ASSEMBLY LANGUAGE

SCP 이정호

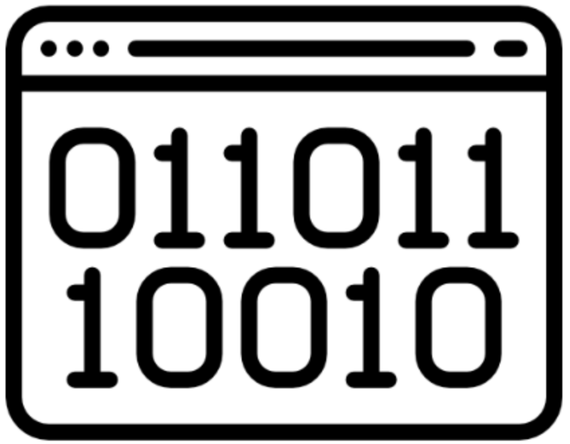
목차

어셈블리어란?

- 컴퓨터와 통신을 위해 사용되는 프로그래밍 언어
- 기계와 일대일 대응이 되는 저급 언어

어셈블리어

기계어



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

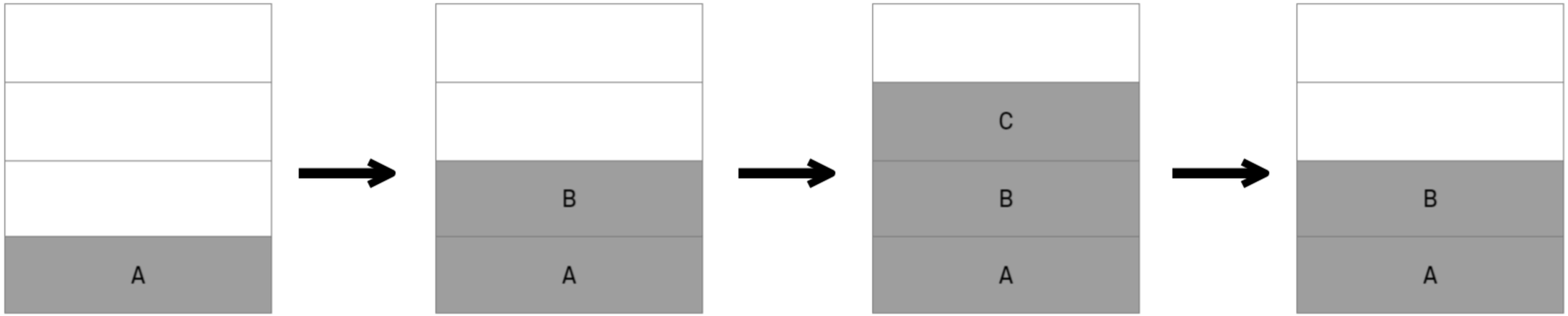
레지스터

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

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

스택

- 데이터 임시 저장을 위한 메모리 공간
- LIFO(후입선출, Last-In-First-Out) 구조



PUSA A → PUSH B → PUSH C → POP C

기본 명령어

산술 연산 명령어

ADD

제1피연산자와 제2피연산자 값을 더한 결과 값을 제1피연산자에 저장

```
EAX = 10
ADD EAX, 5
EAX = EAX(10) + 5 = 15
```

SUB

제1피연산자에서 제2피연사자 값을 뺀 결과 값을 제1피연산자에 저장

```
EAX = 10
SUB EAX, 5
EAX = EAX(10) - 5 = 5
```

기본 명령어

데이터 전송 명령어

MOV

데이터 값을 이동할 때 사용

```
MOV EAX, [EBP+3]
```

LEA

데이터 값을 이동할 때 사용

```
LEA EAX, [EBP+3]
```

제2연산자에 대한 추가 연산 방식이 다르다.

분석

함수 프롤로그

함수가 호출될 때 실행되는 코드

1. PUSH EBP
2. MOV EBP, ESP

1. 함수가 종료된 후 EBP를 이전 함수의 EBP로 재설정하기 위해 스택에 이전 함수의 EBP를 PUSH 한다.
2. 호출된 함수의 시작을 알리기 위해 현재 ESP 값을 EBP에 복사한다.

분석

함수 에필로그

함수가 종료될 때 실행되는 코드

LEAVE

1. **MOV ESP, EBP**
2. **POP EBP**

1. 기본 포인터(EBP)의 값을 스택 포인터(ESP)에 복사한다.
   일반적으로 함수를 종료하기 전에 스택 포인터를 이전 위치로 복원하는 데 사용
2. 스택의 맨 위에서 값을 POP하여 기본 포인터(EBP)에 저장한다.
   이는 함수를 종료하기 전에 기본 포인터의 이전 값을 복원하기 위해 수행한다.

분석

함수 에필로그

함수가 종료될 때 실행되는 코드

RET

1. POP EIP
2. JMP EIP

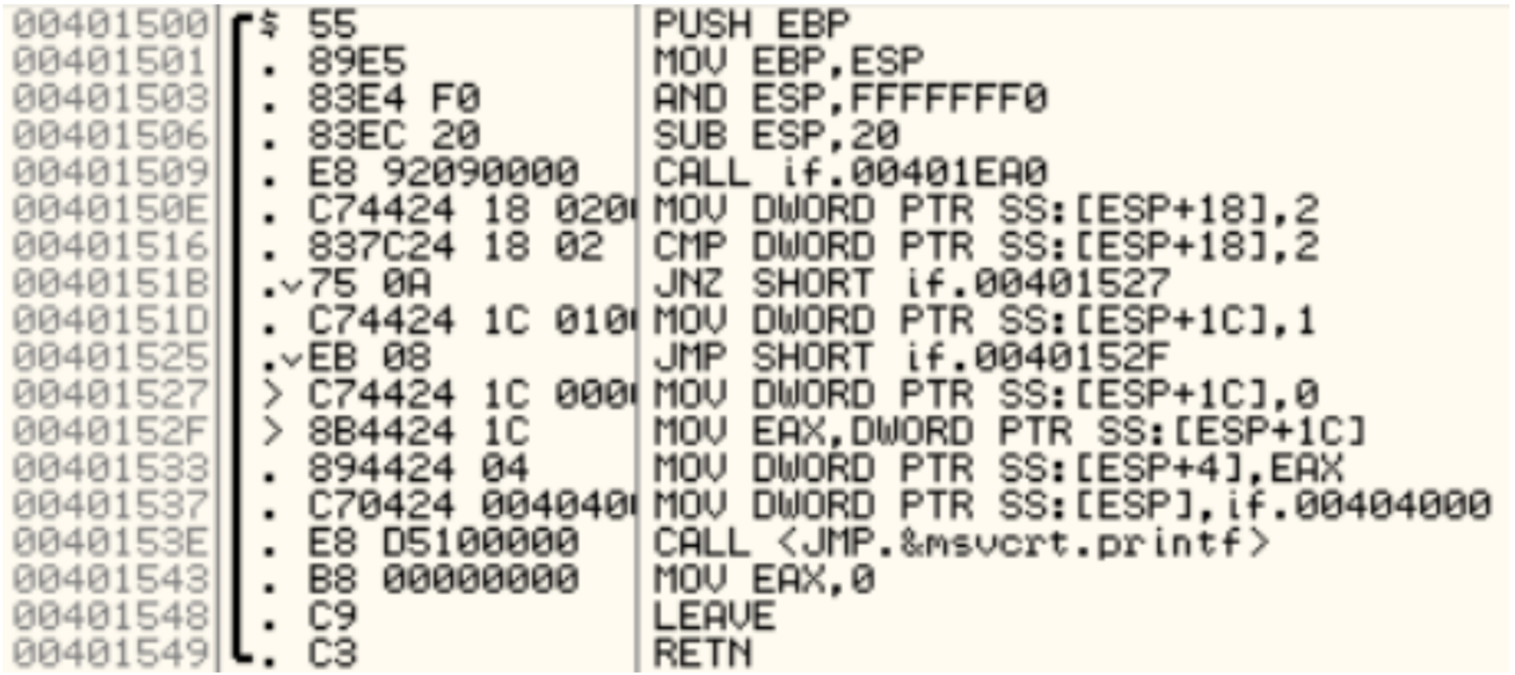
1. 스택에서 최상위 값을 POP하여 명령어 포인터(EIP)에 저장한다.
   반환 주소 바로 뒤에 RET 명령어가 나오지 않을 때 사용한다.
2. 명령어 포인터(EIP)에 저장된 특정 주소로 제어 흐름을 전송하는 또 다른 방법

분석

IF문 분석

```c
#include <stdio.h>

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



```
00401500  ┌$ 55              PUSH EBP
00401501  │. 89E5            MOV EBP,ESP
00401503  │. 83E4 F0         AND ESP,FFFFFFF0
00401506  │. 83EC 20         SUB ESP,20
00401509  │. E8 92090000     CALL if.00401EA0
0040150E  │. C74424 18 020( MOV DWORD PTR SS:[ESP+18],2
00401516  │. 837C24 18 02    CMP DWORD PTR SS:[ESP+18],2
0040151B  │.v75 0A           JNZ SHORT if.00401527
0040151D  │. C74424 1C 010( MOV DWORD PTR SS:[ESP+1C],1
00401525  │.vEB 08           JMP SHORT if.0040152F
00401527  │> C74424 1C 000( MOV DWORD PTR SS:[ESP+1C],0
0040152F  │> 8B4424 1C       MOV EAX,DWORD PTR SS:[ESP+1C]
00401533  │. 894424 04       MOV DWORD PTR SS:[ESP+4],EAX
00401537  │. C70424 004040( MOV DWORD PTR SS:[ESP],if.00404000
0040153E  │. E8 D5100000     CALL <JMP.&msvcrt.printf>
00401543  │. B8 00000000     MOV EAX,0
00401548  │. C9              LEAVE
00401549  └. C3              RETN
```

분석

- IF문 분석

  함수 프롤로그
  int main(void)

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

분석

- IF문 분석

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

```
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
```

분석

- IF문 분석

  else

  k = 0;

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

분석

- IF문 분석
  printf("%d", k);

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

분석

- IF문 분석

  함수 에필로그

  return 0;

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

감사합니다!

Q&A