2]     ; var2의 값 4바이트 출력
    NEWLINE
5   PRINT_HEX 8, [var2]     ; var2의 값 8바이트 출력
    NEWLINE
6   PRINT_HEX 8, [var3]     ; var3의 값 8바이트 출력
    NEWLINE
7   PRINT_HEX 4, [var1]     ; var1의 값 4바이트 출력
    NEWLINE

    xor rax, rax            ; 종료 코드 설정 (0은 정상 종료)
    ret
```

- 실제 메모리에는 var1, var2, var3 순서로 데이터가 저장되는데, 리틀 엔디안 방식으로 저장되기 때문에 아래와 같이 저장
- 초기 상태 :

34 12	55 55 55 55	99 99 88 88 77 77 66 66

- var2 변수의 값을 변경한 이후 상태 :

34 12	ef cd ab 90	78 56 34 12 77 77 66 66

1. 34
2. 1234
3. 55551234
4. 90abcdef
5. 1234567890abcdef
6. 6666777712345678
7. cdef1234

Q A