어셈블리어 발표 스크립트

#1

자기 소개와 주제 소개

#2

목차 설명

#3

어셈블리어란?

- 컴퓨터와 통신을 위해 사용되는 프로그래밍 언어
- 기계와 일대일 대응이 되는 저급 언어
  - 저급 언어는 컴퓨터 시스템의 하드웨어와 직접적으로 상호작용하는 언어로 기계어나 어셈블리어와 같은 형태로 표현됩니다.
  - 이외에도 고급 언어는 사람이 이해하기 쉽고 추상화된 수준에서 프로그래밍할 수 있는 언어입니다.

#4

어셈블리어 구조

- 어셈블리어에는 Intel과 AT&T 문법이 존재
- 보통 윈도우에서는 Intel 문법, 리눅스에서는 AT&T 문법 사용
- Intel 문법과 AT&T의 가장 큰 차이점은 제1피연산자와 제2피연산자의 위치
  ex)
  Intel → ADD Operand1(Destination), Operand2(Source)
  AT&T → ADD Operand1(Source), Operand2(Destination)

#5

레지스터란?

- 처리 중인 데이터나 처리 결과를 임시 보관하는 CPU 안의 기억 장치

- 레지스터의 종류와 기능

범주	레지스터	이름	비트	용도
범용	EAX	누산기	32	산술 연산
범용	EBX	베이스 레지스터	32	특정 주소 저장
세그먼트	CS	코드 세그먼트 레지스터	16	실행할 기계 명령어가 저장된 메모리 주소 지정
세그먼트	DS	데이터 세그먼트 레지스터	16	프로그램에서 정의된 데이터, 상수, 작업 영역 메모리 주소 지정
포인터	EBP	베이스 포인터	32	스택안의 변수 값을 읽음, SS 레지스터와 함께 사용
포인터	ESP	스택 포인터	32	스택의 가장 끝 주소를 가리킴, SS 레지스터와 함께 사용
인덱스	EDI	목적지 인덱스	32	목적지 주소 값 저장
인덱스	ESI	출발지 인덱스	32	출발지 주소 값 저장

#6

스택

스택은 어셈블리어 프로그래밍에서 중요한 개념이며, 어셈블리어 코드에서 스택을 사용하여 데이터를 저장하고 관리합니다.

- 데이터 임시 저장을 위한 메모리 공간
- LIFO(후입선출, Last-In-First-Out) 구조
    - 스택은 후입선출(Last-In-First-Out, LIFO) 구조를 가지는 데이터 구조입니다. 이는 가장 최근에 추가된 데이터가 가장 먼저 제거되는 구조를 의미합니다.

명령어는 데이터를 스택에 PUSH하거나 POP하는 데 사용합니다.

A

PUSH A

B
A

PUSA A → PUSH B

C
B
A

PUSA A → PUSH B → PUSH C

B
A

PUSA A → PUSH B → PUSH C → POP C

#7

기본 명령어

산술 연산 명령어

ADD

- 제1피연산자와 제2피연산자 값을 더한 결과 값을 제1피연산자에 저장
- EAX = 10
  ADD EAX, 5
  EAX = EAX(10) + 5 = 15

EAX 레지스터에 10이 할당되고, 그 다음 명령어인 ADD EAX, 5를 통해 EAX에 5를 더한 결과가 EAX에 다시 할당됩니다. 따라서 EAX의 최종 값은 15가 됩니다

SUB

- 제1피연산자에서 제2피연사자 값을 뺀 결과 값을 제1피연산자에 저장
- EAX = 10
  SUB EAX, 5
  EAX = EAX(10) - 5 = 5

마찬가지고 주어진 코드에서 EAX 레지스터에 10이 할당되고, 그 다음 명령어인 SUB EAX, 5를 통해 EAX에서 5를 뺀 결과가 EAX에 다시 할당됩니다. 따라서 EAX의 최종 값은 5가 됩니다.

#8

데이터 전송 명령

MOV

- 데이터 값 이동할 때 사용
- MOV EAX, [EBP+3]

LEA

- 데이터 값을 이동할 때 사용
- LEA EAX, [EBP+3]

둘 다 데이터 값을 이동할 때 사용되지만 차이점이 있습니다. MOV EAX, [EBP+3] 의 경우 [EBP+3]를 하나의 주소 값으로 처리하고, LEA EAX, [EBP+3] 의 경우 [EBP]만 주소 값으로 인식하여 +3는 주소 값에 대한 추가 연산으로 처리합니다. 즉, 제2연산자에 대한 추가 연산 방식이 다르다고 할 수 있습니다.

#9

분석

함수 프롤로그는 함수가 호출될 때 실행되는 코드이다.

```
① PUSH EBP
② MOV EBP, ESP
```

① push ebp : 함수가 종료된 후 ebp를 이전 함수의 ebp로 재설정하기 위해 스택에 이전 함수의 ebp를 push 한다.

② mov ebp, esp : 호출된 함수의 시작을 알리기 위해 현재 esp 값을 ebp에 복사한다.

#10

함수 에필로그

함수 에필로그는 함수가 종료될 때 실행되는 코드

```
LEAVE
RET
```

함수 프롤로그

함수 프롤로그는 함수가 호출될 때 실행되는 코드

```
① PUSH EBP
② MOV EBP, ESP
```

① push ebp : 함수가 종료된 후 ebp를 이전 함수의 ebp로 재설정하기 위해 스택에 이전 함수의 ebp를 push 한다.

② mov ebp, esp : 호출된 함수의 시작을 알리기 위해 현재 esp 값을 ebp에 복사한다.

#11

```
RET
① POP EIP
② JMP EIP
```

① 스택에서 최상위 값을 POP하여 명령어 포인터(EIP)에 저장한다. 반환 주소 바로 뒤에 RET 명령어가 나오지 않을 때 사용한다.

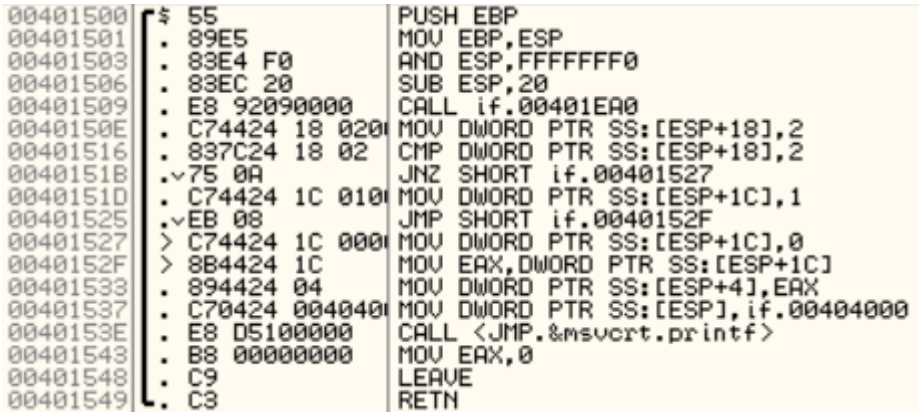
② 명령어 포인터(EIP)에 저장된 특정 주소로 제어 흐름을 전송하는 또 다른 방법

#12

IF문 분석

```
#include <stdio.h>

int main(void) {
  int n, k;
  n = 2;
  if (n == 2)
    k = 1;
  else
    k = 0;
  printf("%d", k);
  return 0;
}
```



C언어 if문 코드를 ollydbg를 통해 분석해봤습니다.

#13

```
00401500: PUSH EBP
00401501: MOV EBP, ESP
00401503: AND ESP, FFFFFFF0
00401506: SUB ESP, 20
```

해당 부분은 앞선 설명드렸던 함수 프롤로그 부분입니다.

C언어 코드에서는 int main(void)부분을 나타내고

EBP 값을 스택에 PUSH하고

ESP 값을 EBP로 이동시킵니다.

ESP 값을 FFFFFFF0과 비트 AND 연산, 스택 포인터를 16바이트 경계로 정렬하는 작업으로

ESP 값이 20만큼 감소시, 스택에 추가 공간 할당합니다.

#14

```
0040150E: MOV DWORD PTR SS:[ESP+18], 2
00401516: CMP DWORD PTR SS:[ESP+18], 2
0040151D: MOV DWORD PTR SS:[ESP+1C], 1
00401525: JMP SHORT if.0040152F
0040152F: MOV EAX, DWORD PTR SS
```

해당 부분은 C언어 코드에서

```
n = 2;
  if (n == 2)
    k = 1;를 나타냅니다
```

ESP+18 위치에 2를 저장하고

ESP+18 위치의 값과 2를 비교합니다.

만약 n의 값이 2라면

ESP+1C 위치에 1을 저장하고

if.0040152F으로 점프합니다.

그런 다음 ESP+1C 위치의 값을 EAX 레지스터에 저장합니다.

#15

```
0040151B: JNZ SHORT if.00401527
00401527: MOV DWORD PTR SS:[ESP+1C], 0
00401543: MOV EAX, 0
```

해당 부분은 C언어 코드에서

```
else
  k = 0;를 나타냅니다.
```

비교 결과가 0이 아닌 경우 즉 n의 값이 2가 아닌 경우 if.00401527으로 점프하고,

ESP+1C 위치에 0을 저장합니다

그런 다음 EAX 레지스터에 0을 저장합니다.

#16

```
00401533: MOV DWORD PTR SS:[ESP+4], EAX
00401537: MOV DWORD PTR SS:[ESP], if.00404000
0040153E: CALL <JMP.&msvcrt.printf>
```

해당 부분은 C언어에서

```
printf("%d", k);를 나타냅니다.
```

ESP+4 위치에 EAX 값을 저장하고

ESP 위치에 if.00404000 값을 저장합니다 해당 부분은 ASCII 문자열 "%d"를 가리킵니다.

그 이후 printf 함수를 호출합니다.

#17

```
00401548: LEAVE
00401549: RETN
```

마지막으로 함수 에필로그입니다.
return 0;을 나타내고

스택 프레임을 복원하고

함수 종료를 나타내는 리턴 명령을 내립니다.

#18

QNA와 발표 마무리 인사