EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

제 4기

2022. 09. 23

진동민

학습목표 & 3회차 날짜

EDDI
Electronic Design
Development Institute

학습목표

- if와 for가 어셈블리어로 어떻게 구성되어있는지 배우기

수업 날짜

2022-09-03 (토) 오후 6시~9시

목차

(참고)
어셈블리어 분석은 2회차 숙제때 자세히 분석 및 설명했으므로, 이번 숙제부터는 간단히 설명하겠다.
만약 어셈블리어를 보고 바로 해석하지 못한다면 2주차 숙제의 fun.c를 직접 gdb로 분석하는 것을 여러 번 추천한다.

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

1) if.c

```c
#include <stdio.h>

int main(void)
{
    int num = 3;

    if (num > 7)
    {
        printf("3 > 7\n");
    }
    else
    {
        printf("3 <= 7\n");
    }

    return 0;
}
```

carbon
carbon.now.sh

https://carbon.now.sh/ 사용

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

2) 보편적인 if의 형식

1. mov로 비교 대상 배치

1. cmp로 실제 비교 수행

1. 조건부 jmp로 분기
   - le: Less or Equal ← jle
   - ge: Greater or Equal ← jge
   - 비교하는 기준값은 0

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

3) if.c의 어셈블리어

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     sub    $0x10,%rsp
   0x0000555555555155 <+12>:    movl   $0x3,-0x4(%rbp)
   0x000055555555515c <+19>:    cmpl   $0x7,-0x4(%rbp)
   0x0000555555555160 <+23>:    jle    0x555555555170 <main+39>
   0x0000555555555162 <+25>:    lea    0xe9b(%rip),%rdi        # 0x555555556004
   0x0000555555555169 <+32>:    callq  0x555555555050 <puts@plt>
   0x000055555555516e <+37>:    jmp    0x55555555517c <main+51>
   0x0000555555555170 <+39>:    lea    0xe93(%rip),%rdi        # 0x55555555600a
   0x0000555555555177 <+46>:    callq  0x555555555050 <puts@plt>
   0x000055555555517c <+51>:    mov    $0x0,%eax
   0x0000555555555181 <+56>:    leaveq
   0x0000555555555182 <+57>:    retq
End of assembler dump.
```

if.c 어셈블리어 분석

4) 중요 포인트만 분석

16바이트 스택 생성

```
0x0000555555555151 <+8>:     sub    $0x10,%rsp
0x0000555555555155 <+12>:    movl   $0x3,-0x4(%rbp)
0x000055555555515c <+19>:    cmpl   $0x7,-0x4(%rbp)
0x0000555555555160 <+23>:    jle    0x555555555170 <main+39>
0x0000555555555162 <+25>:    lea    0xe9b(%rip),%rdi        # 0x555555556004
0x0000555555555169 <+32>:    callq  0x555555555050 <puts@plt>
0x000055555555516e <+37>:    jmp    0x55555555517c <main+51>
0x0000555555555170 <+39>:    lea    0xe93(%rip),%rdi        # 0x55555555600a
0x0000555555555177 <+46>:    callq  0x555555555050 <puts@plt>
```

비교 대상인 변수 num을 스택에 배치

7과 num을 비교

계산된 값으로 점프할지 말지 결정

문자열 3 > 7의 시작 주소

문자열 3 <= 7의 시작 주소

1. movl $0x3, -0x4(%rbp): if의 조건식에서 비교할 대상인 변수 num을 스택에 배치

1. cmpl $0x7, -0x4(%rbp): 7과 num을 비교함
   - cmp source, destination의 의미는 destination - source
   - 여기서는 3 - 7 이므로 -4

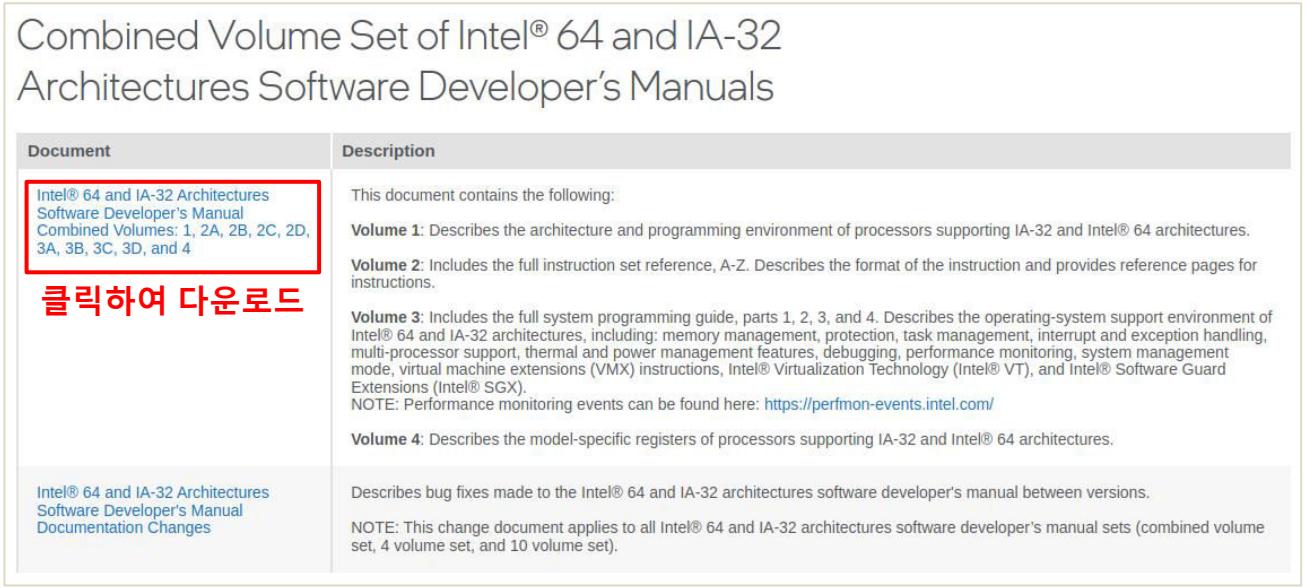
1. jle 0x555555555170: 계산한 값이 0보다 작거나 같으면 5170 위치로 점프
   - -4가 0보다 작으므로 5170으로 점프

1. (5170 위치) lea 명령어로 문자열 3 <= 7의 주소를 rdi에 저장하고, call 명령어를 통해 문자열 3 <= 7을 출력

if.c 어셈블리어 분석

5-1) cmp와 jle 명령어 더 자세히 알아보기

https://cafe.naver.com/eddicorp/918 카페에 접속하여 게시글의 링크로 들어가서 pdf 파일을 다운로드 받는다.



Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals

Document	Description
Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4	This document contains the following: **Volume 1**: Describes the architecture and programming environment of processors supporting IA-32 and Intel® 64 architectures. **Volume 2**: Includes the full instruction set reference, A-Z. Describes the format of the instruction and provides reference pages for instructions. **Volume 3**: Includes the full system programming guide, parts 1, 2, 3, and 4. Describes the operating-system support environment of Intel® 64 and IA-32 architectures, including: memory management, protection, task management, interrupt and exception handling, multi-processor support, thermal and power management features, debugging, performance monitoring, system management mode, virtual machine extensions (VMX) instructions, Intel® Virtualization Technology (Intel® VT), and Intel® Software Guard Extensions (Intel® SGX). NOTE: Performance monitoring events can be found here: https://perfmon-events.intel.com/ **Volume 4**: Describes the model-specific registers of processors supporting IA-32 and Intel® 64 architectures.
Intel® 64 and IA-32 Architectures Software Developer's Manual Documentation Changes	Describes bug fixes made to the Intel® 64 and IA-32 architectures software developer's manual between versions. NOTE: This change document applies to all Intel® 64 and IA-32 architectures software developer's manual sets (combined volume set, 4 volume set, and 10 volume set).

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

5-2) cmp와 jle 명령어 더 자세히 알아보기

p.182

p.189

다운로드 받은 pdf 파일에서 위의 적힌 내용을 찾는다.

추가적으로 p.79의 EFLAGS Register도 살펴볼 것이다.

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

6) cmp 명령어

7.3.2.4 Comparison and Sign Change Instructions

The CMP (compare) instruction computes the difference between two integer operands and updates the OF, SF, ZF, AF, PF, and CF flags according to the result. The source operands are not modified, nor is the result saved. The CMP instruction is commonly used in conjunction with a *Jcc* (jump) or SET*cc* (byte set on condition) instruction, with the latter instructions performing an action based on the result of a CMP instruction.

The NEG (negate) instruction subtracts a signed integer operand from zero. The effect of the NEG instruction is to change the sign of a two's complement operand while keeping its magnitude.

- 해석
  - CMP 명령어는 두 정수 피연산자 간의 차이를 계산하고, (계산) 결과에 따라 OF, SF, ZF, AF, PF, CF 플래그를 갱신합니다.
  - source 피연산자는 수정되지 않으며, 결과도 저장되지 않습니다.
- 정리
  - cmp 명령어는 피연산자를 계산한 결과에 따라 플래그값(eflags 값)을 변경함

if.c 어셈블리어 분석

7) eflags 레지스터

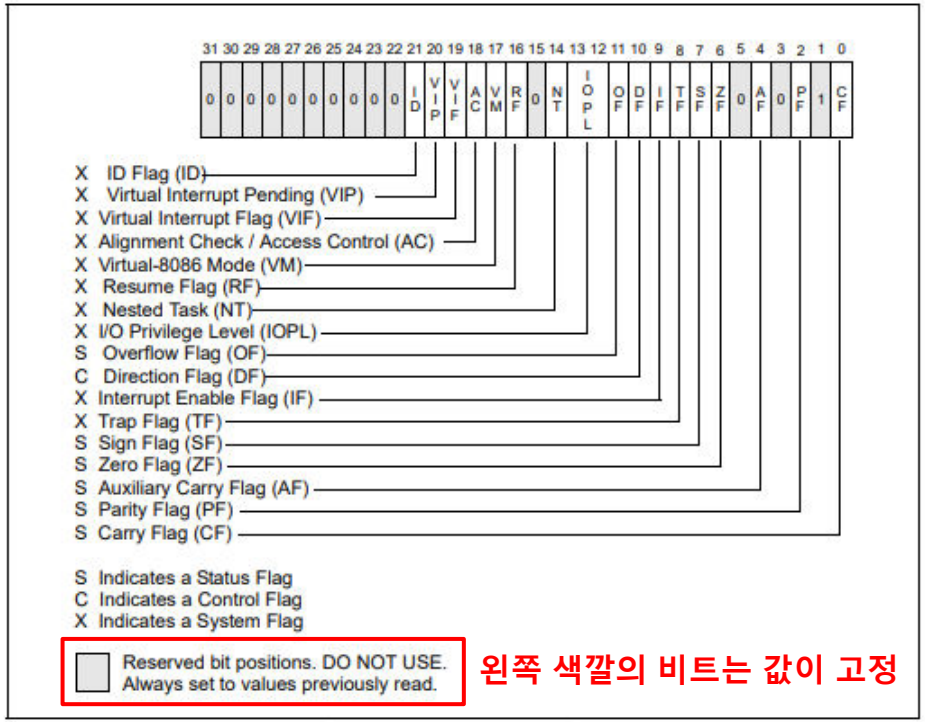


Figure 3-8. EFLAGS Register

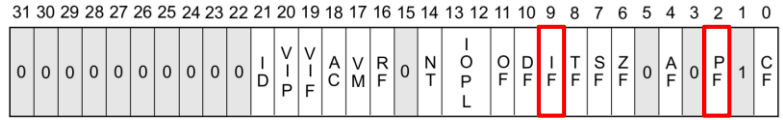
그럼 eflags 레지스터 값을 cmp 명령어를 실행하기 전과 후를 비교해보자

if.c 어셈블리어 분석

8) cmp 명령어 분석

```
eflags          0x206                [ PF IF ]
```

cmp 명령어 실행하기 전



0x206

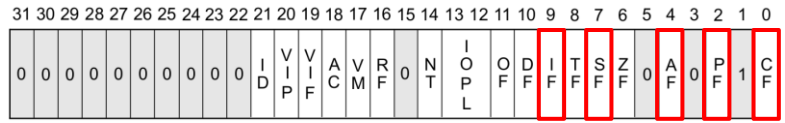
10 0000 0110 (2진수)

```
   0x0000555555555155 <+12>:    movl   $0x3,-0x4(%rbp)
=> 0x000055555555515c <+19>:    cmpl   $0x7,-0x4(%rbp)    실행
   0x0000555555555160 <+23>:    jle    0x555555555170 <main+39>
```

cmp 명령어 실행

```
eflags          0x297                [ CF PF AF SF IF ]
```

cmp 명령어 실행한 후



0x297

10 1001 0111 (2진수)

if.c 어셈블리어 분석

9-1) jle 명령어

7.3.8.2 Conditional Transfer Instructions

The conditional transfer instructions execute jumps or loops that transfer program control to another instruction in the instruction stream if specified conditions are met. The conditions for control transfer are specified with a set of condition codes that define various states of the status flags (CF, ZF, OF, PF, and SF) in the EFLAGS register.

Conditional jump instructions — The *Jcc* (conditional) jump instructions transfer program control to a destination instruction if the conditions specified with the condition code (*cc*) associated with the instruction are satisfied (see Table 7-4). If the condition is not satisfied, execution continues with the instruction following the *Jcc* instruction. As with the JMP instruction, the transfer is one-way; that is, a return address is not saved.

- 해석
  - 제어 전달 조건은 EFLAGS 레지스터의 상태 플래그들(CF, ZF, OF, PF, SF)의 다양한 상태를 정의하는 일련의 조건 코드와 함께 지정된다.

- 정리
  - 조건부 점프는 eflags 레지스터의 상태를 보고 결정한다.

(참고)
조건부 점프: https://terms.naver.com/entry.naver?docId=749801&cid=50324&categoryId=50324

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

9-2) jle 명령어

Table 7-4. Conditional Jump Instructions

Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAE/JNB	CF = 0	Above or equal/not below
JB/JNAE	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero
Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

JLE의 조건은 ((SF xor OF)) or ZF) = 1일 경우 점프한다.

if.c 어셈블리어 분석

10) jle 명령어 분석

cmp 명령어 실행 후, eflags에서 1인 값을 가진 상태 플래그는 CF, PF, AF, SF, IF 이다.
jle 명령어의 조건은 ((SF xor OF)) or ZF) = 1일 경우 점프하는데, 계산 결과 ((1 xor 0)) or 0) = 1 이므로 점프하게 된다.

```
   0x0000555555555155 <+12>:    movl   $0x3,-0x4(%rbp)
   0x000055555555515c <+19>:    cmpl   $0x7,-0x4(%rbp)
=> 0x0000555555555160 <+23>:    jle    0x555555555170 <main+39>
   0x0000555555555162 <+25>:    lea    0xe9b(%rip),%rdi        # 0x555555556004
   0x0000555555555169 <+32>:    callq  0x555555555050 <puts@plt>
   0x000055555555516e <+37>:    jmp    0x55555555517c <main+51>
   0x0000555555555170 <+39>:    lea    0xe93(%rip),%rdi        # 0x55555555600a
   0x0000555555555177 <+46>:    callq  0x555555555050 <puts@plt>
   0x000055555555517c <+51>:    mov    $0x0,%eax
   0x0000555555555181 <+56>:    leaveq
   0x0000555555555182 <+57>:    retq
End of assembler dump.
(gdb) si
13                      printf("3 <= 7\n");
(gdb) disas
Dump of assembler code for function main:
   0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     sub    $0x10,%rsp
   0x0000555555555155 <+12>:    movl   $0x3,-0x4(%rbp)
   0x000055555555515c <+19>:    cmpl   $0x7,-0x4(%rbp)
   0x0000555555555160 <+23>:    jle    0x555555555170 <main+39>
   0x0000555555555162 <+25>:    lea    0xe9b(%rip),%rdi        # 0x555555556004
   0x0000555555555169 <+32>:    callq  0x555555555050 <puts@plt>
   0x000055555555516e <+37>:    jmp    0x55555555517c <main+51>
=> 0x0000555555555170 <+39>:    lea    0xe93(%rip),%rdi        # 0x55555555600a
```

jle 명령어 실행

jle 명령어 실행하기 전

jle 명령어 실행한 후, 점프한 것을 확인

if.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

11) 정리

if문을 어셈블리어로 분석하면, 다음과 같다.

1. 조건문 내에 있는 비교 대상인 변수를 mov 명령어로 스택에 배치한다.
2. cmp 명령어로 비교를 수행하는데, 결과값에 따라 eflags 레지스터 값을 갱신한다.
3. (실습에서 사용했던) jle 명령어는 elfags 레지스터를 참조하여 점프할지 결정한다.

그러므로 사람은 계산한 결과값을 보고 점프하는지 확인하면 되지만, 컴퓨터는 eflags 레지스터 값을 보고 점프할지 결정한다.

for.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

1) for.c

```c
#include <stdio.h>

int main(void)
{
    int i;
    char ch = 'A';

    for (i = 0; i < 26; i++, ch++)
    {
        printf("%c, %c\n", ch, ch ^ 0x20);
    }

    return 0;
}
```

carbon
carbon.now.sh

https://carbon.now.sh/ 사용

for.c 어셈블리어 분석

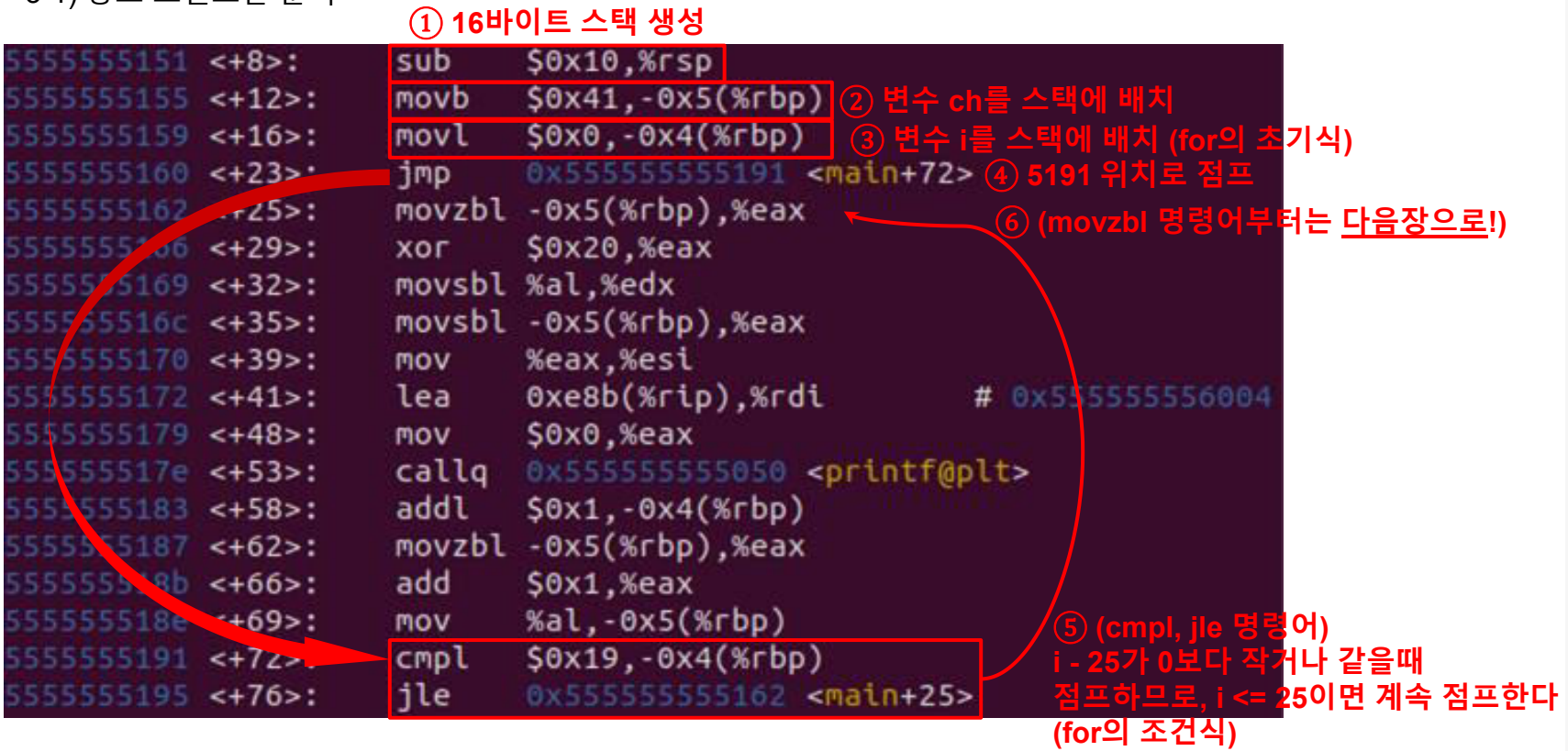
EDDI
Electronic Design Development Institute

2) for.c의 어셈블리어

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     sub    $0x10,%rsp
   0x0000555555555155 <+12>:    movb   $0x41,-0x5(%rbp)
   0x0000555555555159 <+16>:    movl   $0x0,-0x4(%rbp)
   0x0000555555555160 <+23>:    jmp    0x555555555191 <main+72>
   0x0000555555555162 <+25>:    movzbl -0x5(%rbp),%eax
   0x0000555555555166 <+29>:    xor    $0x20,%eax
   0x0000555555555169 <+32>:    movsbl %al,%edx
   0x000055555555516c <+35>:    movsbl -0x5(%rbp),%eax
   0x0000555555555170 <+39>:    mov    %eax,%esi
   0x0000555555555172 <+41>:    lea    0xe8b(%rip),%rdi        # 0x555555556004
   0x0000555555555179 <+48>:    mov    $0x0,%eax
   0x000055555555517e <+53>:    callq  0x555555555050 <printf@plt>
   0x0000555555555183 <+58>:    addl   $0x1,-0x4(%rbp)
   0x0000555555555187 <+62>:    movzbl -0x5(%rbp),%eax
   0x000055555555518b <+66>:    add    $0x1,%eax
   0x000055555555518e <+69>:    mov    %al,-0x5(%rbp)
   0x0000555555555191 <+72>:    cmpl   $0x19,-0x4(%rbp)
   0x0000555555555195 <+76>:    jle    0x555555555162 <main+25>
   0x0000555555555197 <+78>:    mov    $0x0,%eax
   0x000055555555519c <+83>:    leaveq
   0x000055555555519d <+84>:    retq
End of assembler dump.
```

for.c 어셈블리어 분석

3-1) 중요 포인트만 분석



for.c 어셈블리어 분석

3-2) 중요 포인트만 분석

```
5555555151 <+8>:     sub    $0x10,%rsp
5555555155 <+12>:    movb   $0x41,-0x5(%rbp)
5555555159 <+16>:    movl   $0x0,-0x4(%rbp)
5555555160 <+23>:    jmp    0x555555555191 <main+72>
5555555162 <+25>:    movzbl -0x5(%rbp),%eax
5555555166 <+29>:    xor    $0x20,%eax
5555555169 <+32>:    movsbl %al,%edx
555555516c <+35>:    movsbl -0x5(%rbp),%eax
5555555170 <+39>:    mov    %eax,%esi
5555555172 <+41>:    lea    0xe8b(%rip),%rdi        # 0x555555556004
5555555179 <+48>:    mov    $0x0,%eax
555555517e <+53>:    callq  0x555555555050 <printf@plt>
5555555183 <+58>:    addl   $0x1,-0x4(%rbp)
5555555187 <+62>:    movzbl -0x5(%rbp),%eax
555555518b <+66>:    add    $0x1,%eax
555555518e <+69>:    mov    %al,-0x5(%rbp)
5555555191 <+72>:    cmpl   $0x19,-0x4(%rbp)
5555555195 <+76>:    jle    0x555555555162 <main+25>
```

⑥ 변수 ch를 4바이트로 변환하여 eax로 복사

⑦ eax(복사한 ch) 값과 0x20을 xor 연산하여 eax에 저장

⑧ al(gdb에서는 rax) 값을 edx로 복사

⑨ 변수 ch를 4바이트로 변환하여 eax로 복사

⑩ eax 값을 esi에 복사

⑪ 콘솔에 계산한 결과를 출력

⑫ 변수 i를 1 증가

⑬ 변수 ch를 4바이트로 변환하여 eax로 복사

⑭ eax(복사한 ch) 값을 1 증가

⑮ al(gdb에서는 rax) 값을 변수 ch에 복사

for.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

3-3) 보충 설명

어셈블리어단에서 printf 함수를 호출하기 전에 레지스터에 저장된 값은 다음과 같다.

edx에는 ch ^ 0x20이 계산된 결과값이 저장
esi에는 ch 값이 저장

for.c 어셈블리어 분석

4) 간단하게 정리

```
5555555151 <+8>:     sub    $0x10,%rsp
5555555155 <+12>:    movb   $0x41,-0x5(%rbp)
5555555159 <+16>:    movl   $0x0,-0x4(%rbp)        ← for의 초기식
5555555160 <+23>:    jmp    0x555555555191 <main+72>
5555555162 <+25>:    movzbl -0x5(%rbp),%eax        ← for문내의 코드
5555555166 <+29>:    xor    $0x20,%eax
5555555169 <+32>:    movsbl %al,%edx
555555516c <+35>:    movsbl -0x5(%rbp),%eax
5555555170 <+39>:    mov    %eax,%esi
5555555172 <+41>:    lea    0xe8b(%rip),%rdi        # 0x555555556004
5555555179 <+48>:    mov    $0x0,%eax
555555517e <+53>:    callq  0x555555555050 <printf@plt>
5555555183 <+58>:    addl   $0x1,-0x4(%rbp)        ← for의 증감식
5555555187 <+62>:    movzbl -0x5(%rbp),%eax
555555518b <+66>:    add    $0x1,%eax
555555518e <+69>:    mov    %al,-0x5(%rbp)
5555555191 <+72>:    cmpl   $0x19,-0x4(%rbp)       ← for의 조건식
5555555195 <+76>:    jle    0x555555555162 <main+25>
```

for.c 어셈블리어 분석

EDDI
Electronic Design Development Institute

5) 자세히 정리

어셈블리어단에서 1바이트 변수 ch를 4바이트로 변환시켰다. 그 이유를 알아보자.

바이트 변환을 왜 하는가?
비트 연산은 반드시 동일 데이터타입끼리 가능
정수형 기본은 int

그러므로 비트 연산을 하기 위해 기본 정수형인 int 4바이트로 변환시켰다.

for.c 어셈블리어 분석

6) ch ^ 0x20 보충 설명

알파벳 A는 아스키 코드 10진수로 65 ← 64 + 1
알파벳 a는 아스키 코드 10진수로 97 ← 64 + 32 +1

‘A’		1	0	0		0	0	0	1
‘a’		1	1	0		0	0	0	1

입력		출력
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR 진리표

같으면 0, 다르면 1

	1	0	0		0	0	0	1		65	A
^	0	1	0		0	0	0	0		32	0x20
					⬇						
	1	1	0		0	0	0	1		97	a
^	0	1	0		0	0	0	0		32	0x20
					⬇						
	1	0	0		0	0	0	1		65	A

결론은 알파벳에 0x20을 xor 연산하면 대문자는 소문자로, 소문자는 대문자로 바뀐다.

기억나지 않지만 메모한 수업내용

수업에서 강사님이 설명했지만 기억나지 않는 내용을 노트북에 간단하게 메모한 내용이 있어 아래에 작성해보겠다.

가상메모리 크기는 4kb

빠른 검색을 위해 가상메모리를 사용함
이유는 데이터가 대규모이기 때문에

끝..

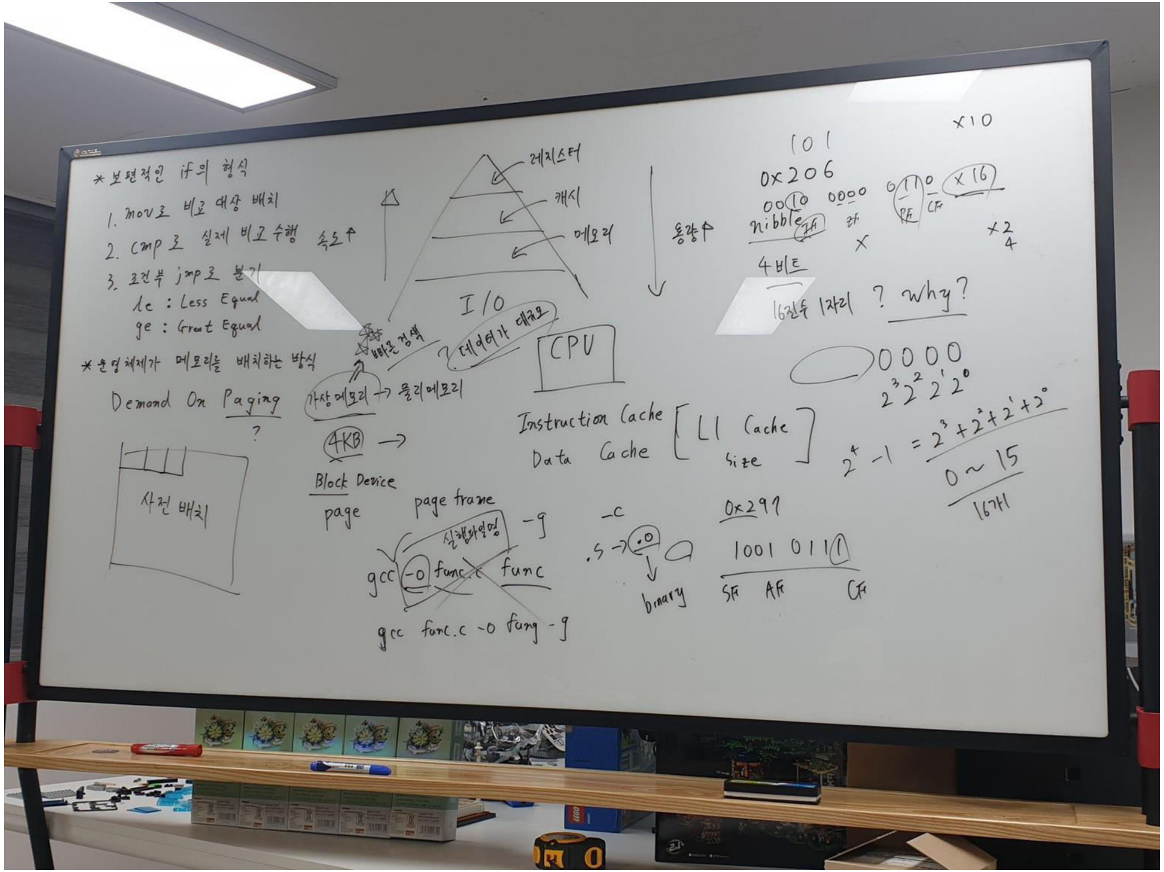
결론

EDDI
Electronic Design
Development Institute

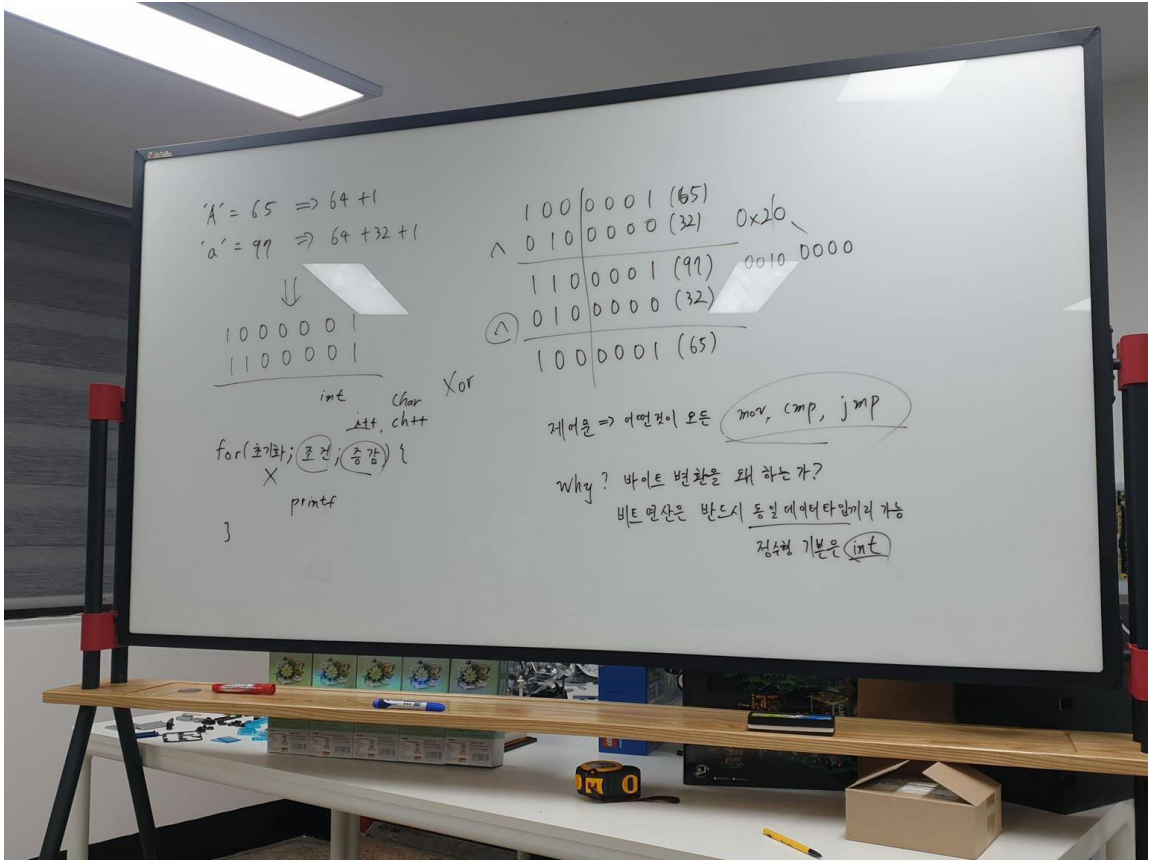
조건식이 쓰이는 곳에는 mov, cmp, jmp 세 명령어를 사용한다.

수업내용 사진

EDDI
Electronic Design
Development Institute



수업내용 사진


'A' = 65 ⇒ 64 +1
'a' = 97 ⇒ 64 +32 +1
1000001
1100001
int
Char
Xor
Att, ch++
for(초기화;조건;증감) {
X
printf
}
100 0001 (65)
^ 010 0000 (32)
0x20
0010 0000
110 0001 (97)
010 0000 (32)
100 0001 (65)
제어문 ⇒ 어떤것이 오든 mov, cmp, jmp
Why? 바이트 변환을 왜 하는가?
비트연산은 반드시 동일 데이터타입끼리 가능
정수형 기본은 int