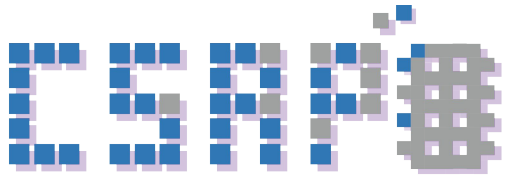
Computer Architecture Lab Session

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

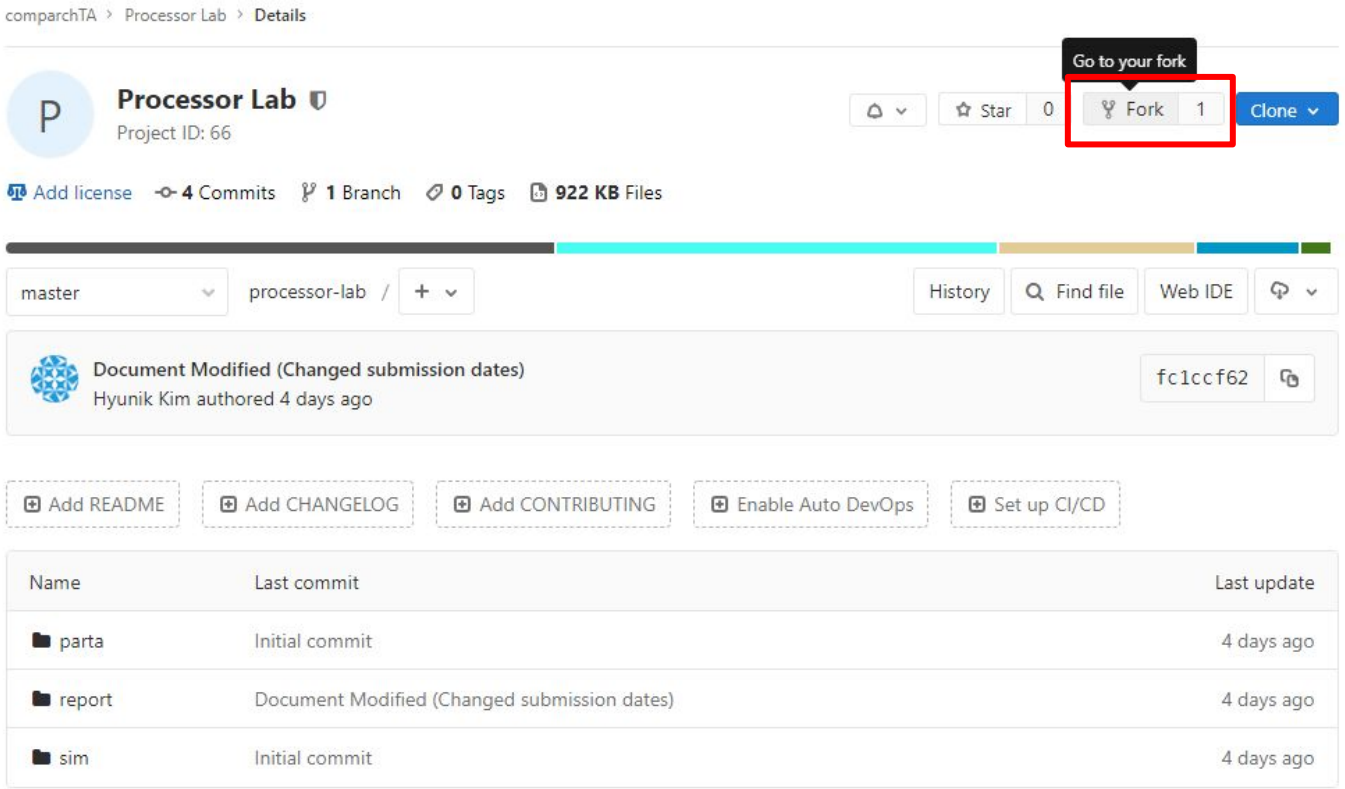
Overview

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

0. Getting Started

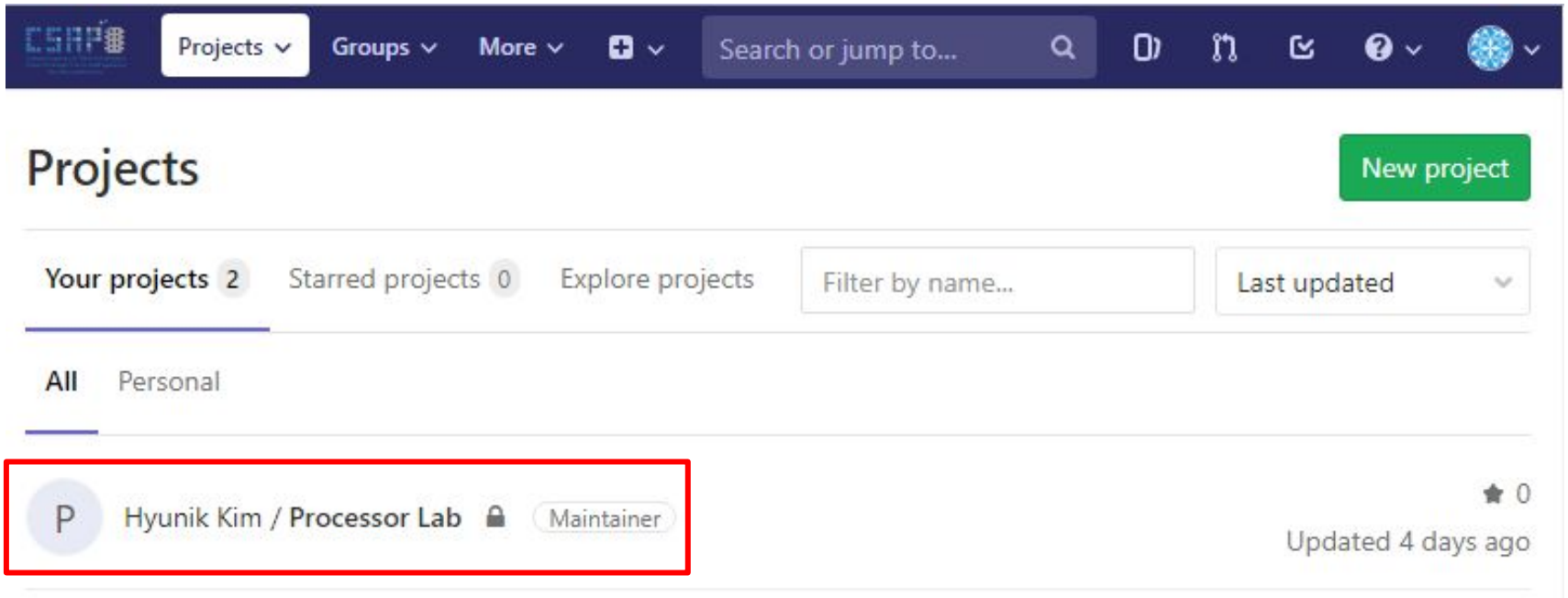
Fork Repository

- Fork from the repository `comparchTA/Processor Lab`



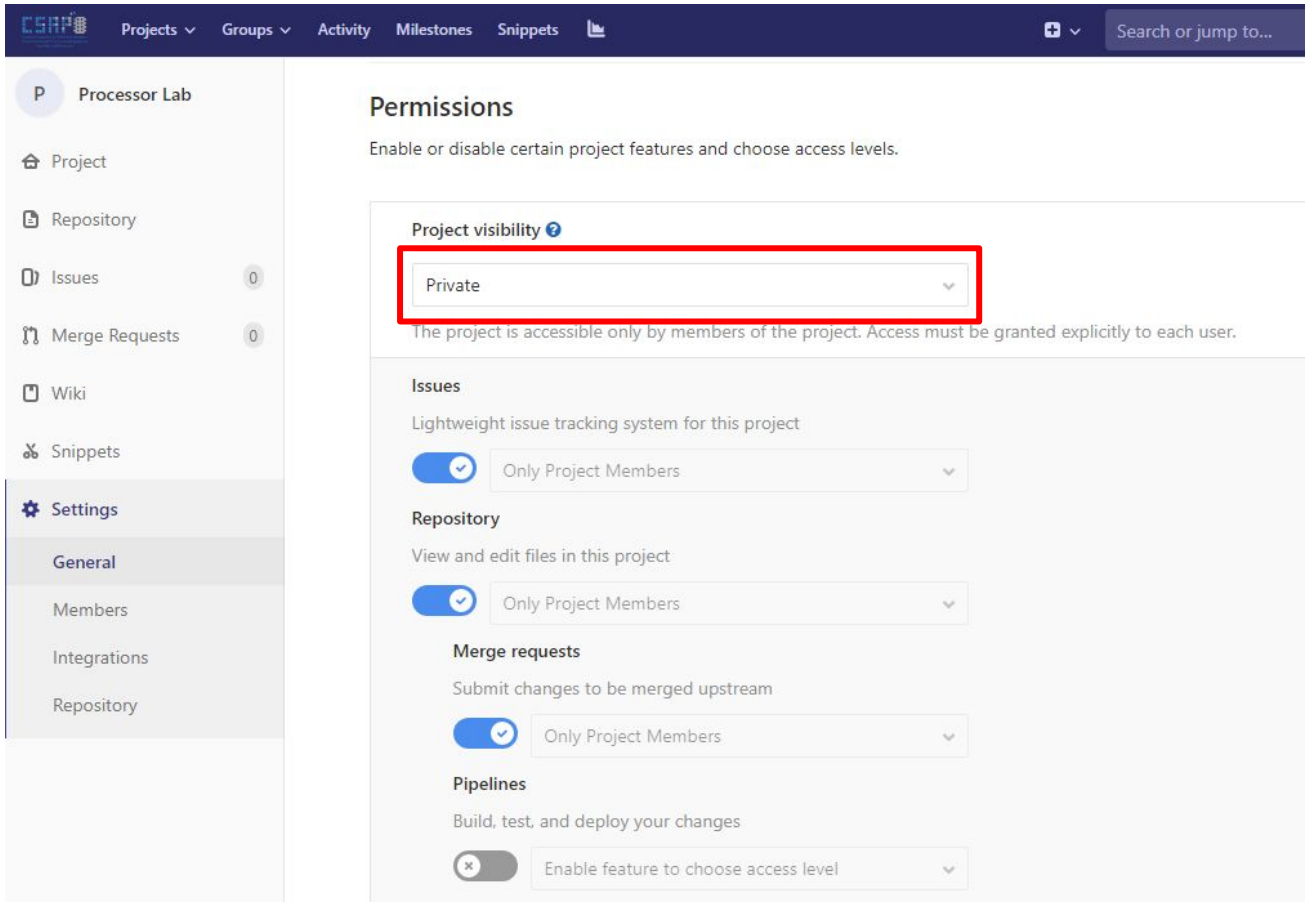
Fork Repository

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



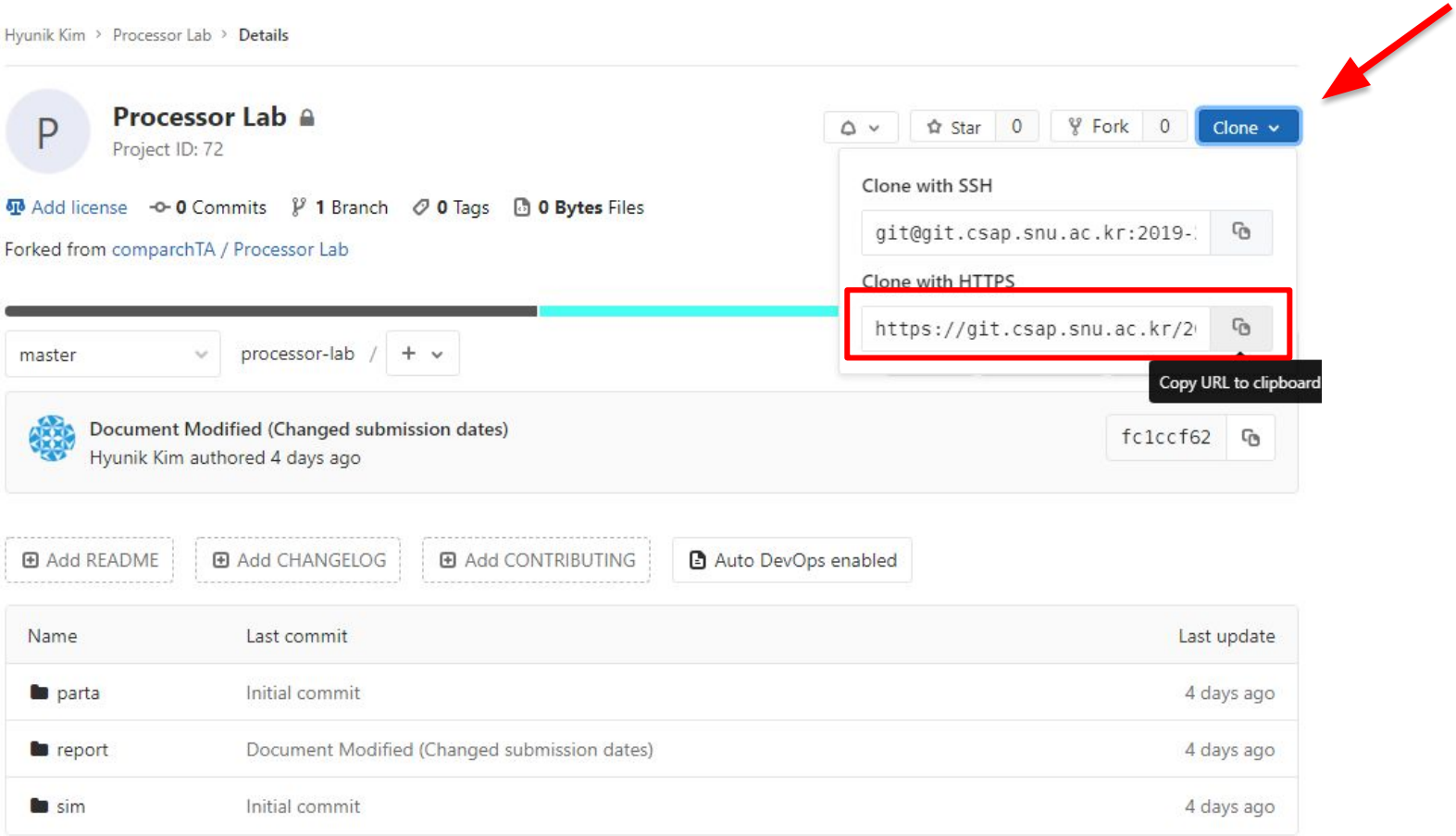
Fork Repository

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



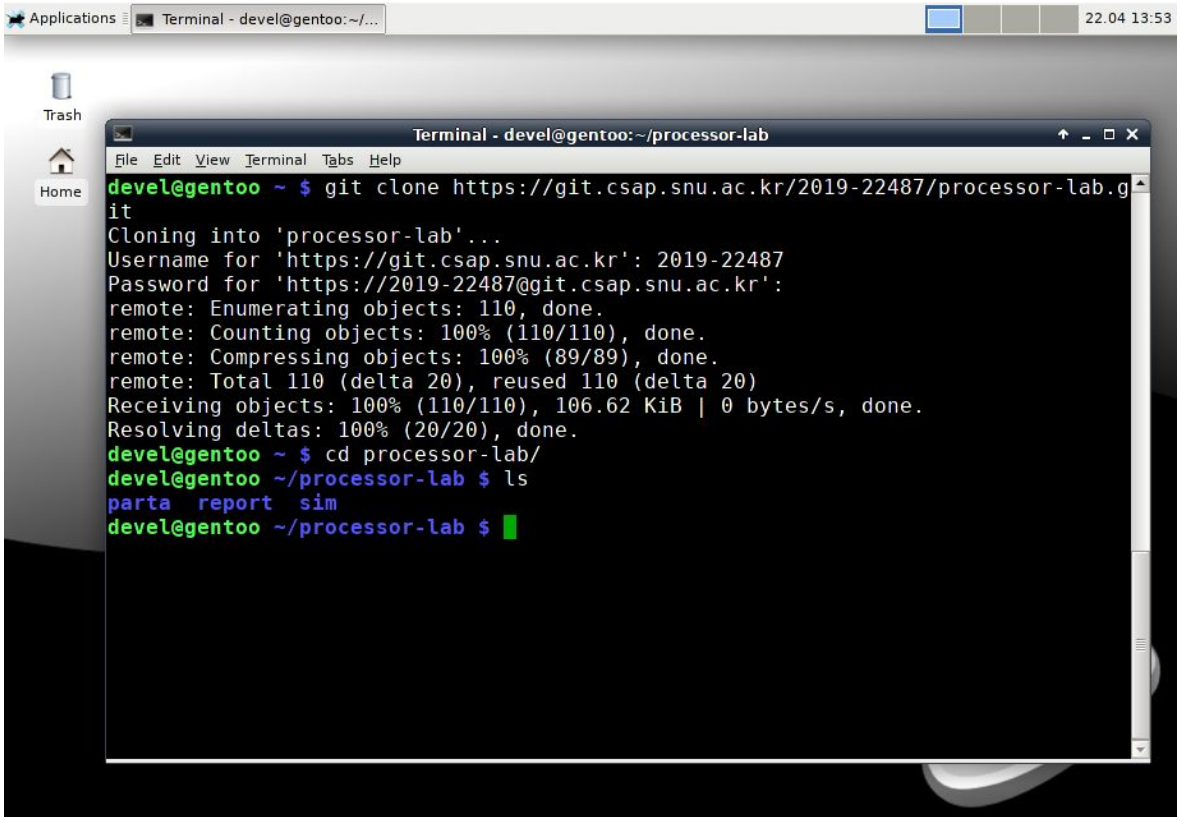
Clone Repository

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



Clone Repository

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



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 (calle-save & caller-save)
  - Programs:
    - `sum.ys`
    - `rsum.ys`
    - `copy.ys`

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.ys`: 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.ys`
  - Simulate:
    - `misc$ ./yis sum.yo`

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      0x0000000000000cba
%rsp:   0x0000000000000000      0x0000000000000100
%r10:   0x0000000000000000      0x0000000000000c00

Changes to memory:
0x00f8: 0x0000000000000000      0x000000000000001d
```

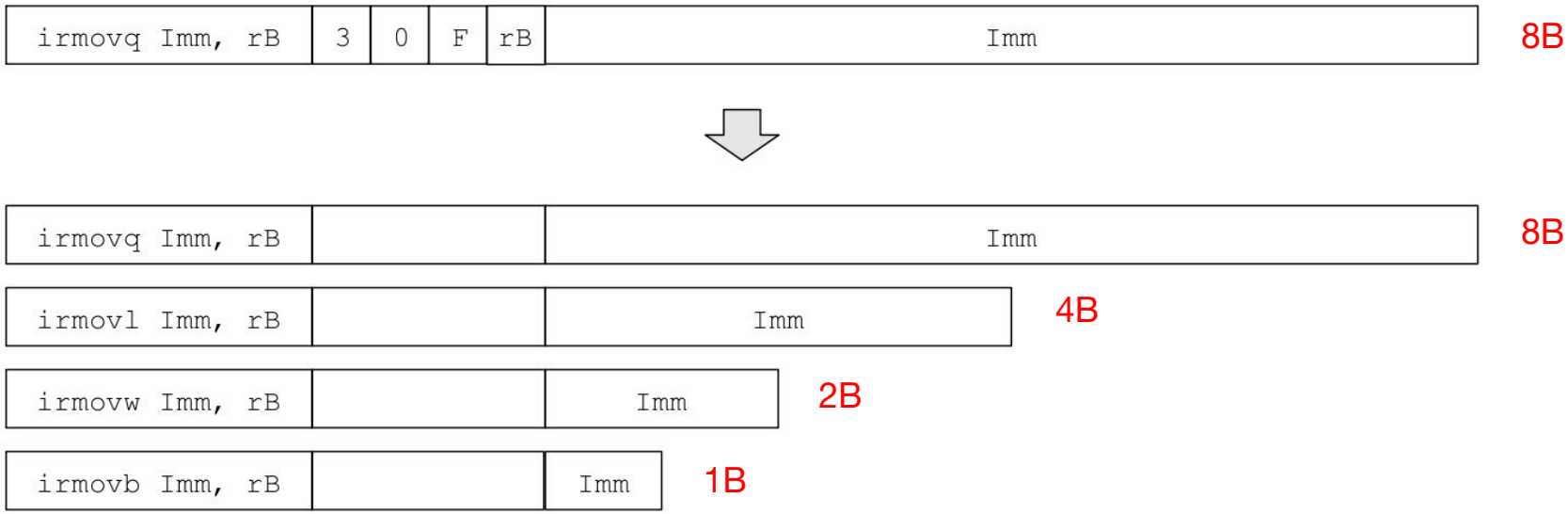
2. Part B

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: 8035000000000000   |   call test
0x02a: 00                 |   halt
0x02b: 33f606000000000000 |   irmovq $6,%rsi
                          |
0x035:                    | test:
0x035: 33f0fbffffffffffff |   irmovq $-5,%rax
0x03f: 90                 |   ret
0x040: 10                 |   nop
                          |
0x050:                    | .pos 0x50:
0x050:                    | stack:
```

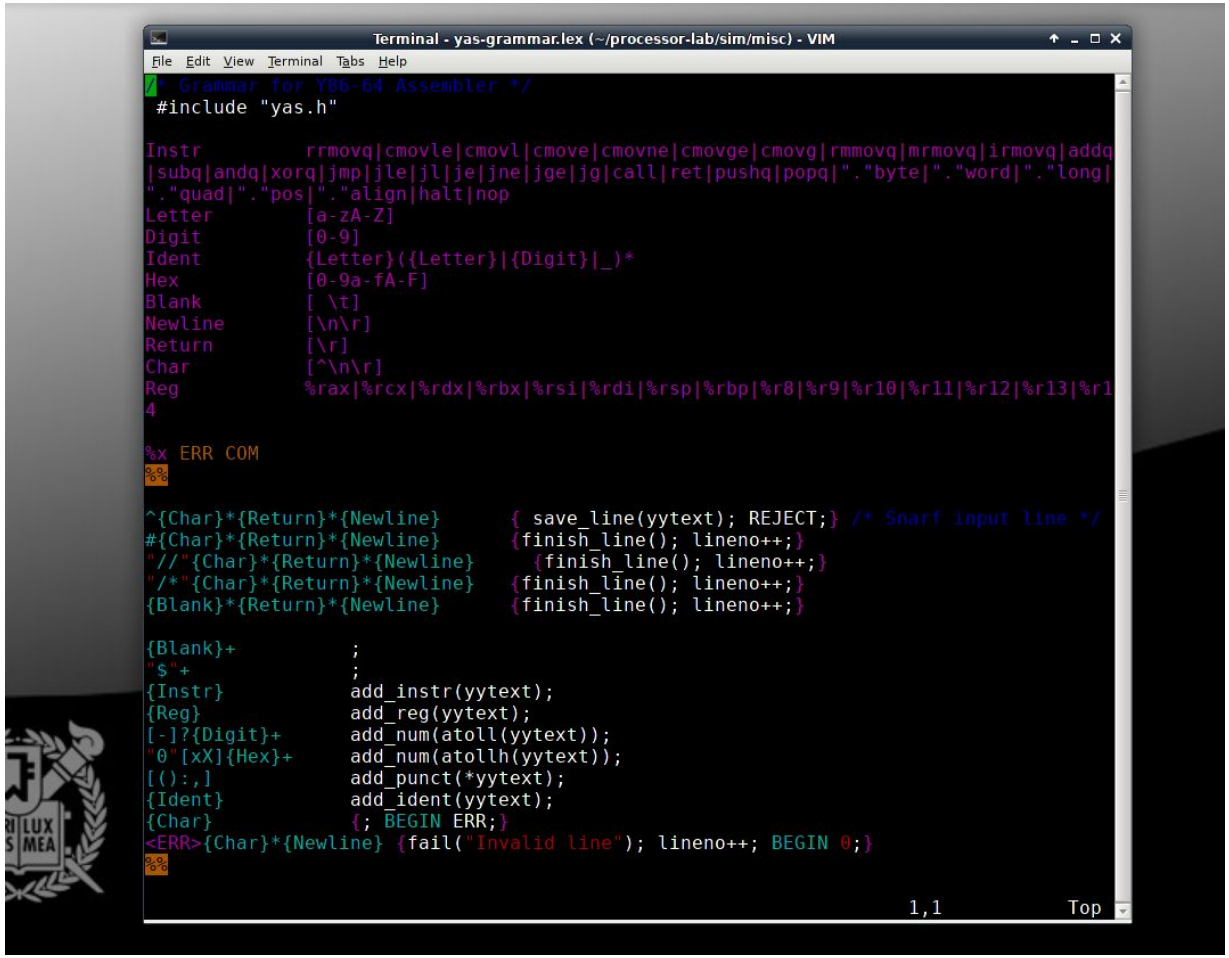
Part B

- 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: 33f0fbffffffffffffff  |      irmovq $-5,%rax
0x03b: 90                    |      ret
0x03c: 10                    |      nop
                             |
0x050:                       | .pos 0x50:
0x050:                       | stack:
```

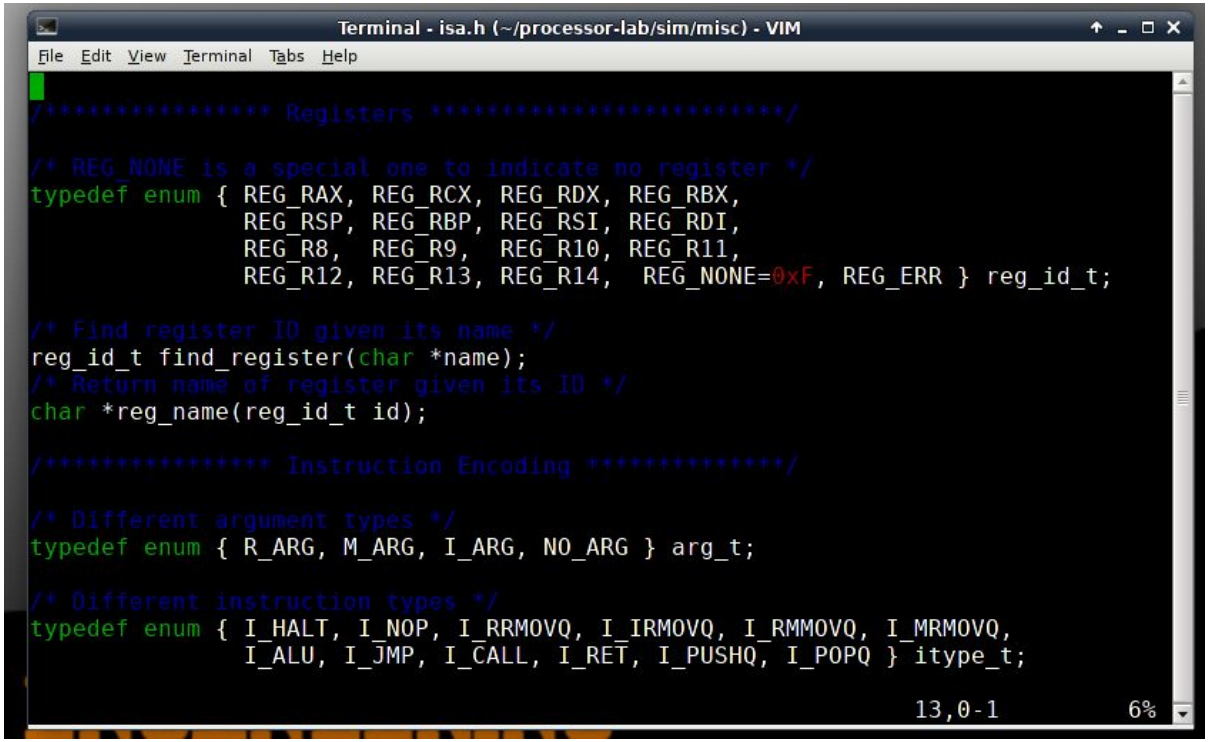
Part B

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



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.



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            |     callo test
0x026: 00                    |     halt
0x027: 33f60600000000000000  |     irmovq $6,%rsi
                             |
0x031:                       | test:
0x031: 33f0fbffffffffffffff  |     irmovq $-5,%rax
0x03b: 90                    |     ret
0x03c: 10                    |     nop
                             |
0x050:                       | .pos 0x50:
0x050:                       | stack:
```

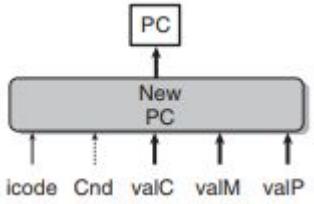
3. Part C

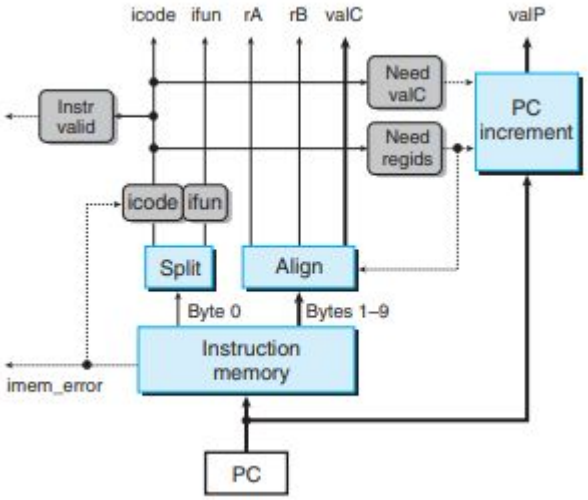
Part C

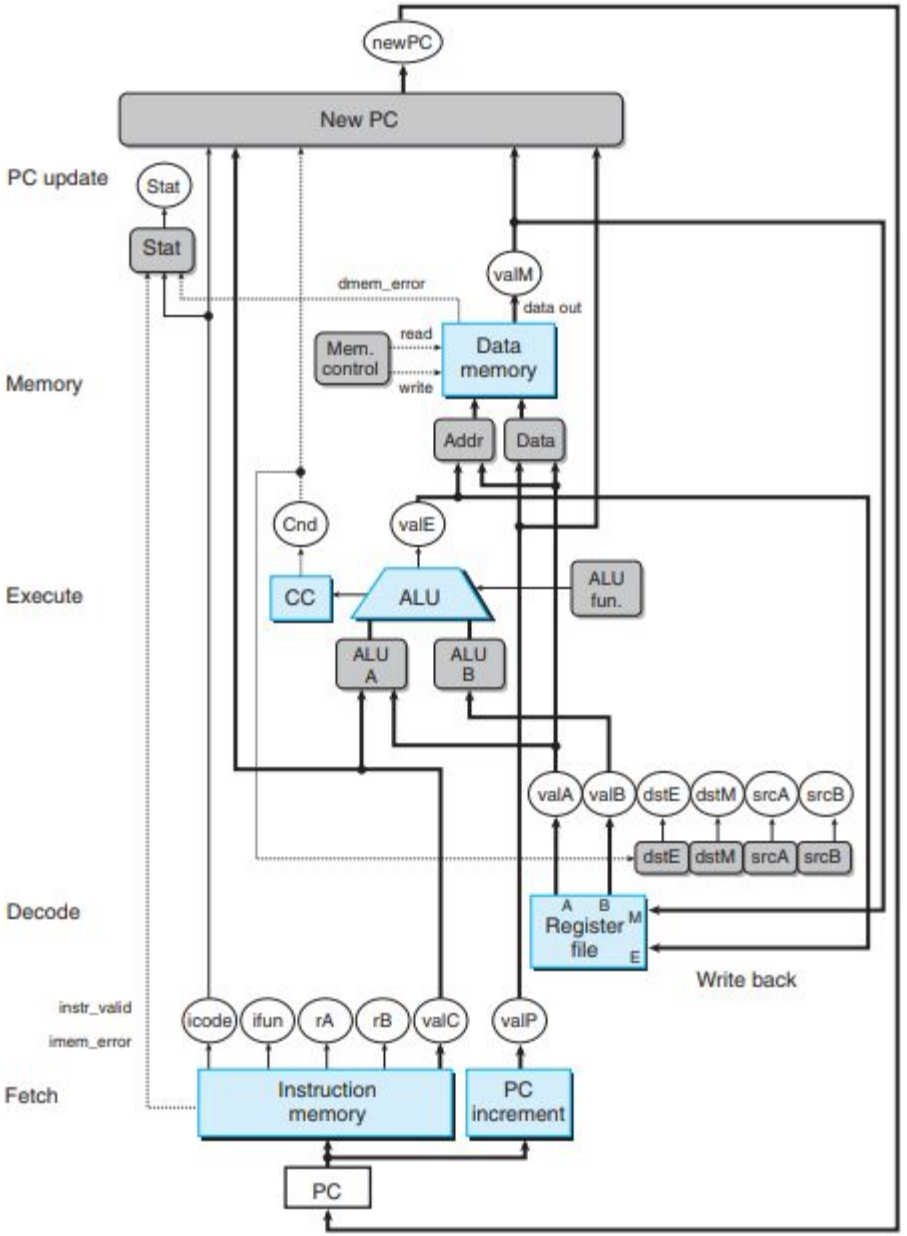
- Work in directory `processor-lab/sim/{seq, pipe}` in this part
  - Extend new instructions to the sim/pipe simulator
  - May have to modify following files, but not limited to:
    - `seq-std.hcl`
    - `ssim.c`
    - `pipe-std.hcl`
    - `psim.c`

Part C

- Generating new PC for SEQ







For questions contact

comparch@csap.snu.ac.kr