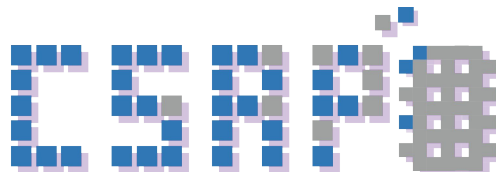


Computer Architecture Lab Session

3. Processor Lab Hints

2019/04/24

comparch@csap.snu.ac.kr



Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

Overview



- Goal
 - Learn about the design and implementation of a pipelined Y86-64 ISA
 - Understand the structure and responsibility of the assembler






0. Getting Started






Fork Repository


- Fork from the repository `comparchTA/Processor Lab`

comparchTA > Processor Lab > Details

**Processor Lab** 
Project ID: 66


  Star 0  Fork 1  Clone 




 Add license  4 Commits  1 Branch  0 Tags  922 KB Files

master 

 processor-lab /

+



History  Find file  Web IDE 

 Document Modified (Changed submission dates)
Hyunik Kim authored 4 days agofc1ccf62 

 Add README

 Add CHANGELOG

 Add CONTRIBUTING

 Enable Auto DevOps

 Set up CI/CD

Fork Repository

- You will have your project on `https://git.csap.snu.ac.kr/<your_id>/processor-lab`

CSAP

Projects ▾ Groups ▾ More ▾ + ▾ Search or jump to... 🔍

📁 🔗 📄 ? ▾ 🌐 ▾

Projects

New project

Your projects 2 Starred projects 0 Explore projects Filter by name... Last updated ▾

All Personal

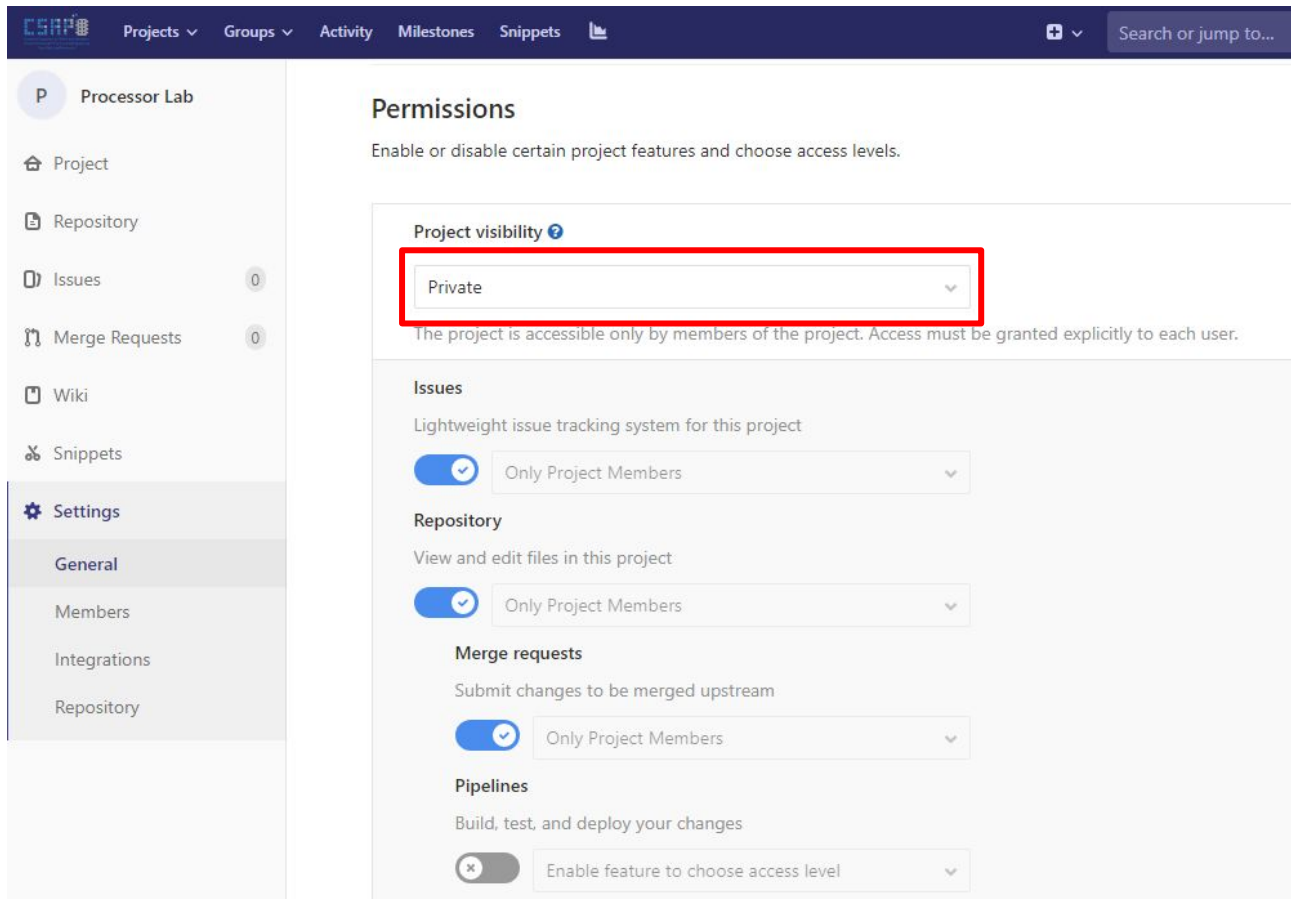
P Hyunik Kim / Processor Lab 🔒 Maintainer

★ 0

Updated 4 days ago

Fork Repository


- Make sure that your repository is **PRIVATE**
(Settings > General > Permissions > Project visibility)



Clone Repository

- Go to your project webpage, and press Clone
- Copy the HTTPS URL to clipboard

Hyunik Kim > Processor Lab > Details




Processor Lab
Project ID: 72

[Add license](#) [0 Commits](#) [1 Branch](#) [0 Tags](#) [0 Bytes Files](#)

Forked from [comparchTA](#) / [Processor Lab](#)

master

processor-lab / +

**Document Modified (Changed submission dates)**
Hyunik Kim authored 4 days ago

[Add README](#)

[Add CHANGELOG](#)

[Add CONTRIBUTING](#)

[Auto DevOps enabled](#)

| Name | Last commit | Last update |
|--------|--|-------------|
| parta | Initial commit | 4 days ago |
| report | Document Modified (Changed submission dates) | 4 days ago |
| sim | Initial commit | 4 days ago |

Clone with SSH

git@git.csap.snu.ac.kr:2019-...

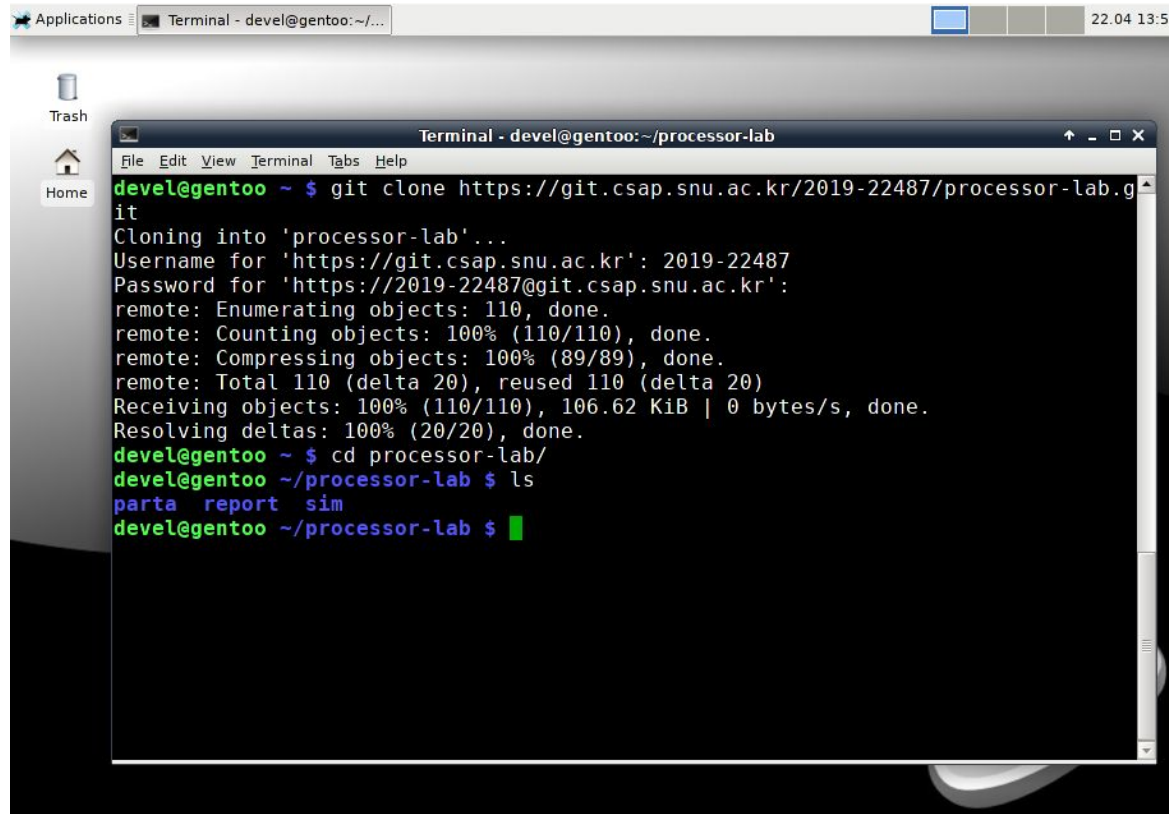
Clone with HTTPS

https://git.csap.snu.ac.kr/2...

Copy URL to clipboard

Clone Repository

- On your terminal in VM, clone repository as follows:
 - `git clone https://<URL>`



The screenshot shows a terminal window titled "Terminal - devel@gentoo:~/processor-lab" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal output shows the execution of the `git clone` command to clone a repository from `https://git.csap.snu.ac.kr/2019-22487/processor-lab.git`. The process includes authentication prompts for username and password, and progress reports for enumerating, counting, and compressing objects. After cloning, the user navigates to the `processor-lab` directory and lists the files, which are `parta`, `report`, and `sim`.

```
devel@gentoo ~ $ git clone https://git.csap.snu.ac.kr/2019-22487/processor-lab.git
Cloning into 'processor-lab'...
Username for 'https://git.csap.snu.ac.kr': 2019-22487
Password for 'https://2019-22487@git.csap.snu.ac.kr':
remote: Enumerating objects: 110, done.
remote: Counting objects: 100% (110/110), done.
remote: Compressing objects: 100% (89/89), done.
remote: Total 110 (delta 20), reused 110 (delta 20)
Receiving objects: 100% (110/110), 106.62 KiB | 0 bytes/s, done.
Resolving deltas: 100% (20/20), done.
devel@gentoo ~ $ cd processor-lab/
devel@gentoo ~/processor-lab $ ls
parta  report  sim
devel@gentoo ~/processor-lab $
```


1. Part A

Part A

- Work in directory `processor-lab/parta` in this part
 - Write and simulate Y86 programs.
 - See `processor-lab/misc/examples.c`
 - Follow x86-64 conventions
 - stack frame structure
 - Registers usage (callee-save & caller-save)
 - Programs:
 - `sum.js`
 - `rsum.js`
 - `copy.js`

Part A

- `sum.ys`: iteratively sum linked list non-zero elements.
- What should you do?
 - Set up the stack structure.
 - Invoke a function `sum_list` & Compute result.
 - Halt.

```
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
        val += ls->val;
        ls = ls->next;
    }
    return val;
}
```

Part A

- `rsum.ys`: **recursively** sum linked list non-zero elements.
- What should you do?
 - Set up the stack structure.
 - Invoke a function
 - `rsum_list`
 - Halt.

```
long rsum_list(list_ptr ls)
{
    if (!ls)
        return 0;
    else {
        long val = ls->val;
        long rest = rsum_list(ls->next);
        return val + rest;
    }
}
```

Part A

- `copy.y`: copies a **block** of words from a region in memory to another.
 - Assume that both regions are not overlapping.
- What should you do?
 - Set up the stack structure.
 - Invoke a function (`copy_block`)
 - Copy from X to Y.
 - Compute Checksum
 - Xor
 - Halt.

```
/* copy_block - Copy src to dest and return xor c
long copy_block(long *src, long *dest, long len)
{
    long result = 0;
    while (len > 0) {
        long val = *src++;
        *dest++ = val;
        result ^= val;
        len--;
    }
    return result;
}
```

Hints for Part A

- How to assemble and simulate?
 - Compile:
 - `misc$./yas sum.y`**s**
 - Simulate:
 - `misc$./yis sum.y`**o**

Hints for Part A

```
# Initial code
    irmovq Stack,%rsp
    irmovq ele1,%rdi
    call sum_list
    halt

# Sample linked list
.align 8
ele1:
    .quad 0x00a
    .quad ele2
ele2:
    .quad 0x0b0
    .quad ele3
ele3:
    .quad 0xc00
    .quad 0

# long sum_list(list_ptr ls)
# ls in %rdi
sum_list:
    xorq %rax,%rax           # val = 0
    andq %rdi,%rdi           # ls == 0?
    je done                  # Yes, goto done
loop:  mrmovq (%rdi),%r10     # t = ls->val
    mrmovq 8(%rdi),%rdi       # ls = ls->next
    addq %r10,%rax            # val += t
    andq %rdi,%rdi           # Check ls
    jne loop                 # If null, goto done
done:  ret                   # return

.pos 0x100
Stack:
```

```
/* linked list element */
typedef struct ELE {
    long val;
    struct ELE *next;
} *list_ptr;

/* sum_list - Sum the elements of a linked list */
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
        val += ls->val;
        ls = ls->next;
    }
    return val;
}
```

Hints for Part A

```
Stopped in 23 steps at PC = 0x1d.  Status 'HLT', CC Z=1 S=0 O=0
```

```
Changes to registers:
```

```
%rax:  0x0000000000000000      0x00000000000000cba
```

```
%rsp:  0x0000000000000000      0x00000000000000100
```

```
%r10:  0x0000000000000000      0x00000000000000c00
```

```
Changes to memory:
```

```
0x00f8: 0x0000000000000000      0x0000000000000001d
```

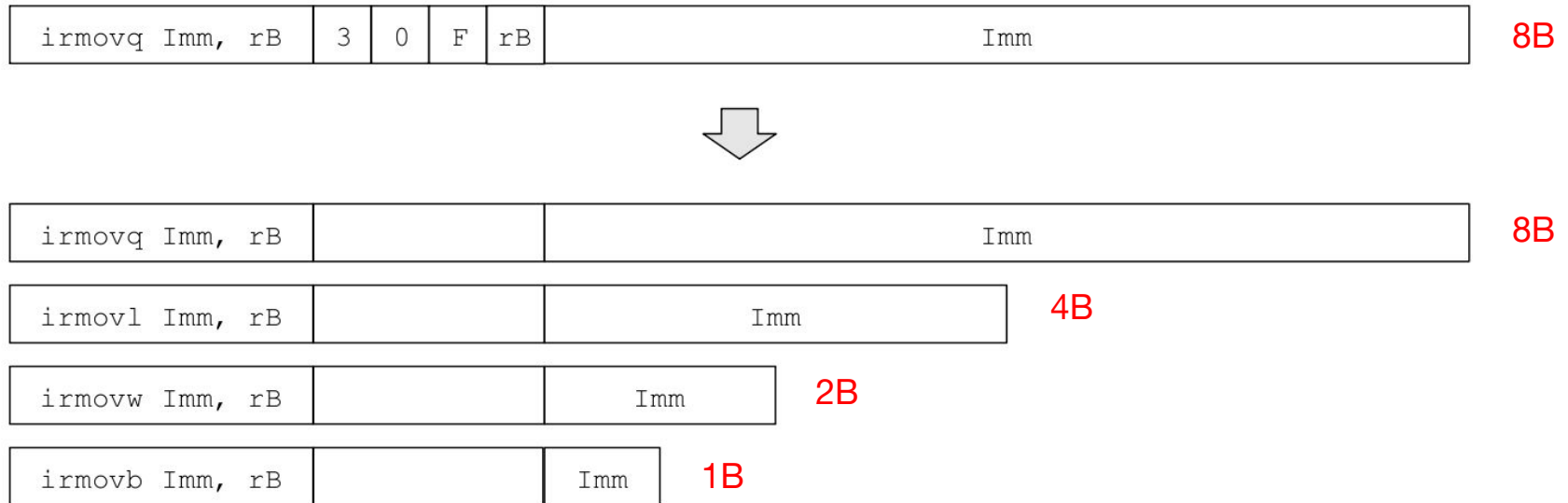

2. Part B

Part B

- Work in directory `processor-lab/sim/misc` in this part
 - Extend new instructions to the code generating tool
 - May have to modify following files, but not limited to:
 - `yas-grammar.lex`
 - `isa.h`
 - `isa.c`
 - `yas.c`

Part B

- Two types of implementation necessary
 - Extend `irmovq` to support different immediate sizes (sign-extended)
 - Which part of the instruction can be used to differentiate among insts?



Part B

- Two types of implementation necessary
 - Create a new instruction `callo` (call offset) for PC-relative addressing

```
0x021: 80350000000000000000 | call test
0x02a: 00 | halt
0x02b: 33f60600000000000000 | irmovq $6,%rsi
|
0x035: | test:
0x035: 33f0fbffffffffffffffff | irmovq $-5,%rax
0x03f: 90 | ret
0x040: 10 | nop
|
0x050: | .pos 0x50:
0x050: | stack:
```

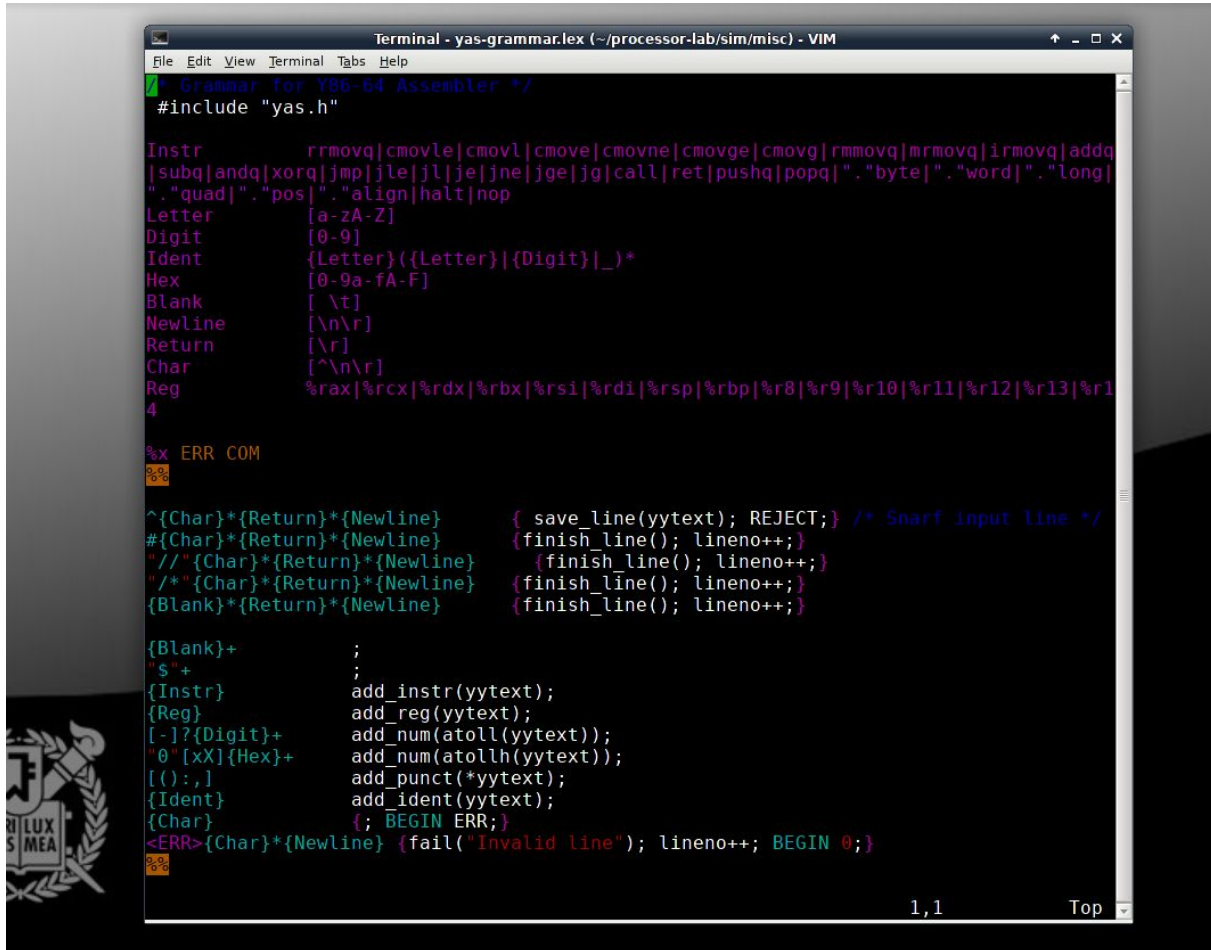
Part B

- Two types of implementation necessary
 - Create a new instruction `callo` (call offset) for PC-relative addressing

```
0x021: c00b000000      |      callo test
0x026: 00              |      halt
0x027: 33f60600000000000000 |      irmovq $6,%rsi
                                |
0x031:                 | test:
0x031: 33f0fbffffffffffffffff |      irmovq $-5,%rax
0x03b: 90              |      ret
0x03c: 10              |      nop
                                |
0x050:                 | .pos 0x50:
0x050:                 | stack:
```

Part B

- First, you need to fix grammar by modifying `yas-grammar.lex`



```
Terminal - yas-grammar.lex (~/.processor-lab/sim/misc) - VIM
File Edit View Terminal Tabs Help
/* Grammar for Y86-64 Assembler */
#include "yas.h"

Instr      rrmovq|cmovle|cmovl|cmove|cmovne|cmovge|cmovg|rrmovq|rrmovq|irmovq|addq
|subq|andq|xorq|jmp|jle|jl|je|jne|jge|jg|call|ret|pushq|popq|".byte"|".word"|".long|
|".quad"|".pos"|".align|halt|nop
Letter      [a-zA-Z]
Digit       [0-9]
Ident       {(Letter){(Letter){(Digit){|_})*}
Hex         [0-9a-fA-F]
Blank       [ \t]
Newline     [\n\r]
Return      [\r]
Char        [^\n\r]
Reg         %rax|%rcx|%rdx|%rbx|%rsi|%rdi|%rsp|%rbp|%r8|%r9|%r10|%r11|%r12|%r13|%r14
4

%x ERR COM
%%

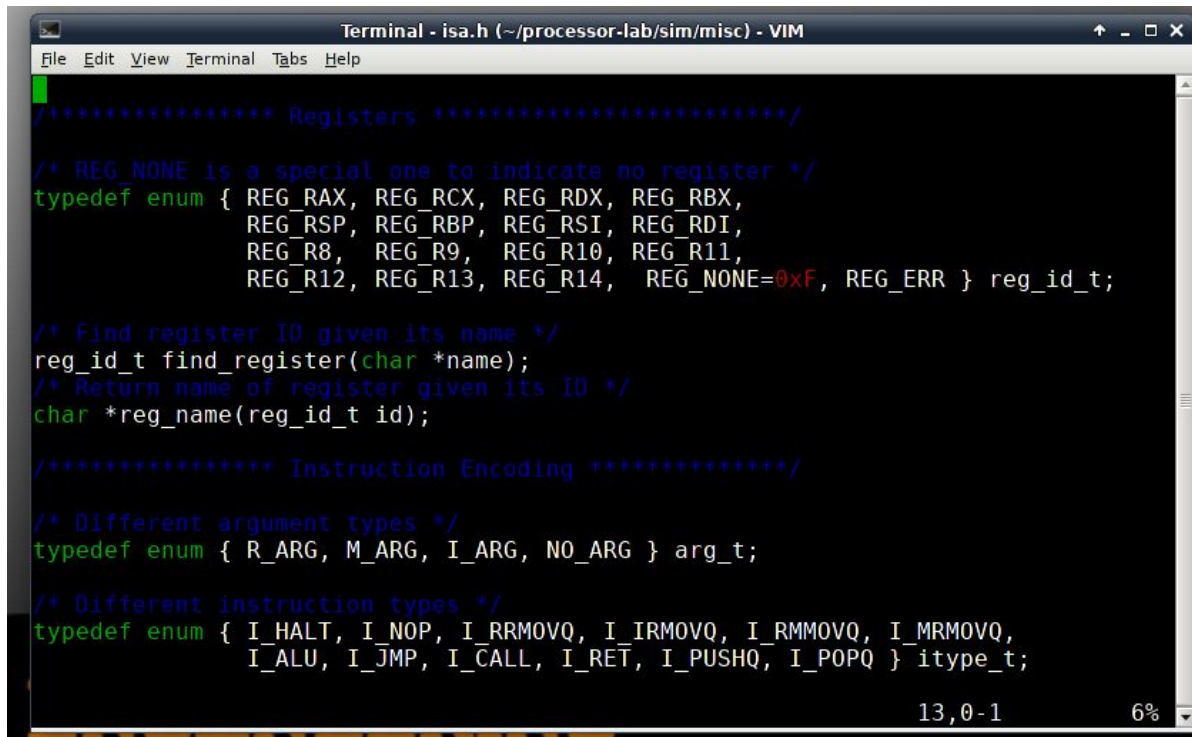
^{Char}*{Return}*{Newline}      { save_line(yytext); REJECT;} /* Snarf input line */
#{Char}*{Return}*{Newline}      { finish_line(); lineno++;}
/*/{Char}*{Return}*{Newline}    { finish_line(); lineno++;}
/*/{Char}*{Return}*{Newline}    { finish_line(); lineno++;}
{Blank}*{Return}*{Newline}      { finish_line(); lineno++;}

{Blank}+                          ;
"$"+                              ;
{Instr}                          add_instr(yytext);
{Reg}                            add_reg(yytext);
[-]?{Digit}+                     add_num(atoll(yytext));
"0"[xX]{Hex}+                   add_num(atollh(yytext));
[(){}],                          add_punct(*yytext);
{Ident}                         add_ident(yytext);
{Char}                          { BEGIN ERR;}
<ERR>{Char}*{Newline} {fail("Invalid line"); lineno++; BEGIN 0;}
%%

1,1 Top
```

Part B

- Next, modify `isa.h` to define behaviors for your new instructions
 - A new instruction type, an operand type, or functions for the new behavior etc.



```
Terminal - isa.h (~/processor-lab/sim/misc) - VIM
File Edit View Terminal Tabs Help

/***** Registers *****/

/* REG_NONE is a special one to indicate no register */
typedef enum { REG_RAX, REG_RCX, REG_RDX, REG_RBX,
               REG_RSP, REG_RBP, REG_RSI, REG_RDI,
               REG_R8, REG_R9, REG_R10, REG_R11,
               REG_R12, REG_R13, REG_R14, REG_NONE=0xF, REG_ERR } reg_id_t;

/* Find register ID given its name */
reg_id_t find_register(char *name);
/* Return name of register given its ID */
char *reg_name(reg_id_t id);

/***** Instruction Encoding *****/

/* Different argument types */
typedef enum { R_ARG, M_ARG, I_ARG, NO_ARG } arg_t;

/* Different instruction types */
typedef enum { I_HALT, I_NOP, I_RRMOVQ, I_IRMOVQ, I_RMMOVQ, I_MRMOVQ,
               I_ALU, I_JMP, I_CALL, I_RET, I_PUSHQ, I_POPQ } itype_t;

13,0-1 6%
```

Part B

- Next, modify `isa.c` accordingly
 - Note: Instruction parsing done only in 1-byte/8-byte manner (`get_byte_val` / `get_word_val`)

```
bool_t get_byte_val(mem_t m, word_t pos, byte_t *dest)
{
    if (pos < 0 || pos >= m->len)
        return FALSE;
    *dest = m->contents[pos];
    return TRUE;
}

bool_t get_word_val(mem_t m, word_t pos, word_t *dest)
{
    int i;
    word_t val;
    if (pos < 0 || pos + 8 > m->len)
        return FALSE;
    val = 0;
    for (i = 0; i < 8; i++) {
        word_t b = m->contents[pos+i] & 0xFF;
        val = val | (b << (8*i));
    }
    *dest = val;
    return TRUE;
}
```


Part B

- Lastly, `yas.c` should be modified to support PC-relative call with **labels**
 - Hint: It's better to separate offset operands from the immediate ones

```
0x021: c00b000000 |      callq test
0x026: 00          |      halt
0x027: 33f60600000000000000 |      irmovq $6,%rsi
                                |
0x031:           | test:
0x031: 33f0fbffffffffffffffff |      irmovq $-5,%rax
0x03b: 90          |      ret
0x03c: 10          |      nop
                                |
0x050:           | .pos 0x50:
0x050:           | stack:
```

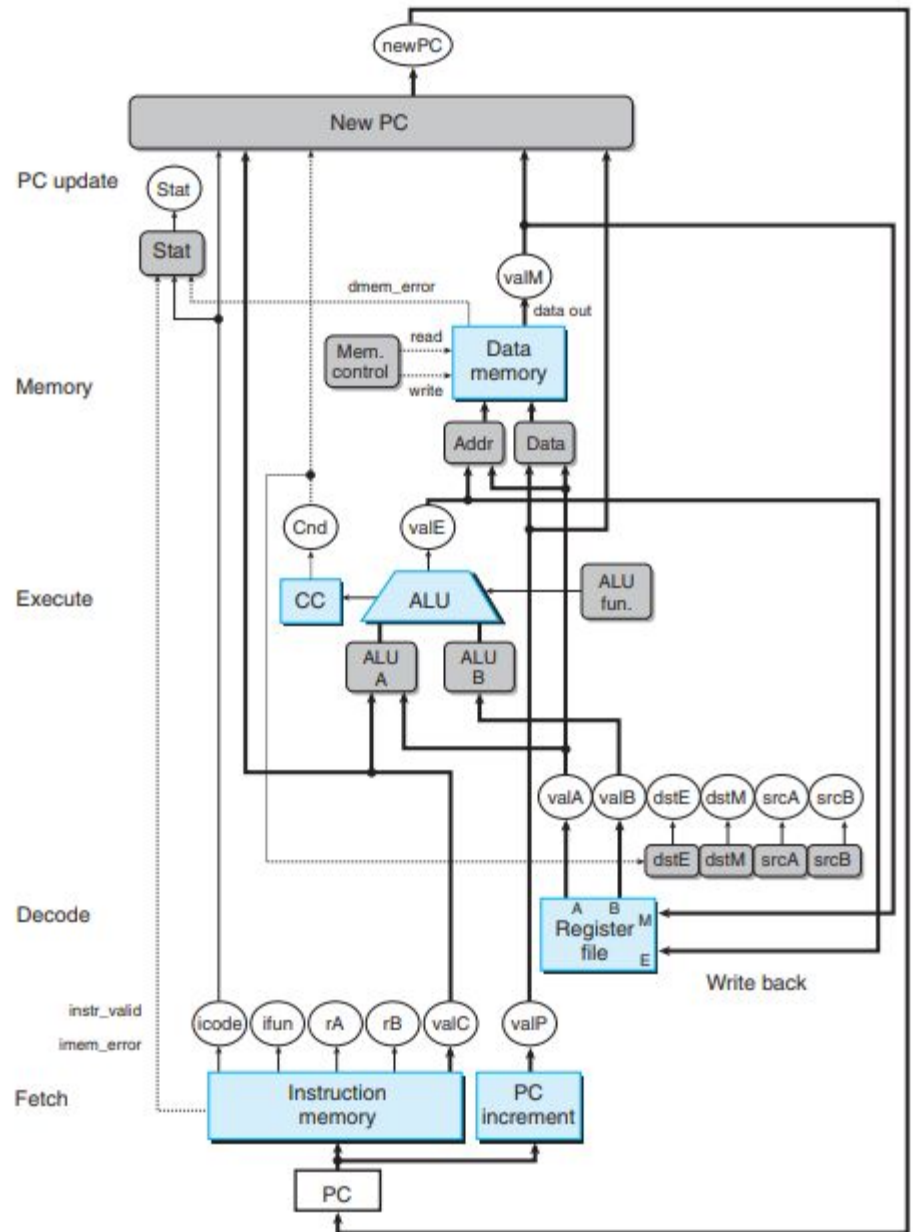
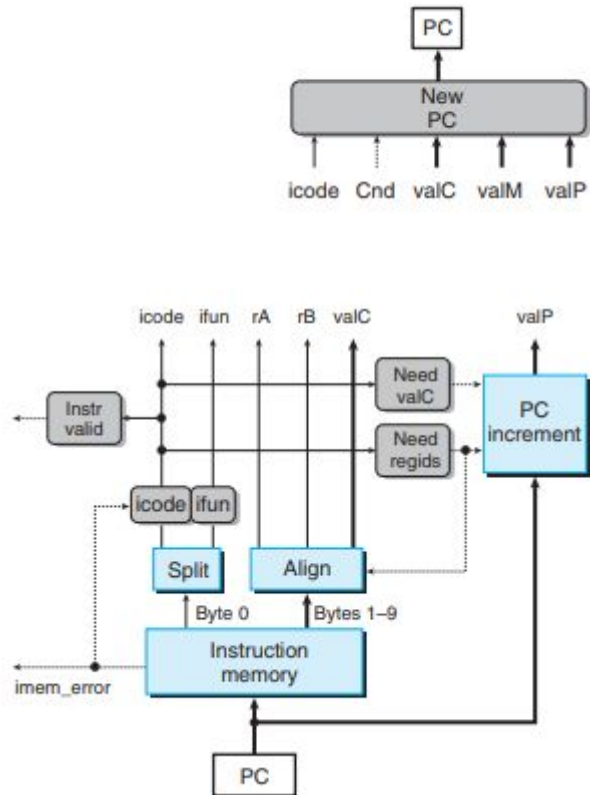
3. Part C

Part C

- Work in directory `processor-lab/sim/{seq, pipe}` in this part
 - Extend new instructions to the `sim/pipe` simulator
 - May have to modify following files, but not limited to:
 - `seq-std.hcl`
 - `ssim.c`
 - `pipe-std.hcl`
 - `psim.c`

Part C

- Generating new PC for SEQ



For questions contact
comparch@csap.snu.ac.kr