Lab3: Single Cycle CPU

함형규

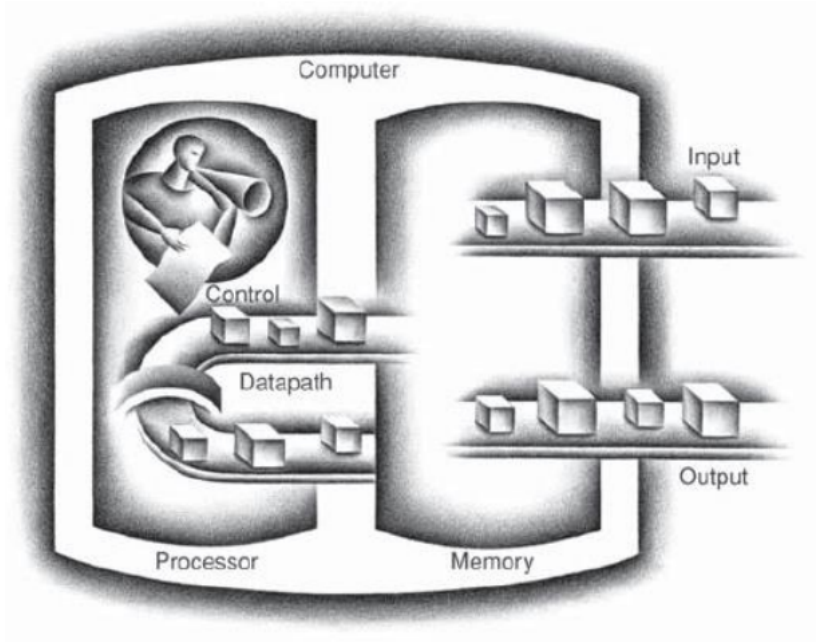
hhk971@postech.ac.kr

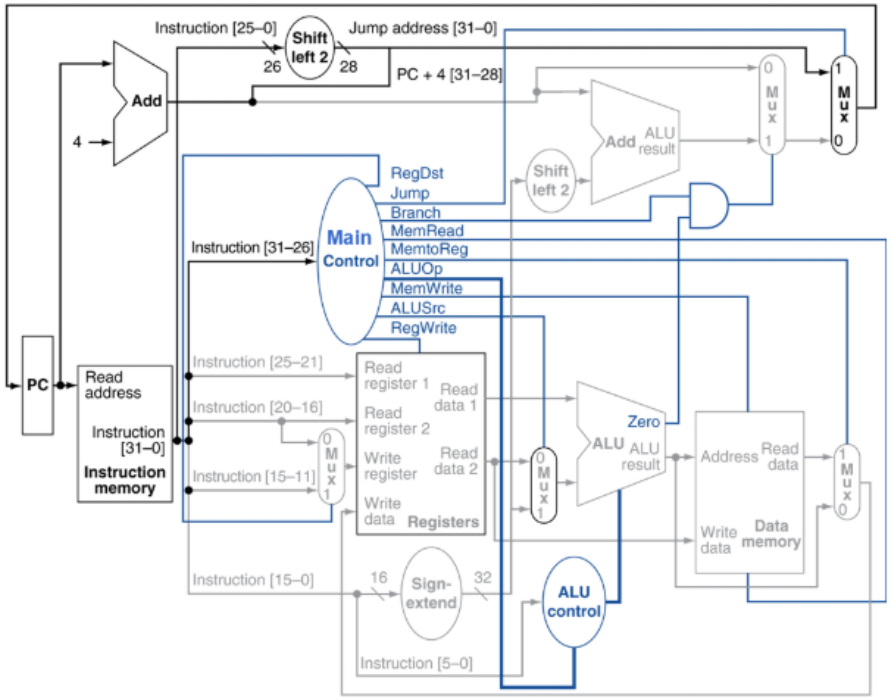
Contact the TAs at csed311-ta@postech.ac.kr

Contents

- CPU structure
- Modularization
- Magic memory
- TSC CPU
- Assignment
- Tips
- Q&A

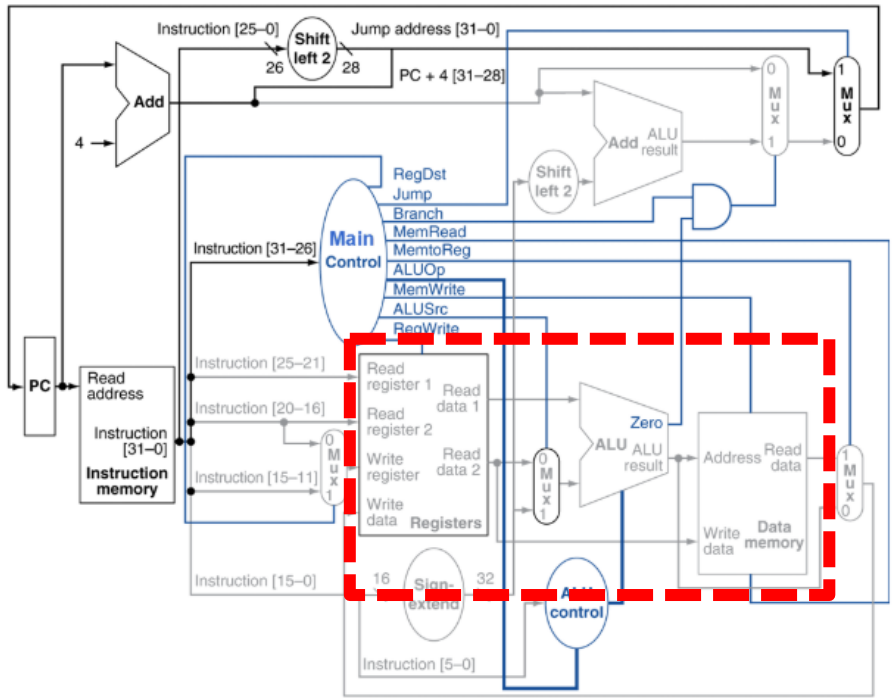
CPU Structure

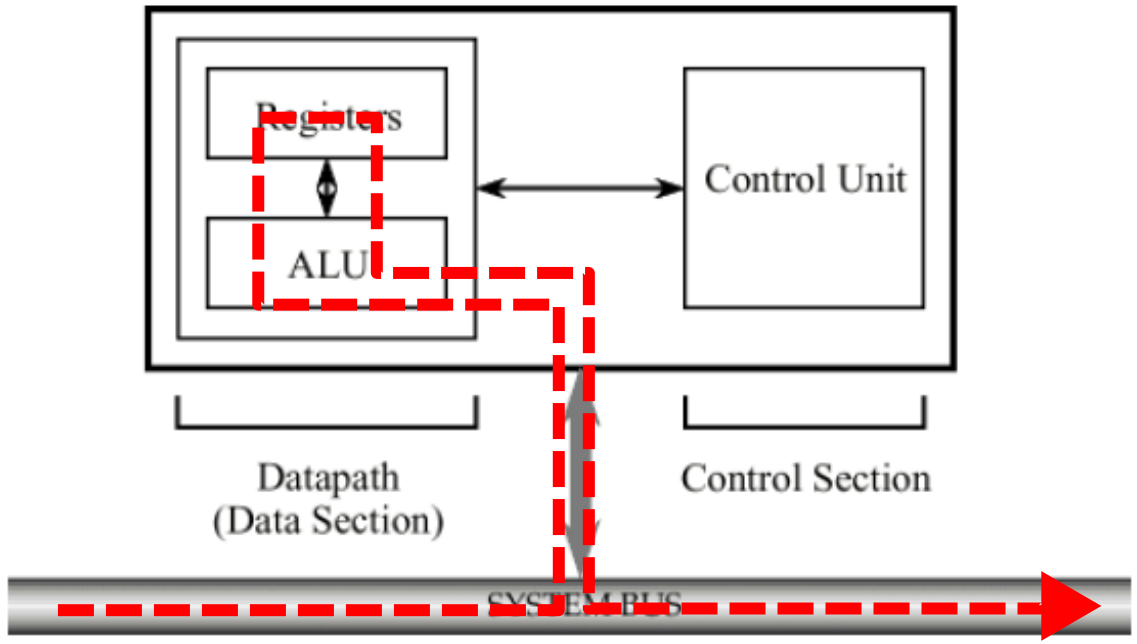




- CPU is an instruction processing FSM
- CPU consists of several components
  - Datapath
    - ALU, Register file, etc..
  - Control Unit

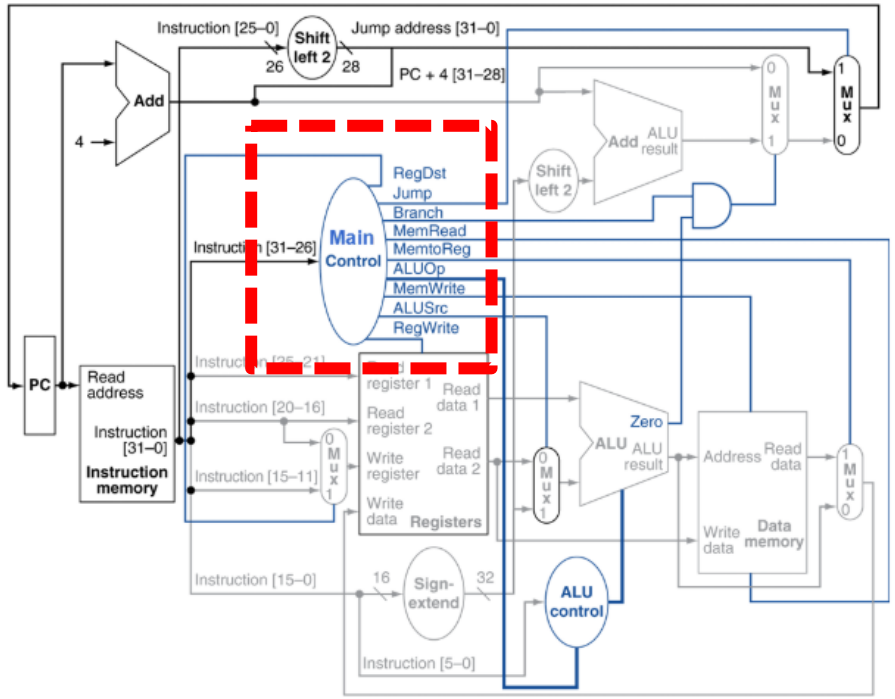
Datapath

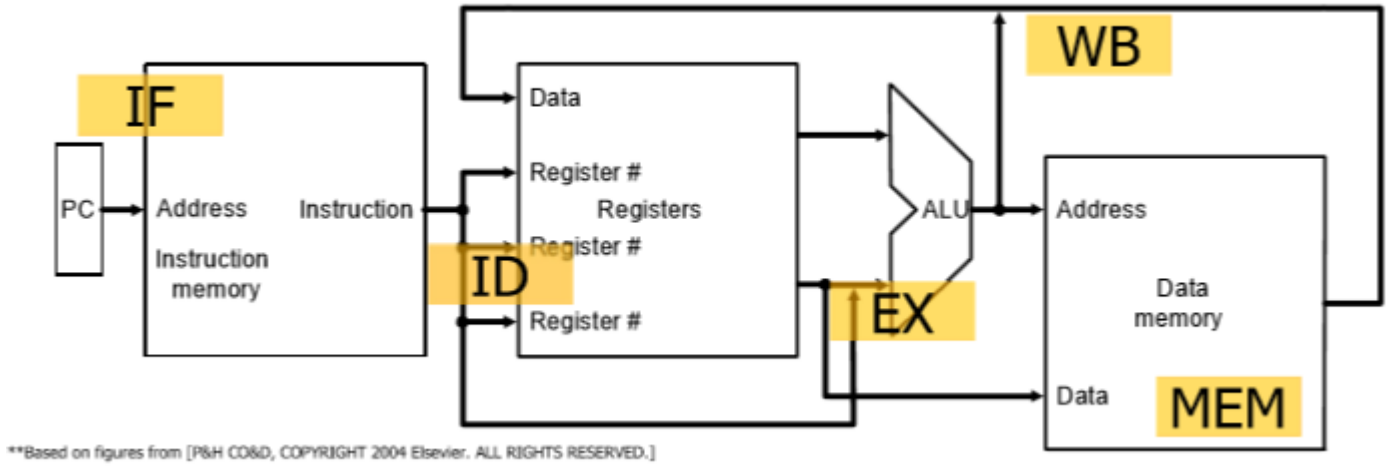




- Datapath:
  - Units in the path of data
    - SYSTEM BUS(deliver the memory data) → ALU → Register → ALU → SYSTEM BUS
  - ALU, Register file, and other data-wires

Control unit





**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

- Control Unit:
  - Decodes instruction
  - Then, generates control signals that control the components in the Datapath

Modularization

- You must modularize the main CPU components
  - Datapath
    - alu.v
    - register_file.v
  - Contorl Unit
    - contorl_unit.v
- Etc.
  - Modularize other components if you need
    - MUX, sign-extender, adder, etc..

Magic Memory

- It gives you memory data with a slight delay (less than a clock cycle)
  - In testbench code (line 38~58)

- It is NOT realistic model of the main memory
  - Memory is much slower than CPU
  - However, we assume there is a magic memory which is faster than CPU

```
always begin
    loadedData = `WORD_SIZE'bz;
    #`PERIOD1;
    forever begin
        wait (readM == 1 || writeM == 1);
        if (readM == 1) begin
            #`READ_DELAY;
            loadedData = memory[address];
            inputReady = 1;
            #(`STABLE_TIME);
            inputReady = 0;
            loadedData = `WORD_SIZE'bz;
        end else if (writeM == 1) begin
            memory[address] = data;
            #`WRITE_DELAY;
            ackOutput = 1;
            #(`STABLE_TIME);
            ackOutput = 0;
        end
    end  // of forever loop
end  // of always block for memory read
```

TSC CPU

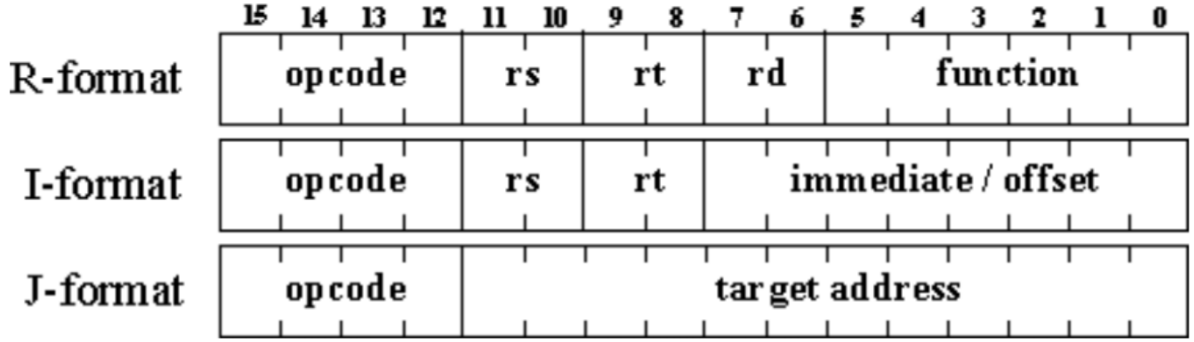
- RISC-V CPU vs TSC CPU

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
B-type*	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
J-type*	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

Remove

*in the textbook and old versions of the RISC-V manual, referred to as SB-type and UJ-type

[From P&H CO&D RISC-V ed.]



  - Has shorter instructions, smaller registers, and fewer instruction types!
    - A simple ISA
  - More details in TSC_manual.pdf

Assignment

- Implement a single-cycle TSC CPU
  - Your submission must include
    - cpu.v
    - alu.v
    - register_file.v (with 4 registers)
    - contorl_unit.v
  - Your implementation of the CPU should process instruction in each clock cycle
  - Due date : ~3/29 (2-weeks assignment)
  - Instructions
    - Implement all instructions in opcodes.v
    - You don’t need to implement instructions that not containing in opcodes.v (e.g : HLT, BNI, etc..)

Implementation Tips

- CPU module ports

output	readM	“read” signal to memory
output	writeM	“write” signal to memory
output	[`WORD_SIZE-1:0] address	target memory address
inout	[`WORD_SIZE-1:0] data	data for reading or writing (can be used as both **input and output**)
input	ackOutput	signal from memory (“data is written”)
input	inputReady	signal from memory (“data is ready for reading”)
input	reset_n	reset your CPU (registers, PC, etc..)
input	clk	clock signal

Implementation Tips (cont'd)

- About inout port
  - Can be used as both input and output port
  - Set wire value 'z' (high impedance) if you are using it as input
    - You could get the input value only if it's 'z' value
  - Google for more details

```
assign data= readOrWrite? writeData : 16'bz;
```

Implementation Tips (cont'd)

- Negedge clk
  - You might need 2 events in one clock cycle
    - For example, CPU fetches instruction and need one more memory data fetch for load store instruction
  - You can use both @(posedge clk) and @(negedge clk)
  - Do not use **Delay**

Report Tips

- Clearly distinguish between design and implementation
  - Design
    Describe how did you divide submodules and the role of the modules
  - Implementation
    Explain how did you implement each submodule
    Please do not just copy your Verilog code

Q&A