[Chap.3-1] Machine-level Representation of Programs

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

SKK UNIVERSITY

Contents

- Introduction
- Program encodings
- Data formats
- Intel processors
- Accessing information
- Primitive instructions
- Data movement instructions
- Arithmetic and logic instructions
- Control instructions
- Procedures
- …

Introduction

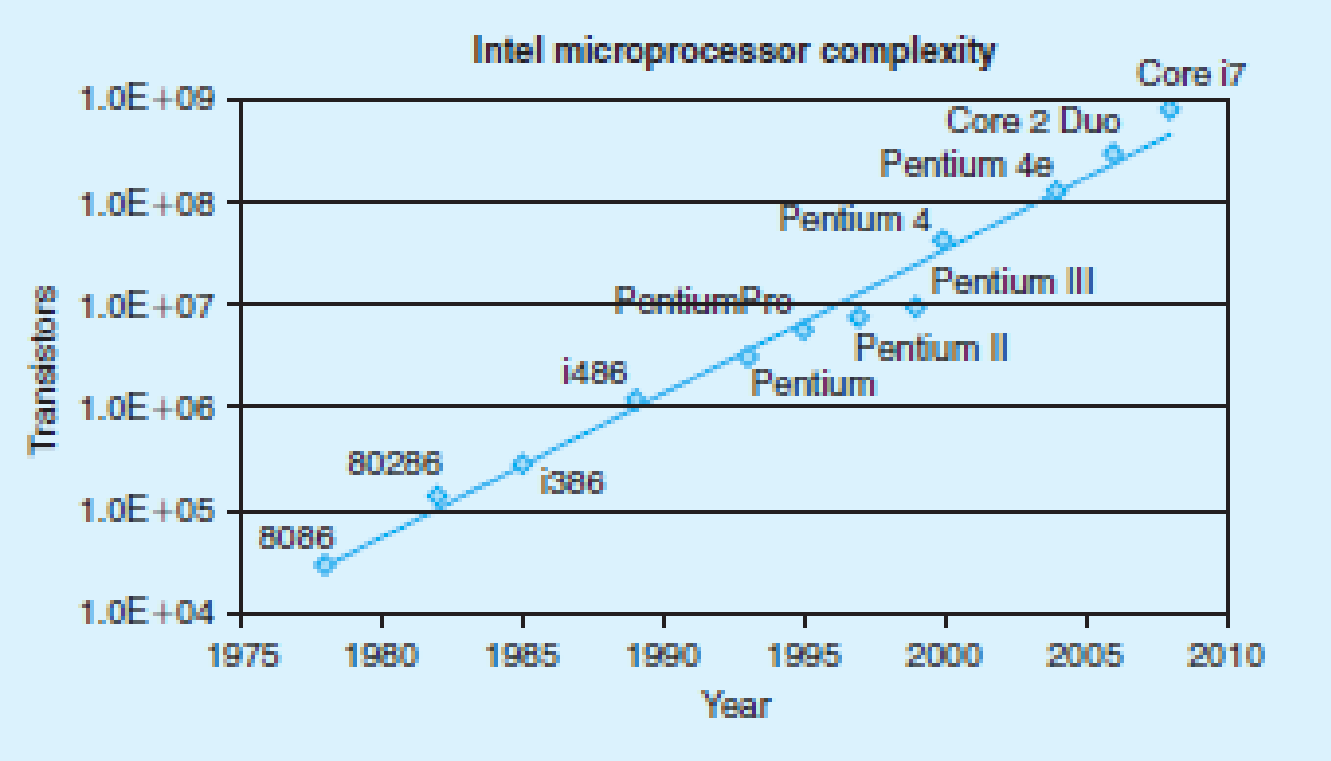
Assembly code

- By reading assembly code, we can
  - Understand the optimization capabilities of the compiler
    - ✓ Analyze the underlying inefficiencies in the code
  - Understand how several vulnerabilities arise, and
    - ✓ How to guard against them

- C language
- x86-64 (and IA32)
- Linux and GCC

Introduction

Models of Intel processors



Introduction

Models of Intel processors

1978	8086	x86 is born
1980	8087	x87 is born
1985	i386	IA32
1995	Pentium Pro	PAE
1997	Pentium MMX	MMX
1999	Pentium III	SSE
2000	Pentium 4	SSE2
2004	Pentium 4E	Hyperthreading, 64-bit extension of IA32
2005	Pentium 4 662	Intel VT
2006	Core 2	SSE4
2008	Core i7 Nehalem	Hyperthreading + Multicore
2011	Core i7 Sandy Bridge	AVX(extension of SSE), Support of 256-bit vectors
2013	Core i7 Haswell	AVX2

More details in Chap. 3-2

Introduction

IA32

- 32-bit machine
- Support 4GB(2^{32}B) of memory

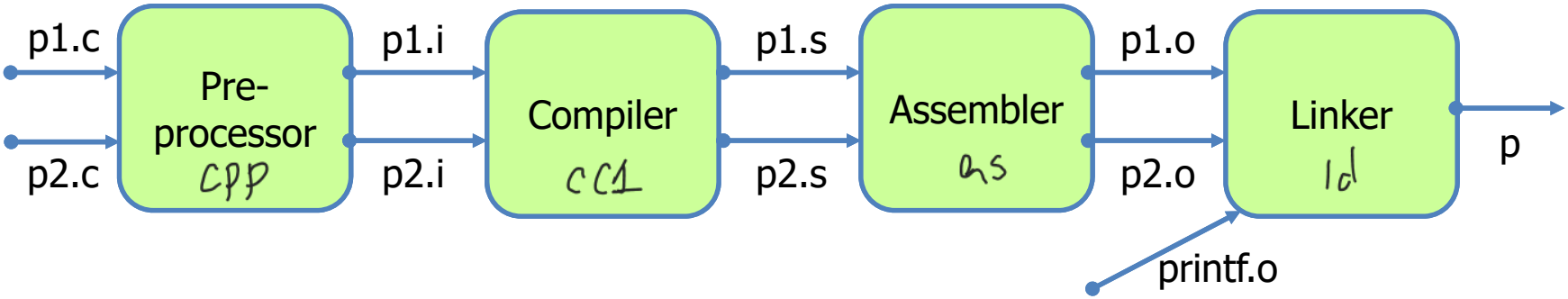
x86-64

- 64-bit machine
- Most of the processors found in today's laptop and desktop machines as well as in supercomputers
- Supports 256TB(2^{48}B) of memory and could be extended upto 16EB(2^{64}B) of memory

Program Encodings

Compilation system

```
linux> gcc -Og -o p p1.c p2.c
```



Program Encodings

Compilation system

```
linux> gcc -Og -o p p1.c p2.c
```

- Option -Og
  - Level of optimization that yields machine code that follows the overall structure of the original source code
  - Higher level of optimizations
    - ✓ Level-1 optimization (-O1)
    - ✓ Level-2 optimization (-O2)

Program Encodings

Compilation system

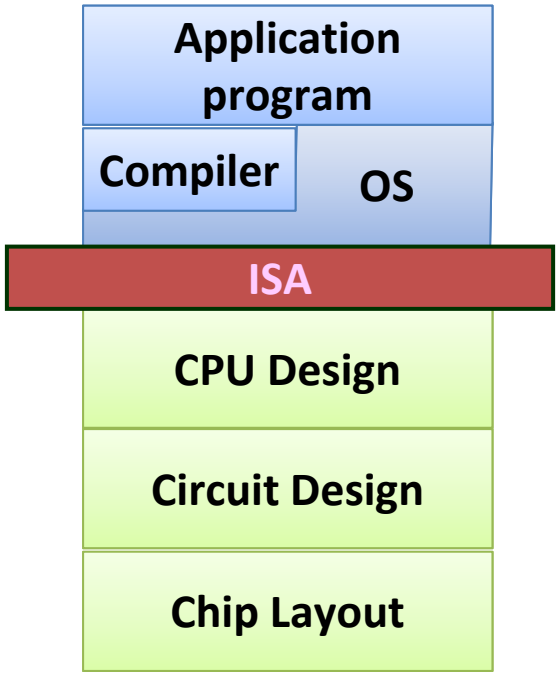
```
linux> gcc -Og -o p p1.c p2.c
```

- Increasing the level of optimization
  - Final executable runs faster
  - Difficult to understand the relationship between the source code and the generated machine (assembly) code
  - Increased compilation time and difficulties in running debugging tools

Program Encodings

ISA (Instruction Set Architecture)

- Defines the format and behavior of a machine-level program
  - Above: how to program machine
    - ✓ Processors execute instructions in sequence
  - Below: what needs to be built
    - ✓ Use variety of tricks to make it run fast

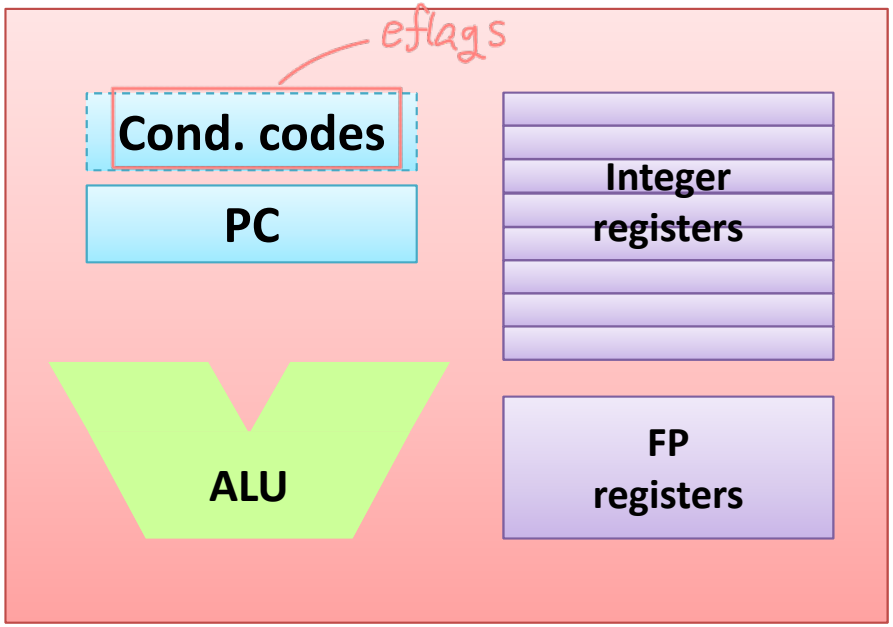


- Instruction set
- Processor registers
- Memory addressing
- Internal representations of data
- Etc…

Program Encodings

Processor (CPU) state

- PC (Program Counter)
  - Address of the next instruction to be executed
  - Register **%rip** in x86-64
- Integer registers
  - Holds integer data or addresses
- Condition code register
  - Store status information on most recent arithmetic or logical instruction
  - Used for conditional changes in control or data flow
- FP registers



Program Encodings

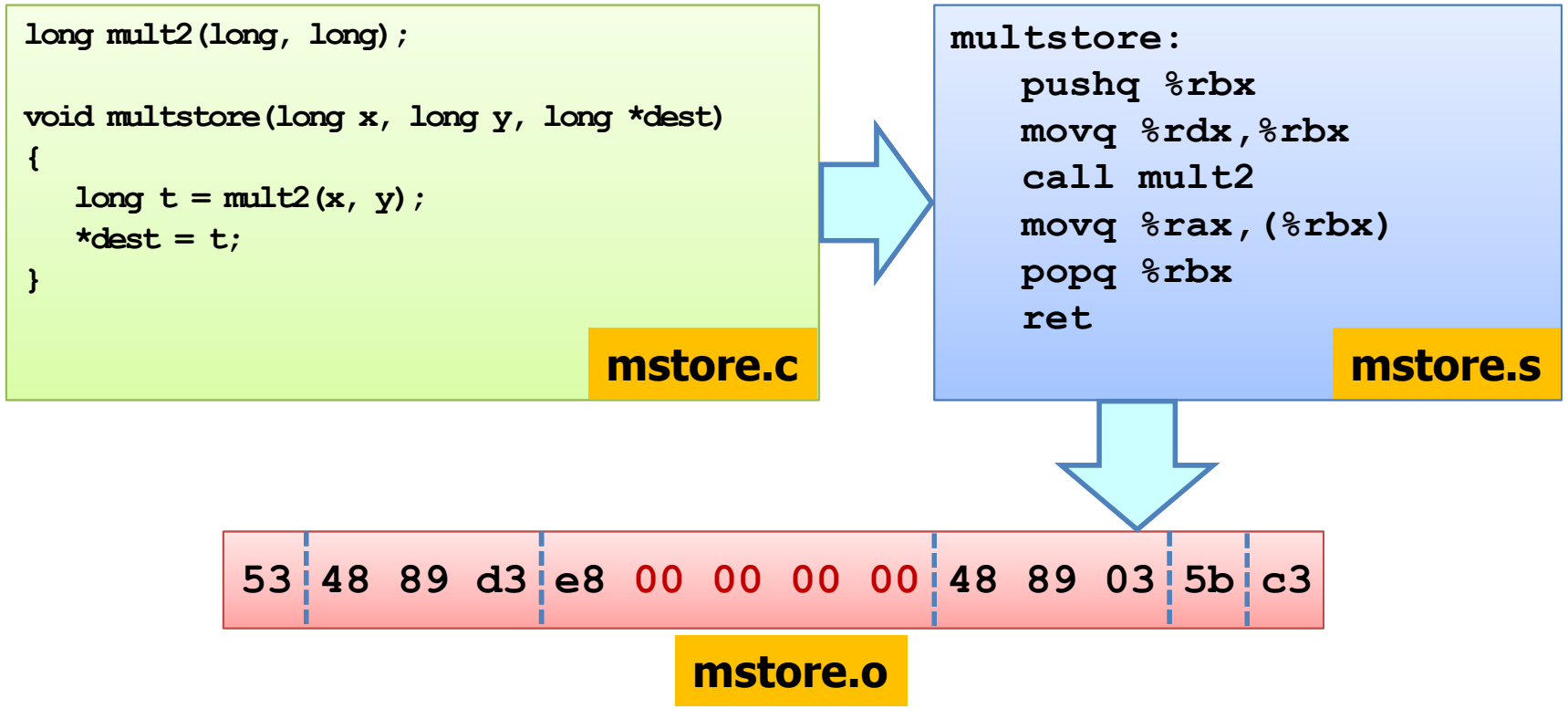
Machine-level code: Some comments

- Data types
  - Several different data types and objects in C
  - No data types in machine code
- Machine instruction
  - Performs only a very elementary operation
    - ✓ Add 2 numbers in registers
    - ✓ Transfer data between memory and register
    - ✓ Conditionally branch to a new instruction address
    - ✓ Etc…
- Program memory
  - Addressed using virtual addresses
  - MMU performs VA-to-PA mapping

Memory Management Unit

Program Encodings

- Code examples (compiling and assembling)
  - linux> gcc –Og [–S] [-c] mstore.c

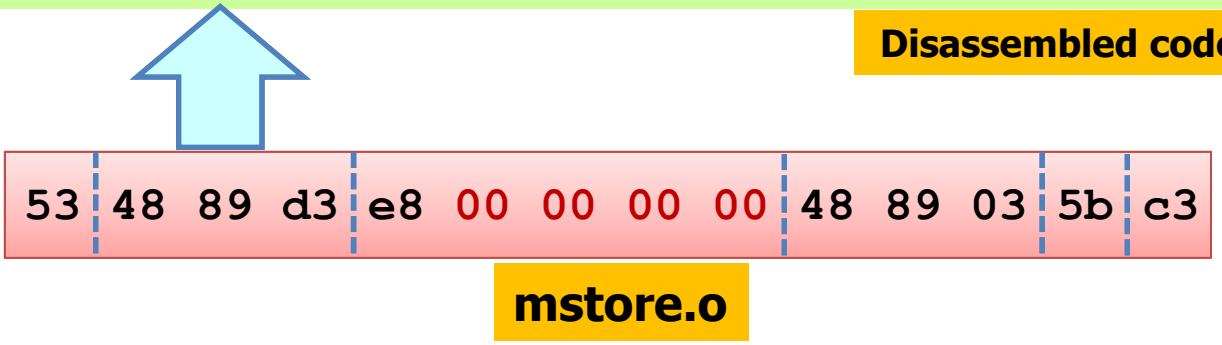


Program Encodings

- Code examples (disassembling)
  - linux> objdump –d mstore.o

```
   Disassembly of function multstore in binary file mstore.o
1  0000000000000000 <multstore>:
   Offset    Bytes                    Equivalent assembly language
2  0:        53                       push %rbx
3  1:        48 89 d3                 mov %rdx,%rbx
4  4:        e8 00 00 00 00           callq 9 <multstore+0x9>
5  9:        48 89 03                 mov %rax,(%rbx)
6  c:        5b                       pop %rbx
7  d:        c3                       retq
```

Disassembled code of mstore.o



Program Encodings

Code examples (linking)

- **linux> gcc –Og –o prog main.c mstore.c**

```
long mult2(long, long);

void multstore(long x, long y, long *dest)
{
   long t = mult2(x, y);
   *dest = t;
}
```

mstore.c

```
#include <stdio.h>

void multstore(long, long, long *);

int main() {
   long d;
   multstore(2, 3, &d);
   printf("2 * 3 --> %ld\n", d);
   return 0;
}

long mult2(long a, long b) {
   long s = a * b;
   return s;
}
```

main.c

Program Encodings

- Code examples (linking)
  - linux> gcc –Og –o prog main.c mstore.c
  - linux> objdump –d prog

```
  Disassembly of function multstore in binary file prog
1 0000000000400540 <multstore>:
  Offset    Bytes                    Equivalent assembly language
2 400540:   53                       push %rbx
3 400541:   48 89 d3                 mov %rdx,%rbx
4 400544:   e8 42 00 00 00           callq 40058b <mult2>
5 400549:   48 89 03                 mov %rax,(%rbx)
6 40054c:   5b                       pop %rbx
7 40054d:   c3                       retq
8 40054e:   90                       nop
9 40054f:   90                       nop
```

Disassembled code of prog

Data Formats

- **In Intel's term**
  - Word: 16-bit data
  - Double word: 32-bit data
  - Quad word: 64-bit data
- **C data types in x86-64**

C declaration	Intel data type	Assembly code suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8

```
movb
movw
movl
movq
```

Summary