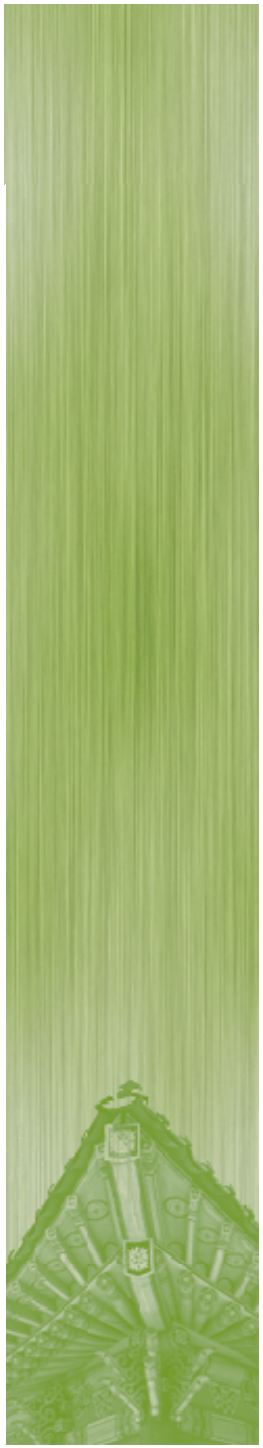


# Machine-level Representation of Programs

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

# Program?

**짬뽕라면**

**준비시간** :10분, **조리시간** :10분

**재료**
라면 1개, 스프 1봉지, 오징어 1/4 마리, 호박 1/4개, 양파 1/2개, 양배추 1장, 당근 1/4개, 물 3컵 (600cc)

**Data**

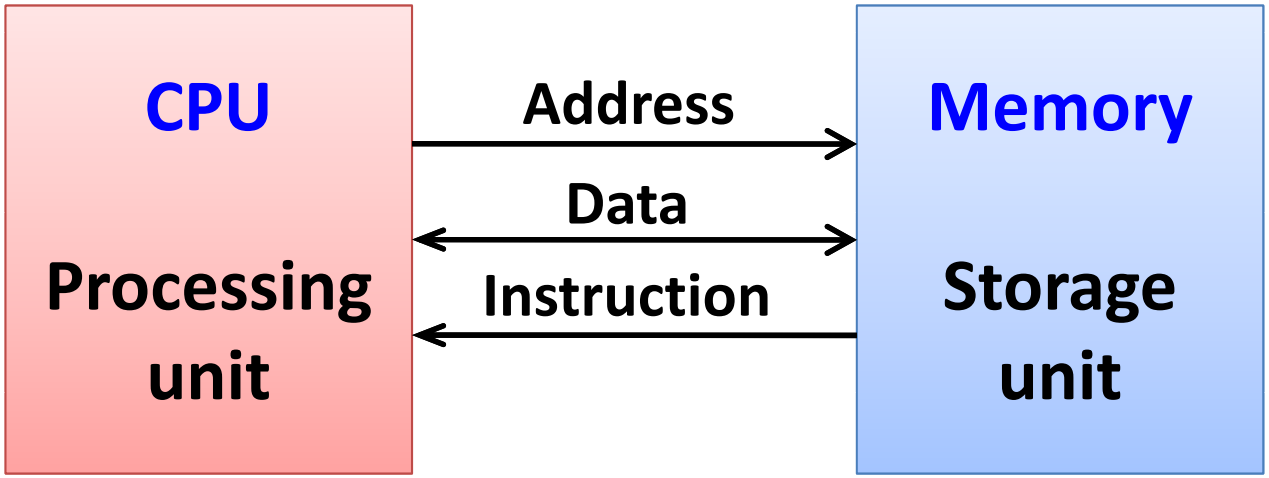
**만드는 법**
1. 오징어는 껍질을 벗기고 깨끗하게 씻어 칼집으로 모양을 낸다.
2. 호박, 양파, 양배추는 모두 채썬다.
3. 냄비에 물 3컵을 붓고 끓인다.
4. 물이 끓으면 스프를 넣고 오징어와 야채를 넣어 충분히 맛이 우러나도록 5분 정도 끓여준다.
5. 끓으면 면을 넣어 익힌다.

**Instructions**

*Source: http://user.chol.com/~yugenie/yo/jjambong.html*

# Introduction (1)

- Computer system

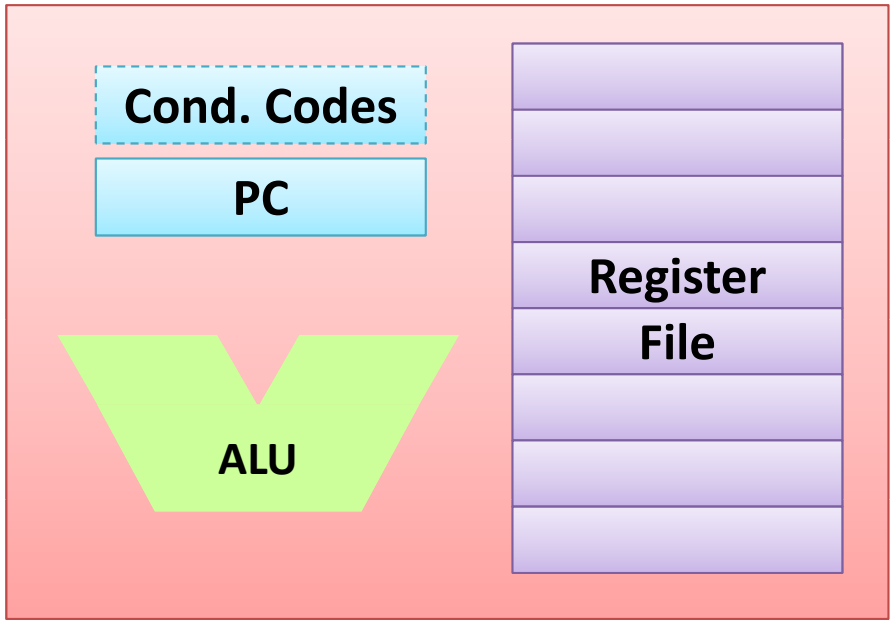



Data movement
Arithmetic & logical ops
Control transfer

Byte addressable array
Program code + data
OS code + data, ...

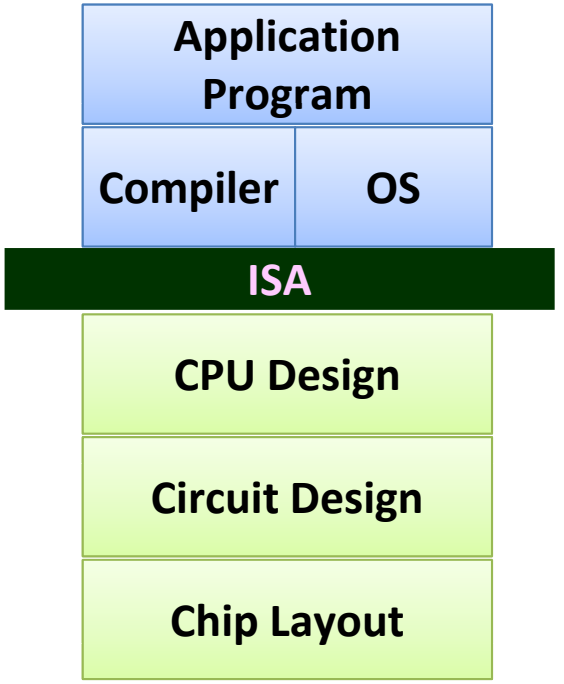
# Introduction (2)

- **CPU (Central Processing Unit)**
  - PC (Program Counter)
    - Address of next instruction
    - Called "EIP" (IA-32) or "RIP" (x86-64)
  - Register File
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching



# Introduction (3)

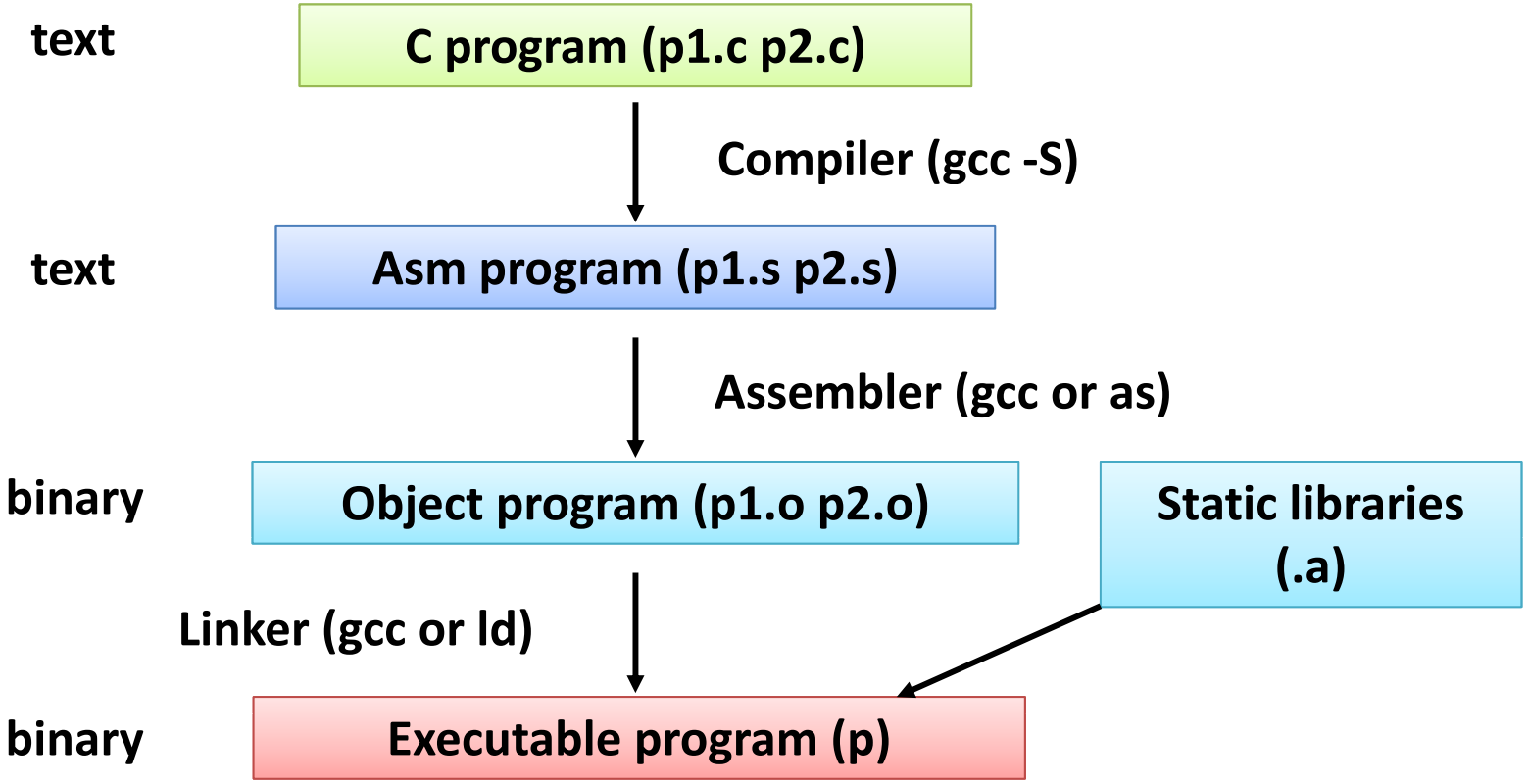
- **Instruction Set Architecture (ISA)**
  - An important part of "Architecture"
  - Above: how to program machine
    - Processors execute instructions in sequence
  - Below: what needs to be built
    - Use variety of tricks to make it run fast




  - Instruction set
  - Processor registers
  - Memory addressing modes
  - Data types and representations
  - …

# Turning C into Object code

- **gcc –O p1.c p2.c –o p**
  - Use optimizations (-O)
  - Put resulting binary in file **p**



# Compiling into Assembly

- gcc –O –S sum.c

**sum.c**

```
int sum(int x, int y)
{
  int t = x + y;
  return t;
}
```

**sum.S**

```
_sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Some compilers use single instruction “leave”

```
0x401040 <sum>:
        0x55 0x89 0xe5 0x8b 0x45 0x0c 0x03 0x45
        0x08 0x89 0xec 0x5d 0xc3
```

# Object Code

- **Assembler**
  - Translates **.s** into **.o**
  - Binary encoding of each instruction
  - Nearly-complete image of executable code
  - Missing linkages between code in different files
- **Linker**
  - Resolves references between files
  - Combines with static run-time libraries
    - e.g., code for malloc(), printf(), etc.
  - Some libraries are dynamically linked
    - Linking occurs when program begins execution.

```
0x401040 <sum>:
0x55
0x89
0xe5
0x8b
0x45
0x0c
0x03
0x45
0x08
0x89
0xec
0x5d
0xc3
```

- **Total of 13 bytes**
- **Each instruction 1, 2, or 3 bytes**
- **Starts at address 0x401040**

# Machine Code Example

- **C code**
  - Add two signed integers

```
int t = x + y;
```

- **Assembly**
  - Add two 4-byte integers
    - "Long" words in GCC parlance
    - Same instruction whether signed or unsigned
  - Operands
    - **x**: Register **%eax**
    - **y**: Memory **M[%ebp+8]**
    - **t**: Register **%eax**

```
addl 8(%ebp),%eax
```

- **Object code**
  - 3-byte instruction
  - Stored at address 0x401046

```
0x401046: 03 45 08
```

# Disassembling (1)

- **Disassembler: objdump –d sum**
  - Useful tool for examining object code
  - Analyzes bit pattern of series of instructions
  - Produces approximate rendition of assembly code
  - Can be run on either **a.out** (complete executable) or **.o** (object code) file.

```
00401040 <_sum>:
   0:      55                   push   %ebp
   1:      89 e5                mov    %esp,%ebp
   3:      8b 45 0c             mov    0xc(%ebp),%eax
   6:      03 45 08             add    0x8(%ebp),%eax
   9:      89 ec                mov    %ebp,%esp
   b:      5d                   pop    %ebp
   c:      c3                   ret
```

# Disassembling (2)

- Using gdb (GNU debugger)

```
$ gdb sum
(gdb) disassemble sum
Dump of assembler code for function sum:
0x401040 <sum>:      push %ebp
0x401041 <sum+1>:    mov  %esp,%ebp
0x401043 <sum+3>:    mov  0xc(%ebp),%eax
0x401046 <sum+6>:    add  0x8(%ebp),%eax
0x401049 <sum+9>:    mov  %ebp,%esp
0x40104b <sum+11>:   pop  %ebp
0x40104c <sum+12>:   ret
```

✓ Disassemble procedure, sum

```
$ gdb sum
(gdb) x/13b sum
0x401040:
  0x55  0x89
  0xe5  0x8b
  0x45  0x0c
  0x03  0x45
  0x08  0x89
  0xec  0x5d
  0xc3
```

✓ Examine 13 bytes starting at sum