EI 209
Computer Organization
Fall 2014

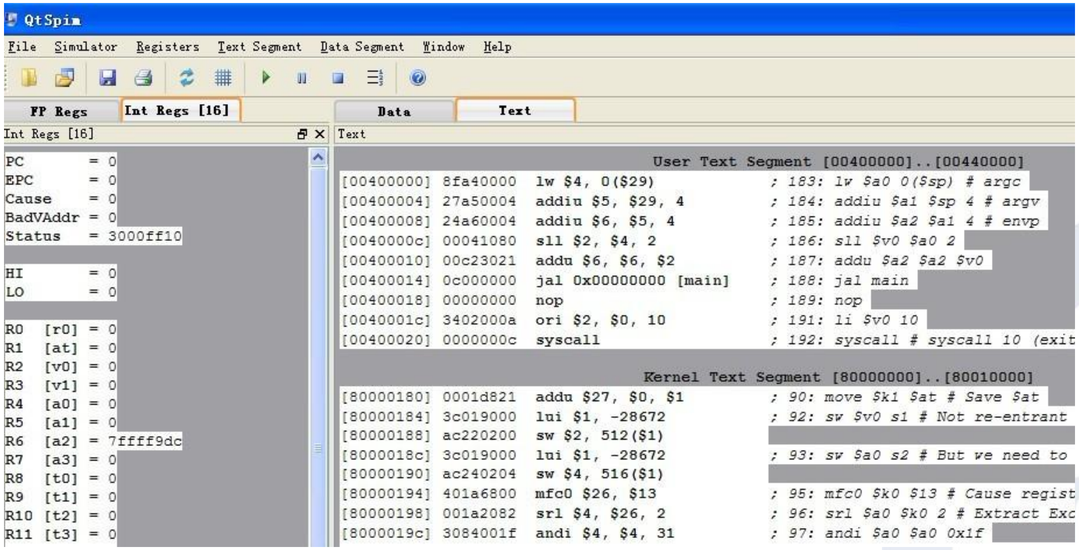
Course Project 2: MIPS CPU Simulator

http://nsec.sjtu.edu.cn/

[Adapted from *Computer Organization and Design, 4th Edition*,
Patterson & Hennessy, © 2012, MK]

Project Requirement

- Write your own MIPS Simulator (like spim)
- Run your own mips code on your simulator



```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffff9dc
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
```

```
                        User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit

                       Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13              ; 95: mfc0 $k0 $13 # Cause regist
[80000198] 001a2082  srl $4, $26, 2             ; 96: srl $a0 $k0 2 # Extract Exc
[8000019c] 3084001f  andi $4, $4, 31            ; 97: andi $a0 $a0 0x1f
```

SPIM: A MIPS32 Simulator

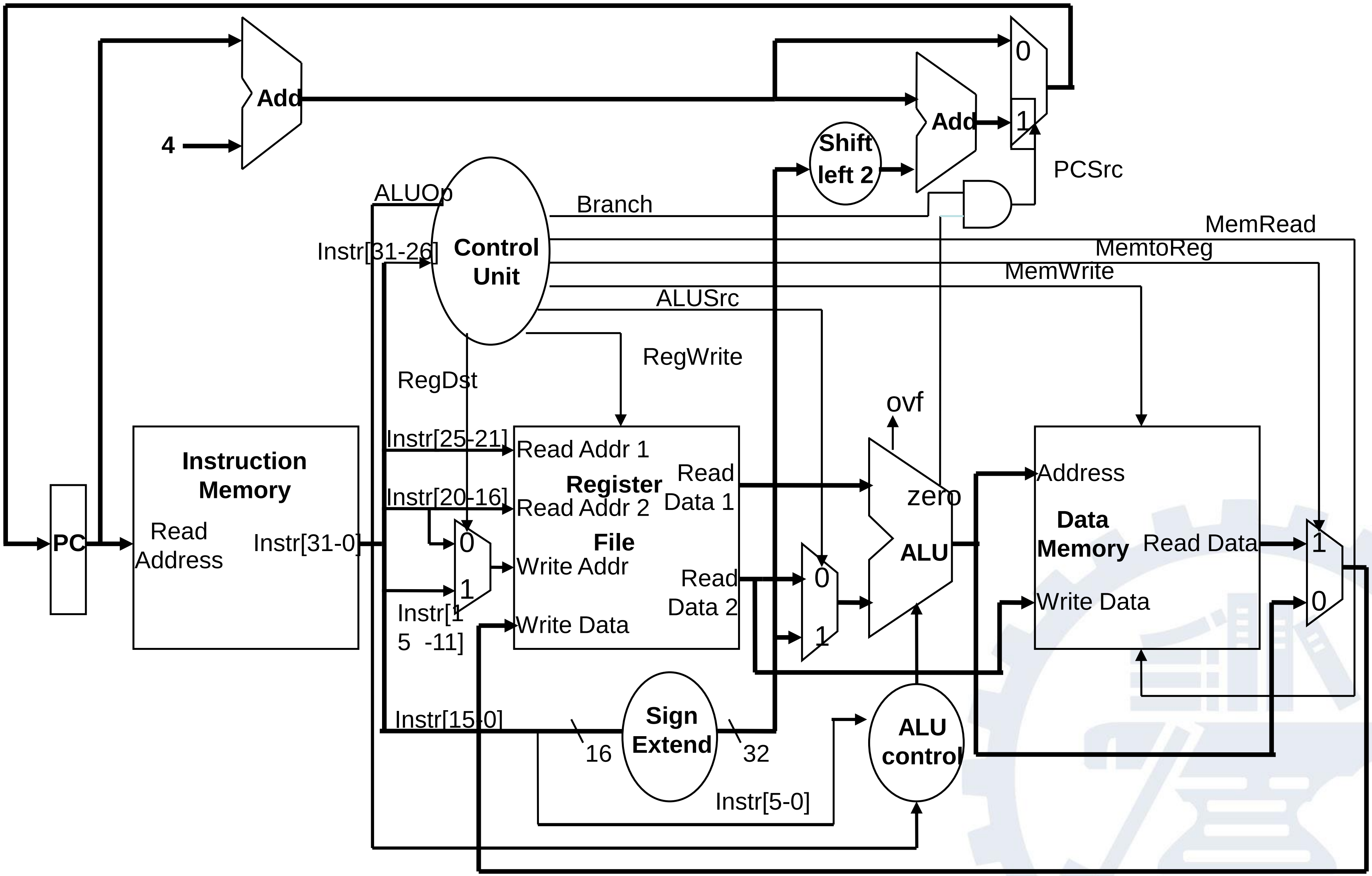
Project Requirement Cont.

- Major task
  - Write all mips cpu component
  - Link all components together
- Additional task
  - Run your mips program in project 1 on your mips simulator

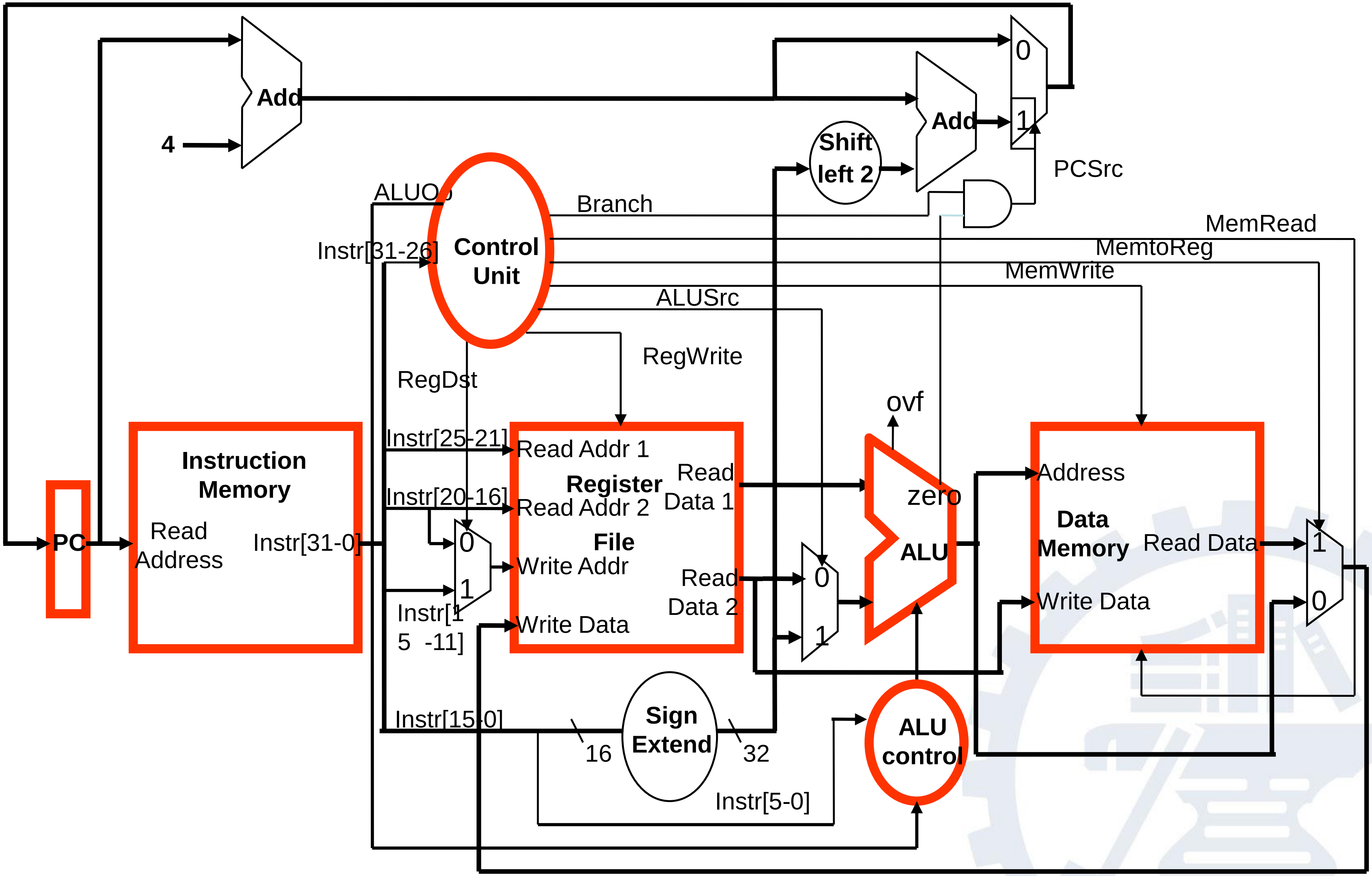
Project Requirement Cont.

- Instructions :
- lw - load word from memory
- sw - store word to memory
- add - add two register
- addi - add immediate
- beq - branch if equal
- j - jumps to the calculated address

Review: Single Cycle Data-path



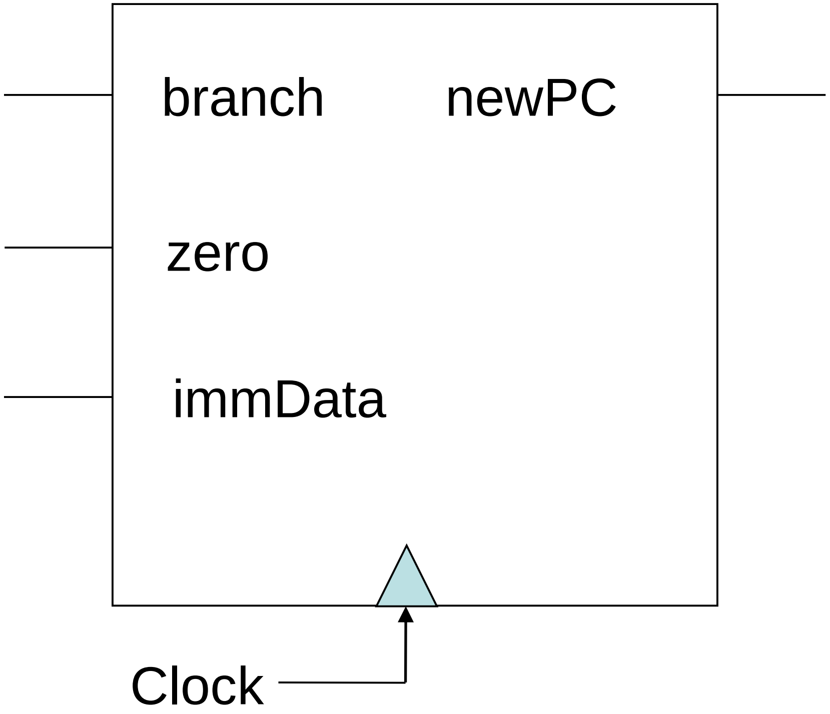
Review: Single Cycle Data-path



You have to implement components which is marked with red border.

Step1: Implement Components - Program Counter

- In this project, we use one component to calculate program counter.

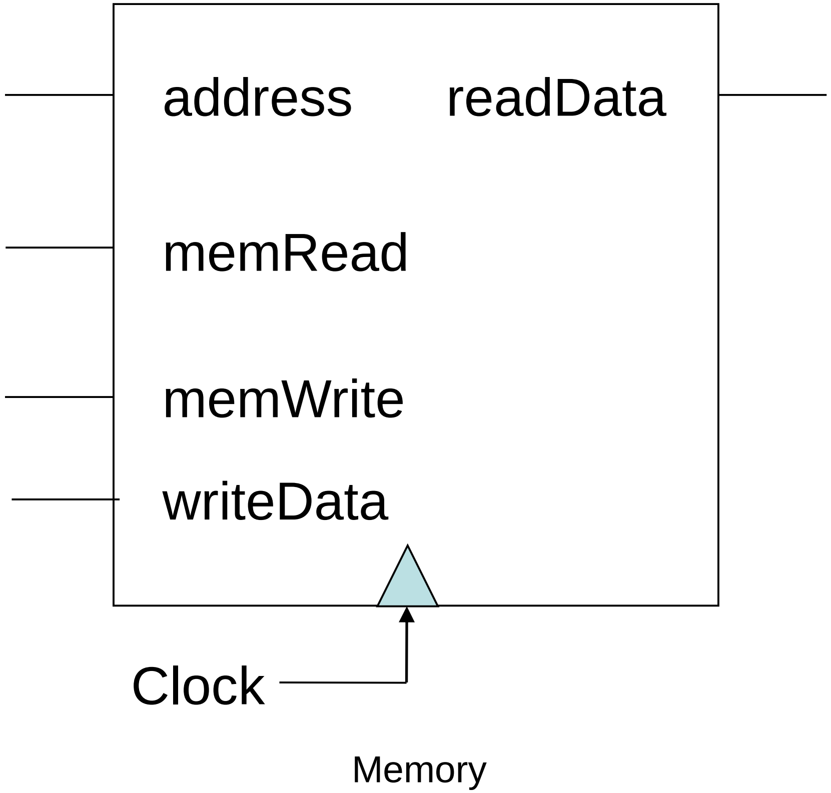


Program Counter

Branch	Zero	Action
0	X	newPC <- PC + 4
1	0	newPC <- PC + 4
1	1	newPC <- PC + immData << 2

Step1: Implement Components - Memory

- Two memory units in mips cpu: instruction memory and data memory

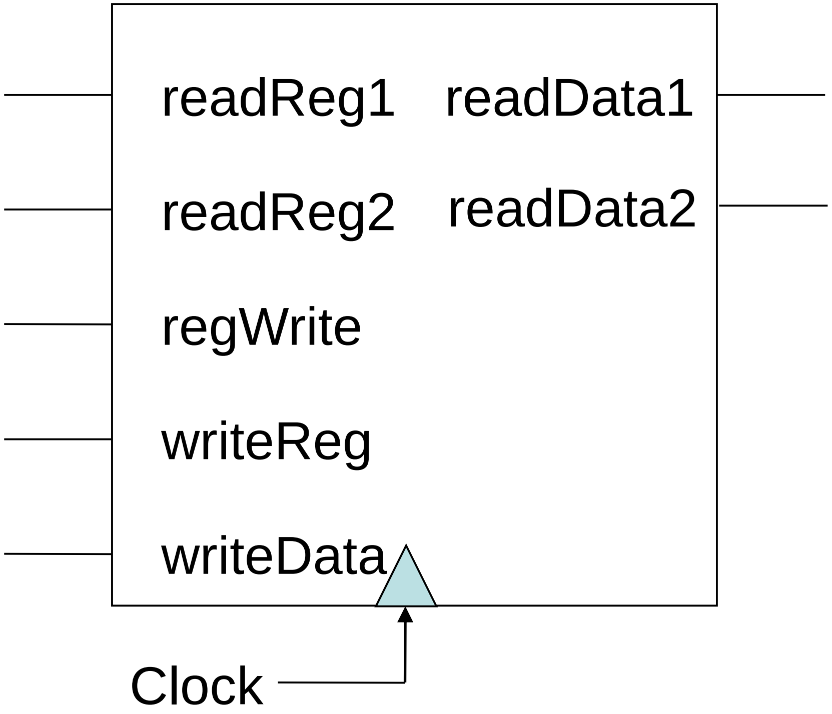


Memory

memRead	memWrite	Action
1	X	readData <- mem[address]
X	1	mem[address] <- writeData

Step1: Implement Components - Register

- All mips instruction are operated with mips cpu, the register component contains 32 registers



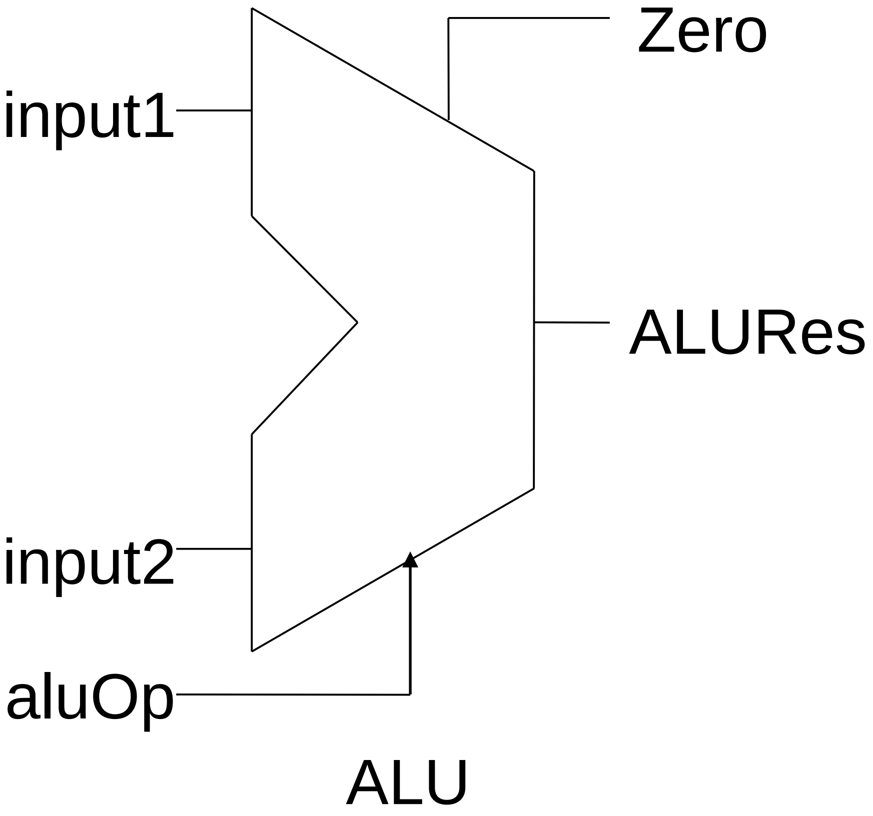
Register

reg Write	Action
1	reg[writeReg] <- writeData

X	readData1 <- reg[readReg1]
	readData2 <- reg[readReg2]

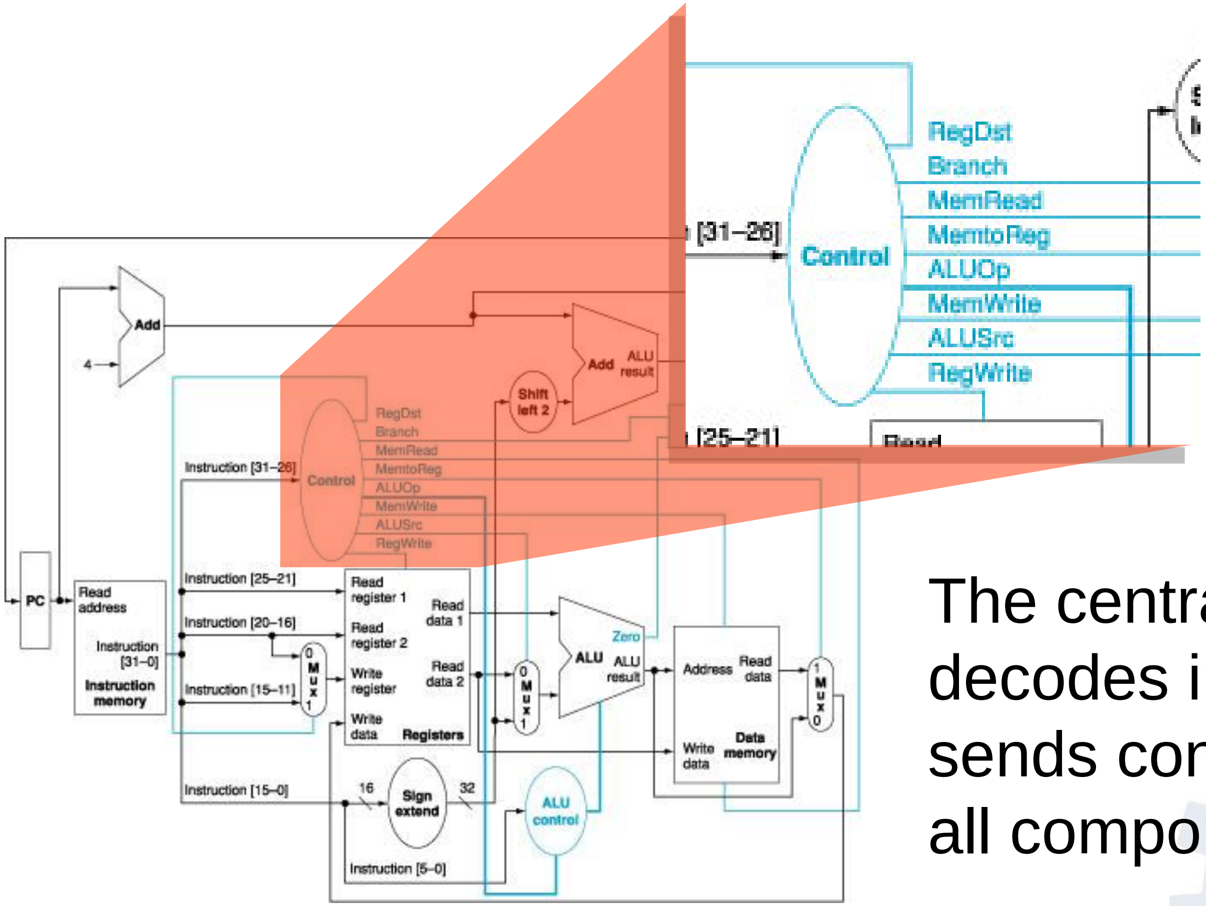
Step1: Implement Components - ALU

- ALU unit uses aluOp signal to determine its action



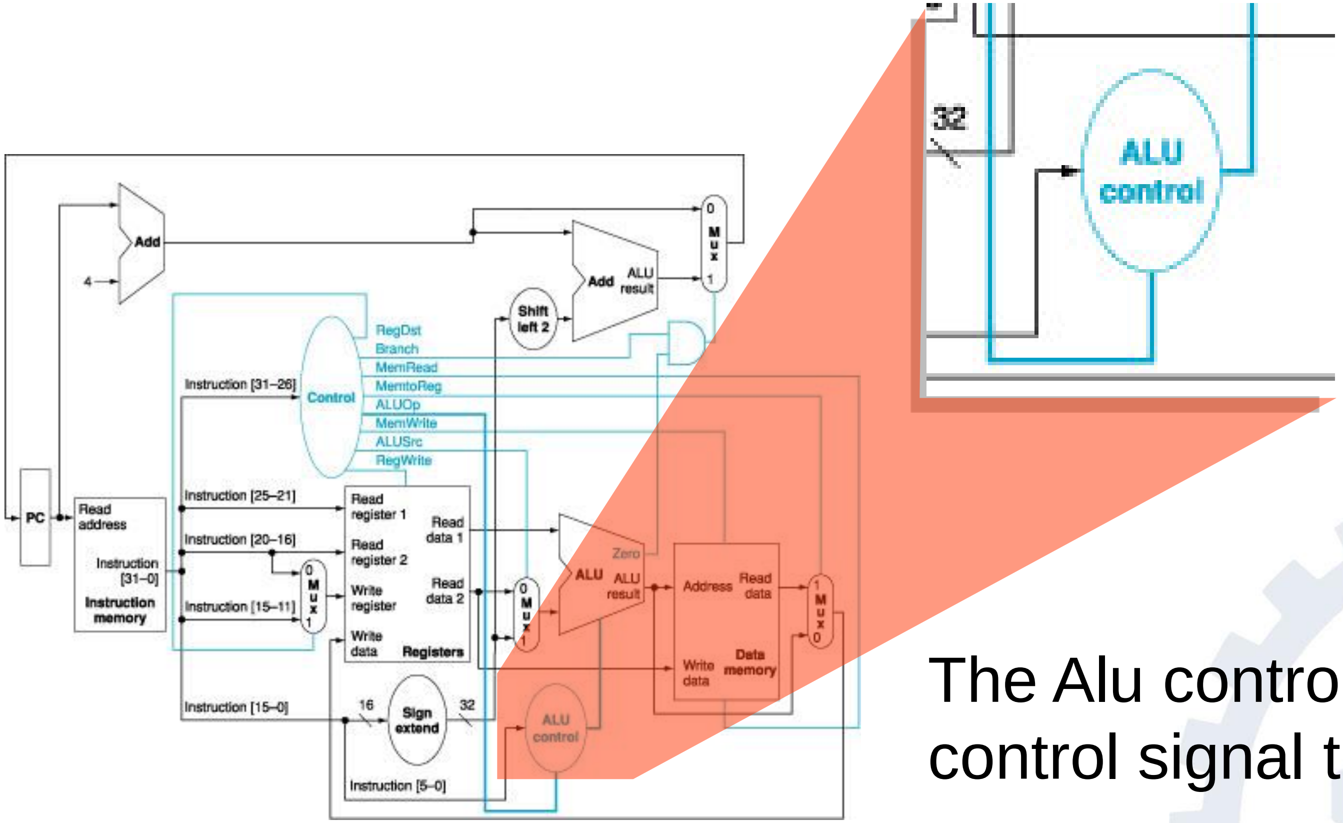
aluOp	Action
0000	And
0001	Or
0010	Add
0110	Subtract
0111	Set on less than
1100	NOR

Step1: Implement Components-Ctr and Aluctr



The central control unit decodes instruction and sends control signal to all component

Step1: Implement Components-Ctr and Aluctr

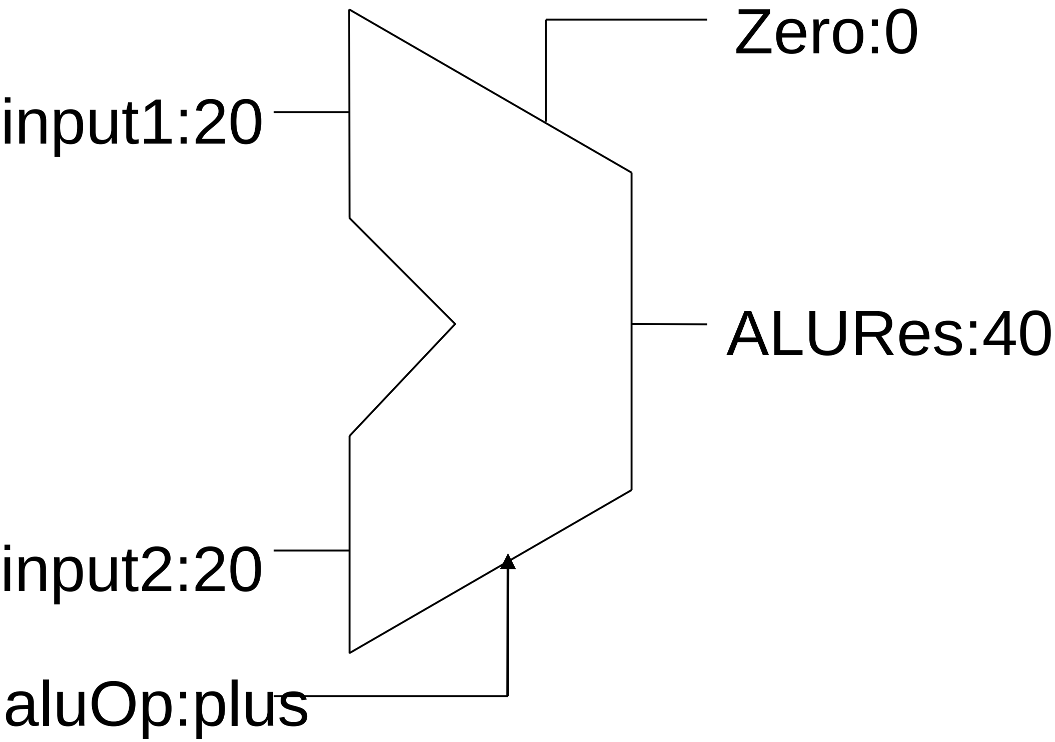


The Alu control unit give control signal to ALU

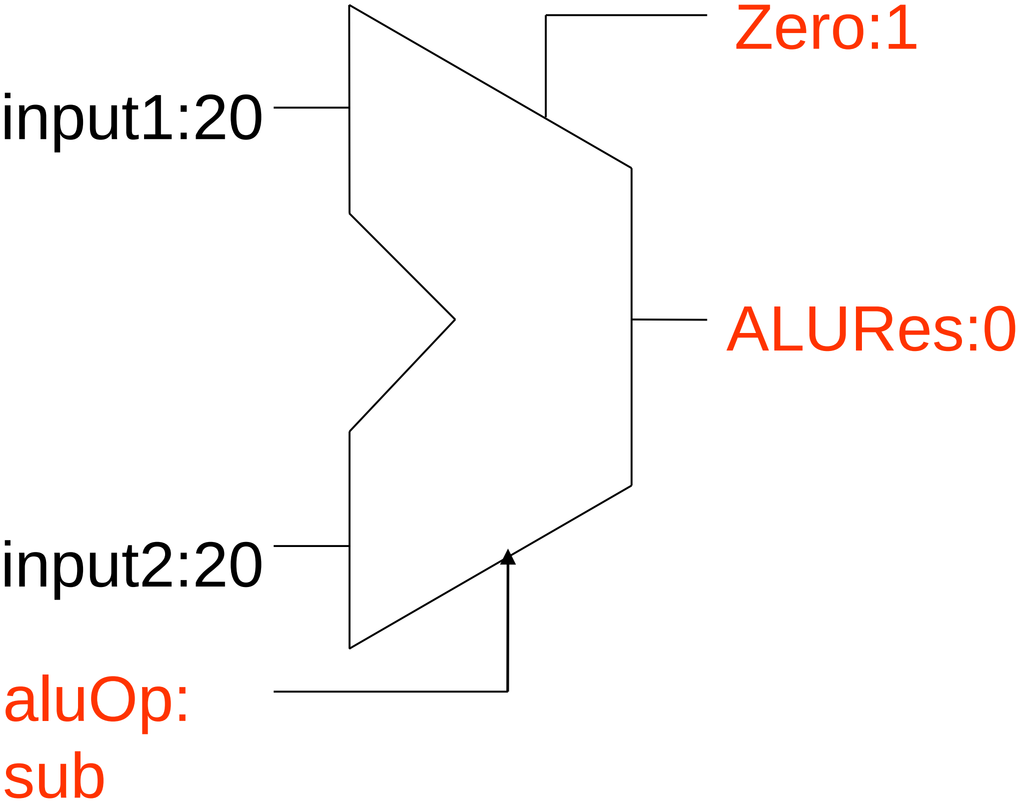
Step1: Implement Components-How To

- Each component is a c++ class in our project. Eg. Control Unit is a class called Ctr
- You have to overwrite two member function of each class: *onChange* and *onClock*
- If one of input line changed, the function *onChange* will be called.
- If component receive clock signal, the function *onClock* will be called.

Step1: Implement Components-How To Cont.



Step1: Implement Components-How To Cont.



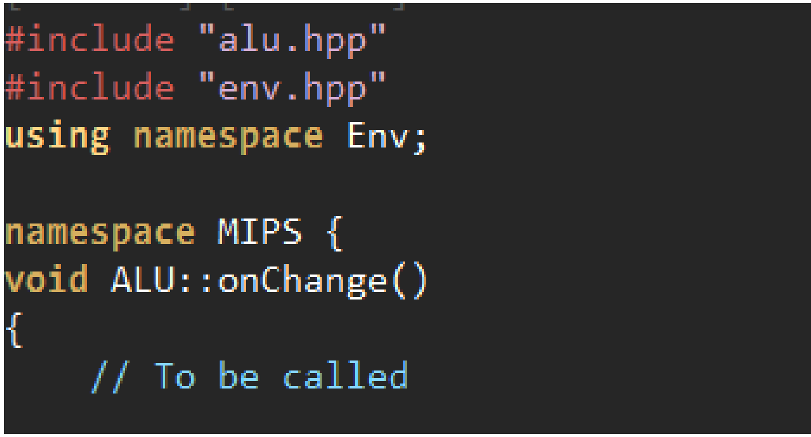
Zero:1

input1:20

ALURes:0

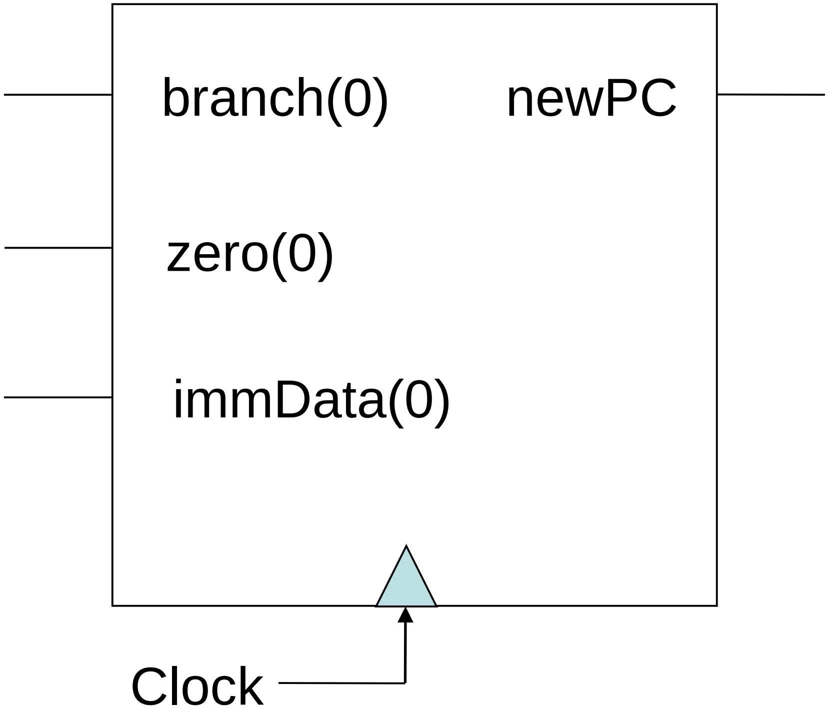
input2:20

aluOp:
sub



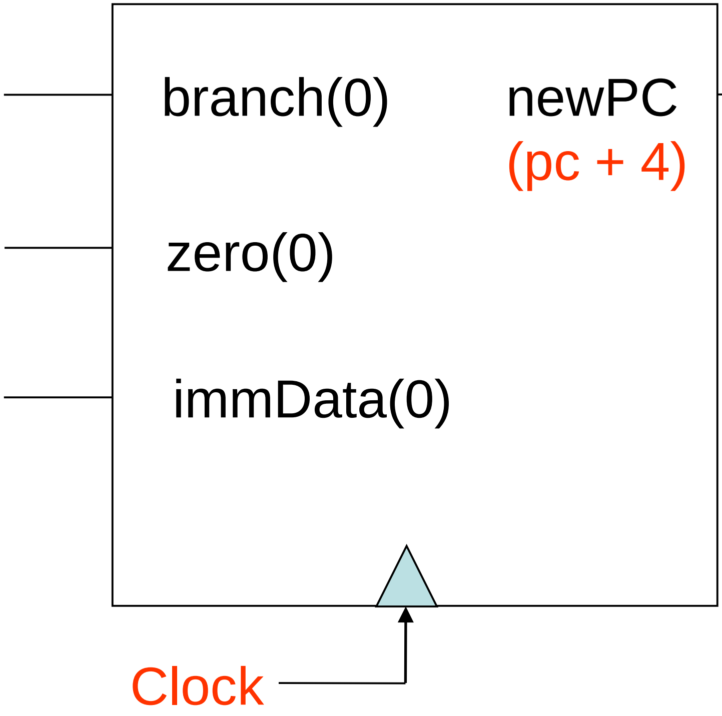
```
#include "alu.hpp"
#include "env.hpp"
using namespace Env;

namespace MIPS {
void ALU::onChange()
{
    // To be called
```

Step1: Implement Components-Clocking Methodologies

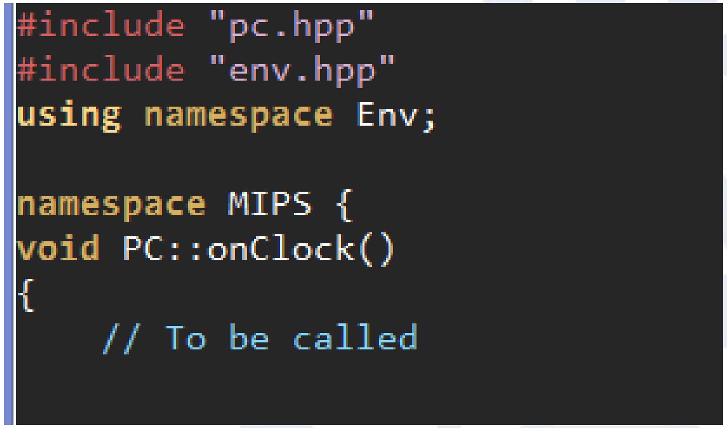


- newPC will not change until it receives a clock signal.

Step1: Implement Components-Clocking Methodologies



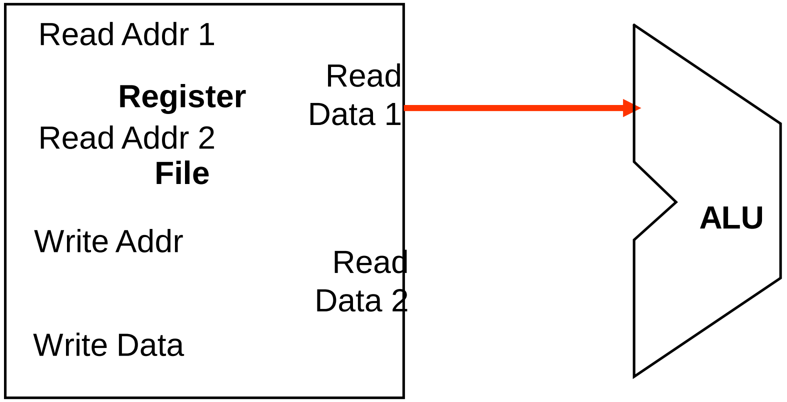
When component receive a clock signal, the onClock method will be called

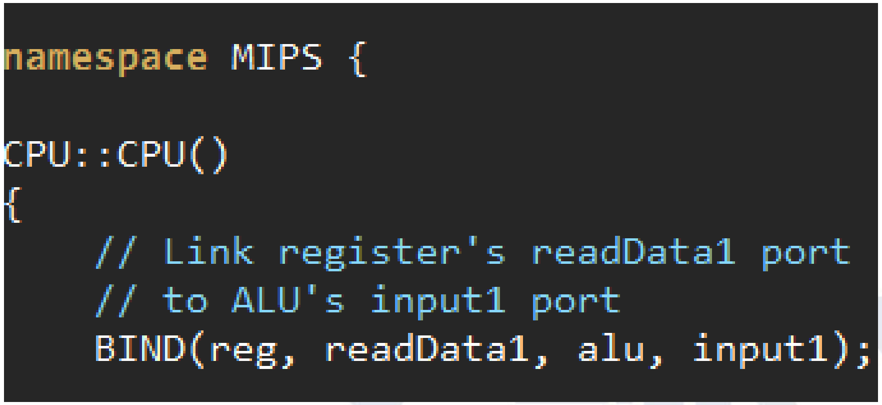


```
#include "pc.hpp"
#include "env.hpp"
using namespace Env;

namespace MIPS {
void PC::onClock()
{
    // To be called
```

Step2: Connect Components

- You can connect all components in the initializer of class CPU





```
namespace MIPS {

CPU::CPU()
{
    // Link register's readData1 port
    // to ALU's input1 port
    BIND(reg, readData1, alu, input1);
```

Review: Major Task

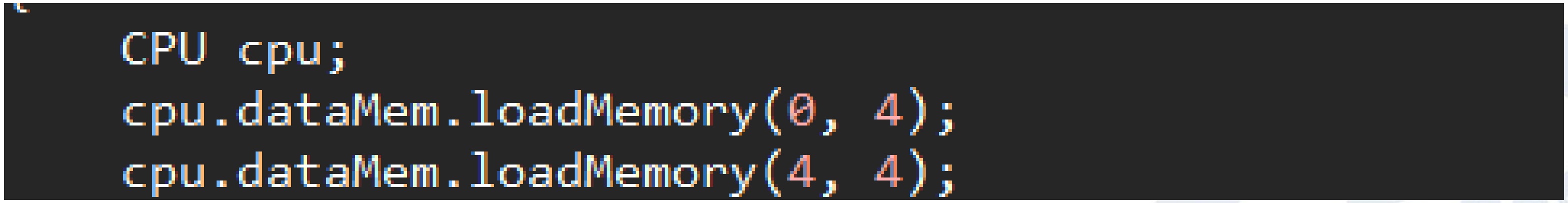
- You have to implement internal logic for all components of MIPS processor
- You have to connect all units of MIPS processor in the initializer of class CPU

Additional Task: Run Program on Your Processor

- Challenges:
- Input & Output
- How to determine whether a program is terminated

Additional Task: Run Program on Your Processor

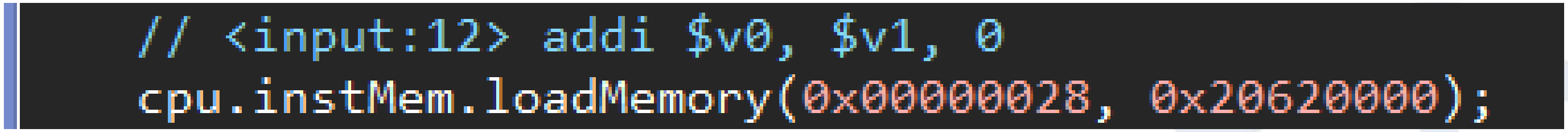
- Challenges:
- Input & Output
  - Input: you can load data into data memory before the execution of program



```
    CPU cpu;
    cpu.dataMem.loadMemory(0, 4);
    cpu.dataMem.loadMemory(4, 4);
```

Additional Task: Run Program on Your Processor

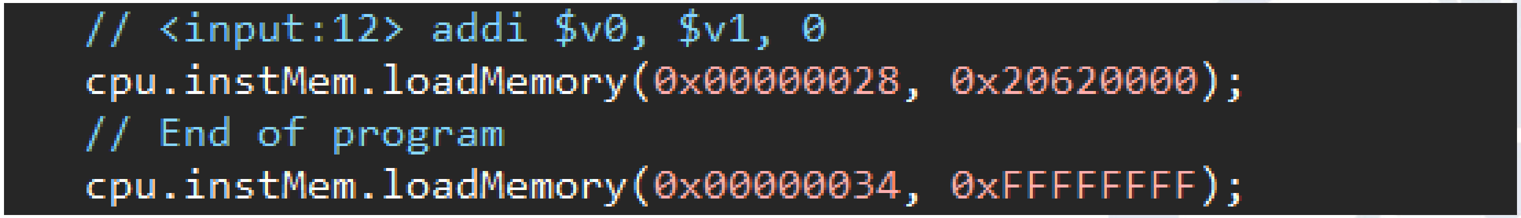
- Challenges:
- Input & Output
    - Output: the result of your program should be save into $v0 register



```
// <input:12> addi $v0, $v1, 0
cpu.instMem.loadMemory(0x00000028, 0x20620000);
```

Additional Task: Run Program on Your Processor

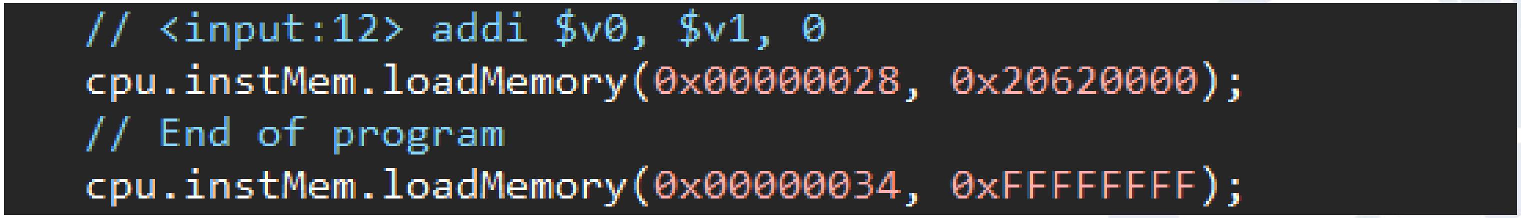
- Challenges:
- How to determine whether a program is terminated
  - At the end of program, you have to append an instruction which code encoding is 0xFFFFFFFF



```
    // <input:12> addi $v0, $v1, 0
    cpu.instMem.loadMemory(0x00000028, 0x20620000);
    // End of program
    cpu.instMem.loadMemory(0x00000034, 0xFFFFFFFF);
```

Additional Task: Run Program on Your Processor

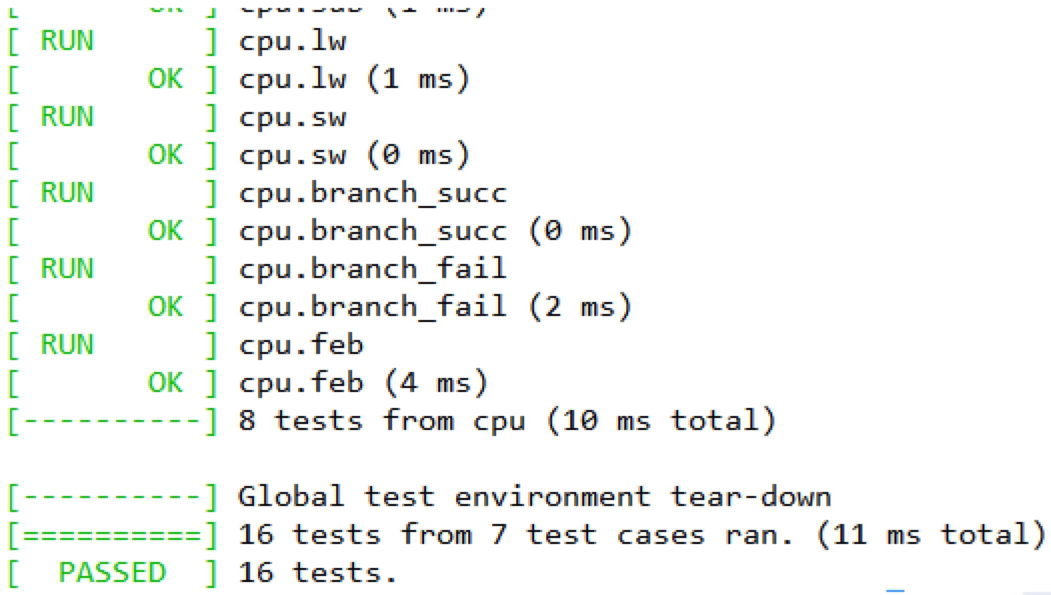
- Challenges:
- How to determine whether a program is terminated
  - When the instruction memory fetch an instruction with code 0xFFFFFFFF, the program terminated



```
// <input:12> addi $v0, $v1, 0
cpu.instMem.loadMemory(0x00000028, 0x20620000);
// End of program
cpu.instMem.loadMemory(0x00000034, 0xFFFFFFFF);
```

Project Evaluation

- We designed 16 test cases for MIPS processor. You will gain 5 point for each test cases you pass



```
[       OK ] cpu.sub (1 ms)
[ RUN      ] cpu.lw
[       OK ] cpu.lw (1 ms)
[ RUN      ] cpu.sw
[       OK ] cpu.sw (0 ms)
[ RUN      ] cpu.branch_succ
[       OK ] cpu.branch_succ (0 ms)
[ RUN      ] cpu.branch_fail
[       OK ] cpu.branch_fail (2 ms)
[ RUN      ] cpu.feb
[       OK ] cpu.feb (4 ms)
[----------] 8 tests from cpu (10 ms total)

[----------] Global test environment tear-down
[==========] 16 tests from 7 test cases ran. (11 ms total)
[  PASSED  ] 16 tests.
```

Project Evaluation Cont.

- All test cases are passed: 80%
- Report: 20%
- Each addition task: 20%

Useful resources

- EI209 course slides (very important!!)
  - Available on our website
- MIPS Instruction Reference
  - http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html
- Computer Organization and Design Experiment
  - http://jcube.sjtu.edu.cn/a/60ah9O534Do5L1Ub/252282a42d5ec3f63660f09eef28b2c68c90db1a
- Office Time
  - Every Tuesday afternoon 2 pm ~ 5 pm at SEIEE building 3-522

Thank You