



Digital System Design

Homework 3

Hardware Implementation of Single Cycle MIPS

Instructor: 吳安宇教授

Date: 2018/04/19



Introduction to Single-cycle MIPS Processor

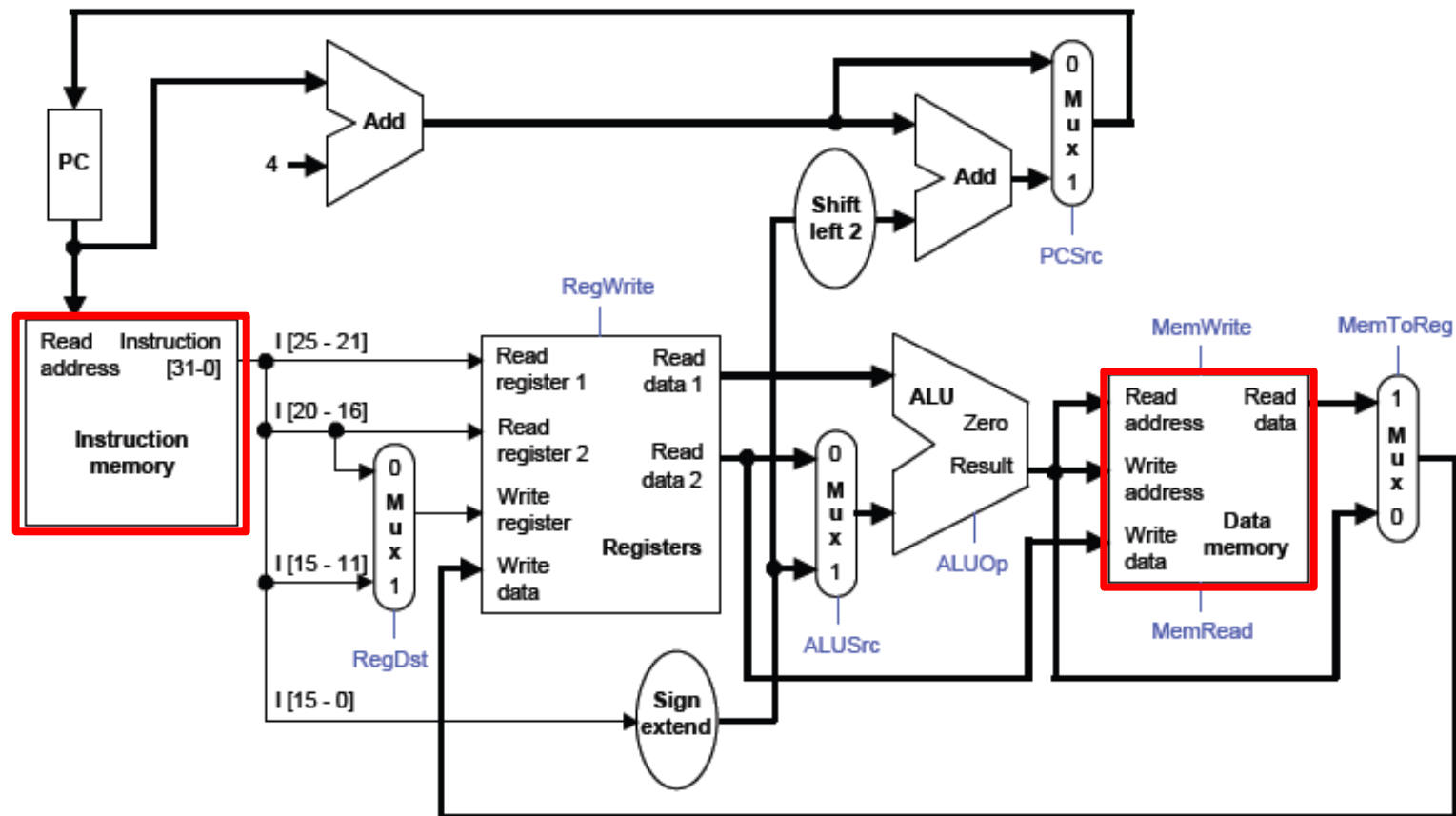


MIPS processor

- ❖ Any instruction set can be implemented in many different ways.
 - ❖ — In a basic **single-cycle** implementation all operations take the same amount of time—a single cycle.
 - ❖ — A **multicycle** implementation allows faster operations to take less time than slower ones, so overall performance can be increased.
 - ❖ — Finally, **pipelining** lets a processor overlap the execution of several instructions, potentially leading to big performance gains.



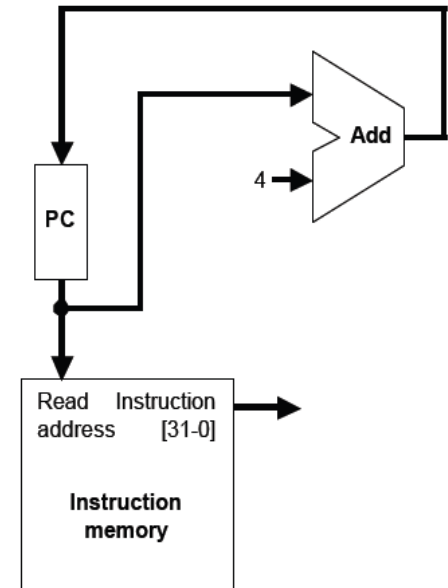
Datapath





Instruction fetching

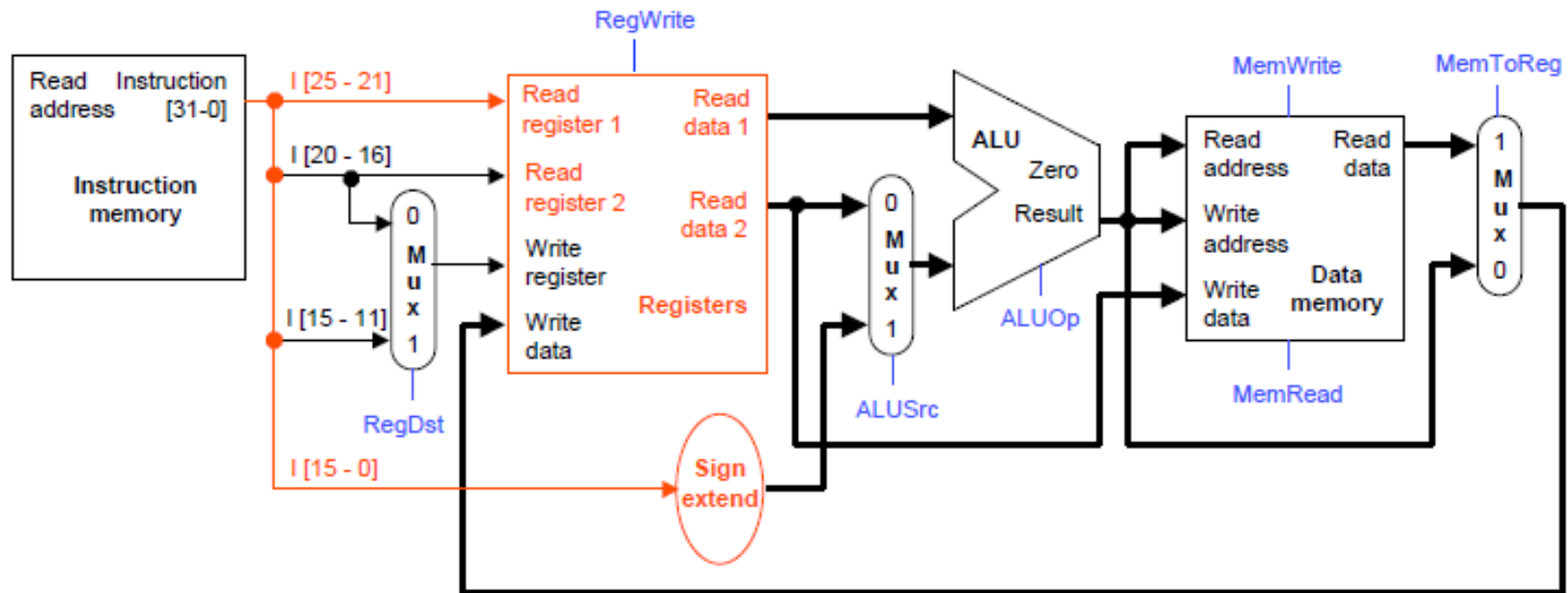
- ❖ The CPU is always in an infinite loop, fetching instructions from memory and executing them.
- ❖ The program counter or PC register holds the address of the current instruction.
- ❖ MIPS instructions are each four bytes long, so the PC should be incremented by four to read the next instruction in sequence.





Instruction Decode

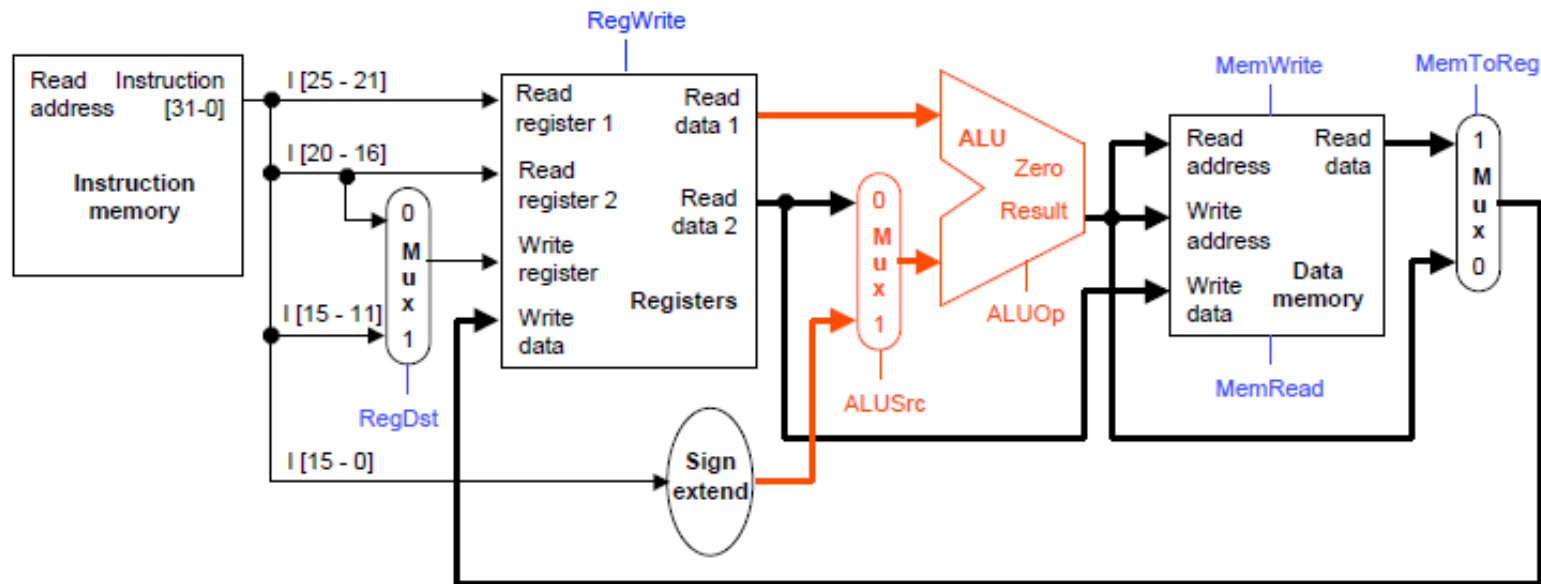
- ❖ The Instruction Decode (ID) step reads the source register from the register file.





Execute

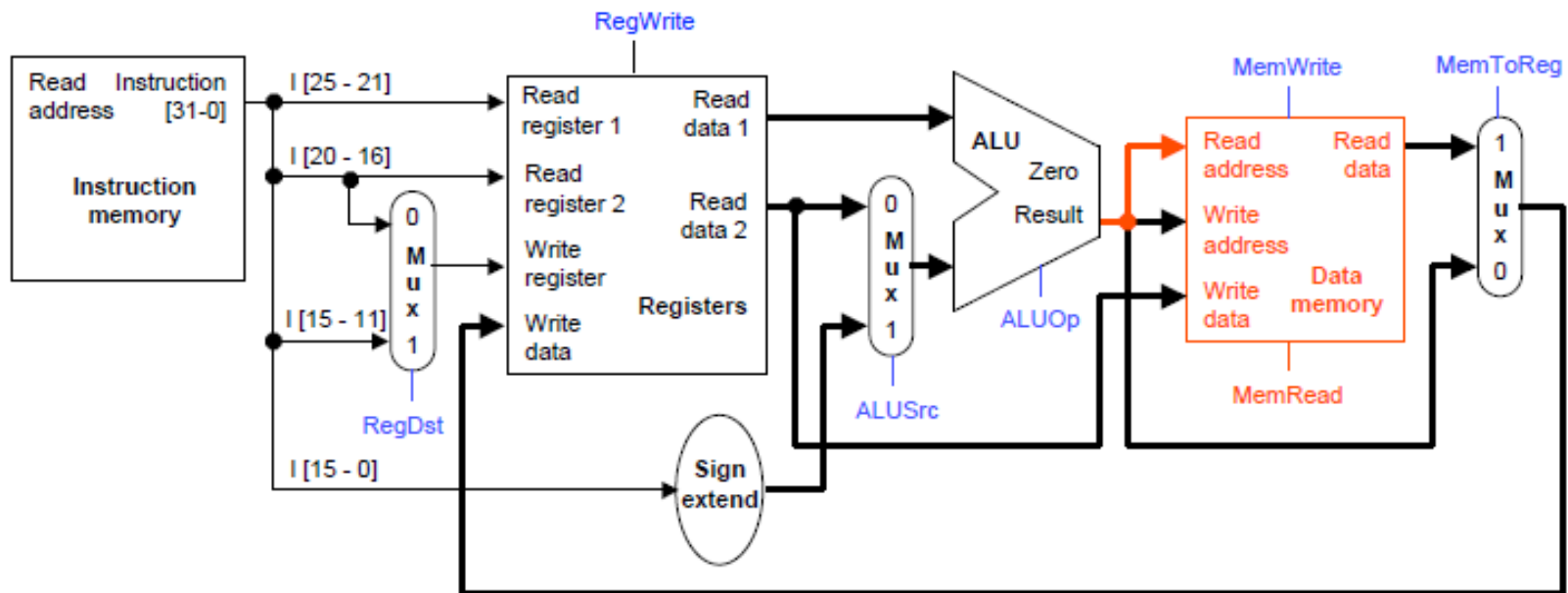
- ❖ The third step, Execute (EX), computes the effective memory address from the source register and the instruction's constant field.





Memory

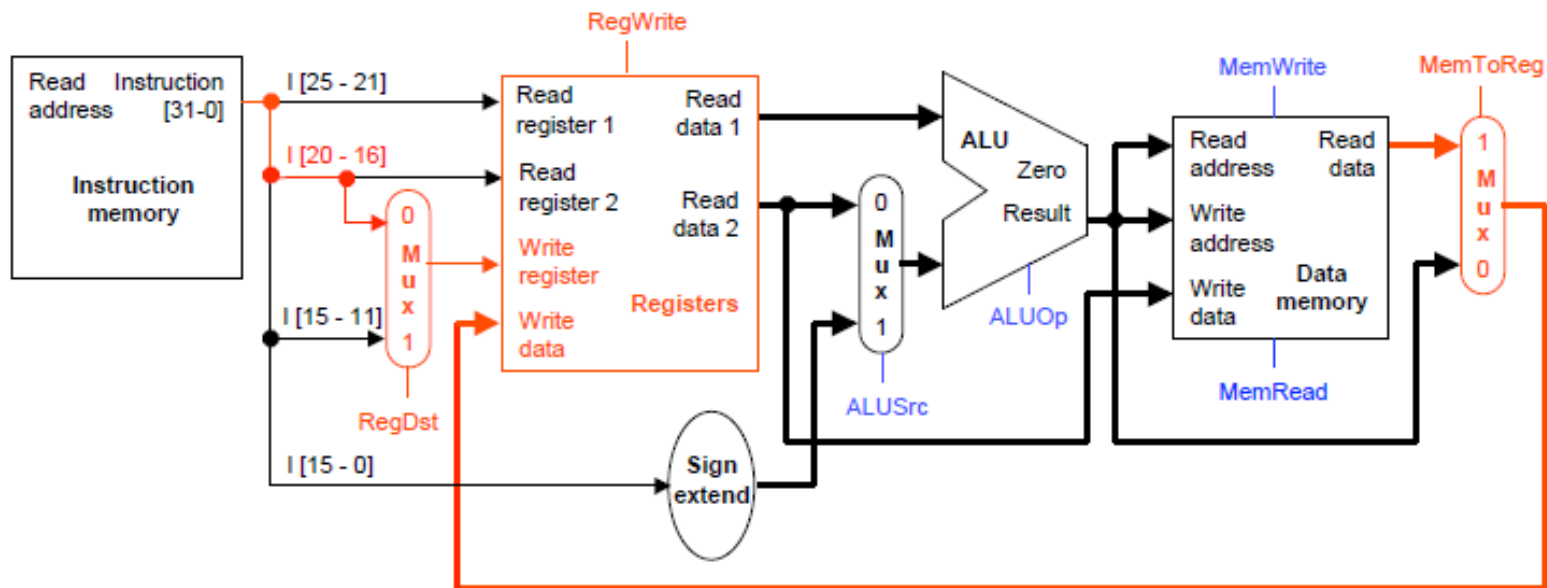
- ❖ The Memory (MEM) step involves reading the data memory, from the address computed by the ALU.





Write Back

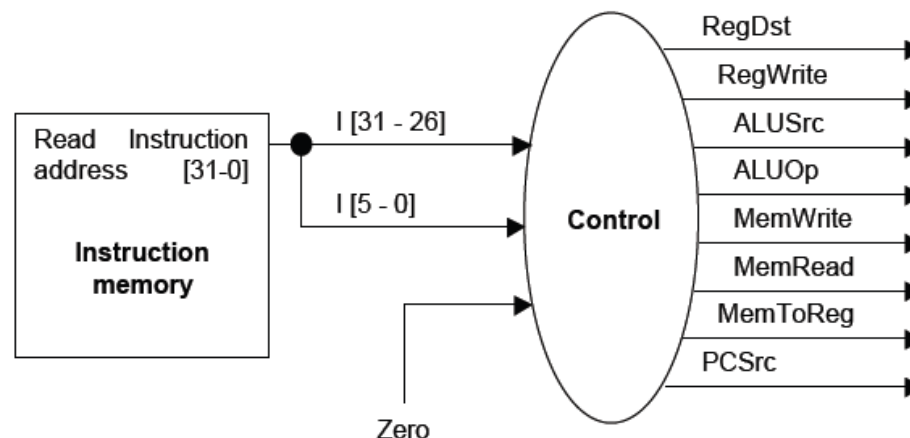
- ❖ Finally, in the Writeback (WB) step, the memory value is stored into the destination register





Control Unit

- ❖ The control unit needs 13 bits of inputs.
 - ❖ — Six bits make up the instruction's opcode.
 - ❖ — Six bits come from the instruction's func field.
 - ❖ — It also needs the Zero output of the ALU.
- ❖ The control unit generates 10 bits of output, corresponding to the signals mentioned on the previous page.





Homework 3

Single-cycle MIPS Processor

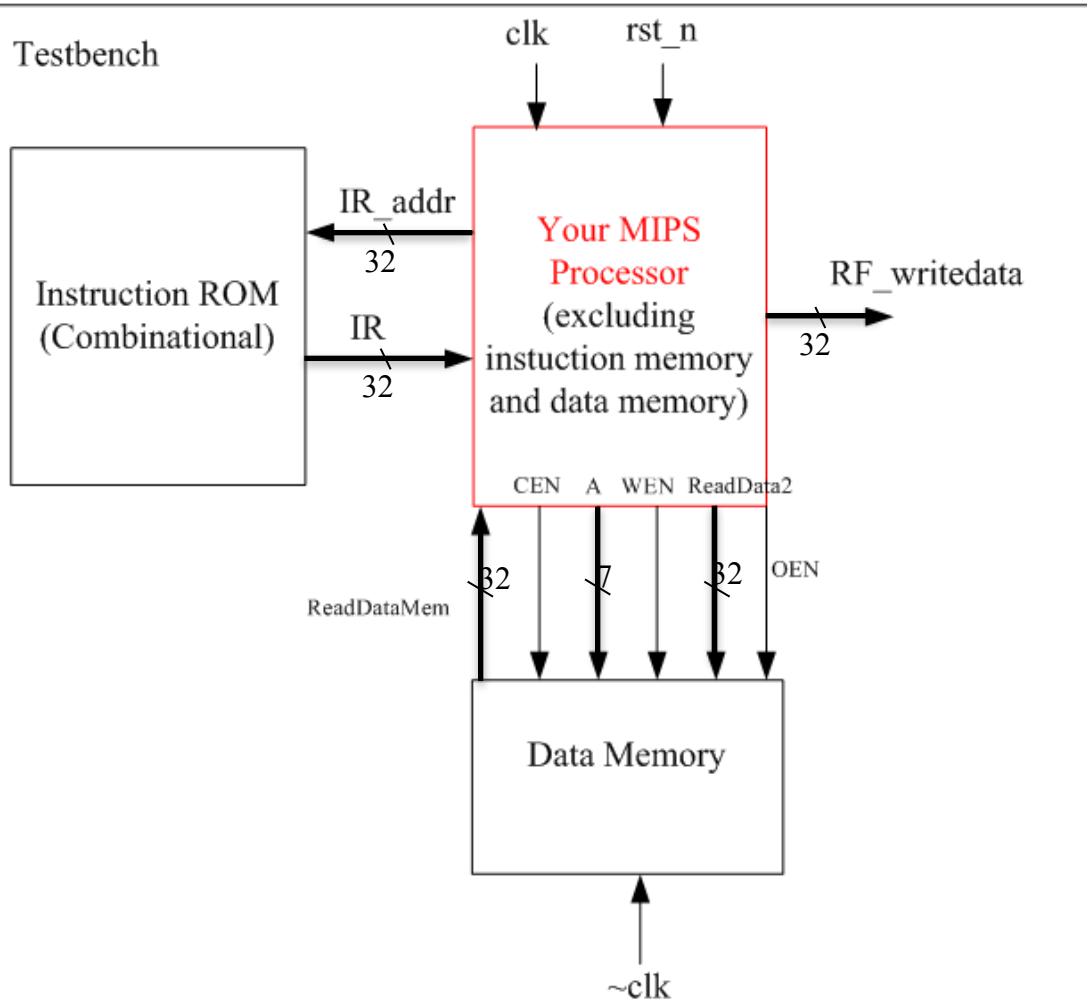


Problem Statement

- ❖ Using Verilog, implement the single-cycle MIPS processor:
 - ❖ Supported instructions:
 - add, sub, and, or, slt
 - lw, sw
 - beq
 - j, jal, jr
- ❖ Testbench/Memory model provided



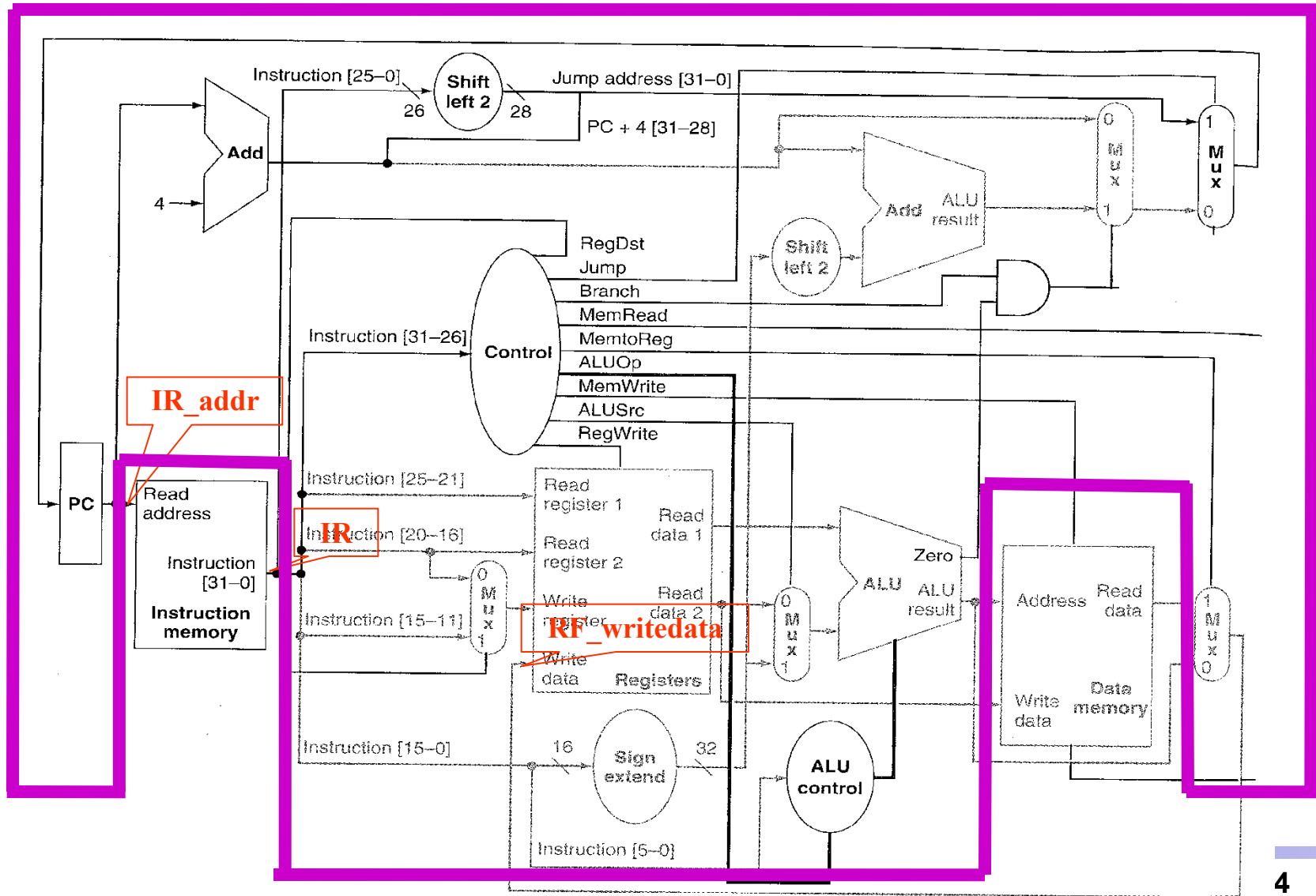
Block Diagram(1/2)



- ❖ **Instruction ROM**
contains the testing instructions
- ❖ **RF_writedata:**
the data to be written into the register file
❖ Used for testing your circuit
- ❖ **IR_addr:**
instruction address
- ❖ **IR:**
instruction



Block Diagram(2/2)





Testbench

- ❖ The testbench will
 - ❖ Initialize the instruction memory and the data memory
 - ❖ Reset your circuit
 - ❖ Execute the instructions, and read the values of *RF_writedata* and *IR_addr* to see if your circuit is correct
 - ❖ If your function is correct,

Congratulations!! Your design has passed all the test!!



Clock/Reset/Register File

- ❖ Clock: positive edge triggered
- ❖ Reset: active low asynchronous reset
- ❖ Register file
 - ❖ All registers are reset to 0 when reset occurs.
 - ❖ Register \$0 must be always 0



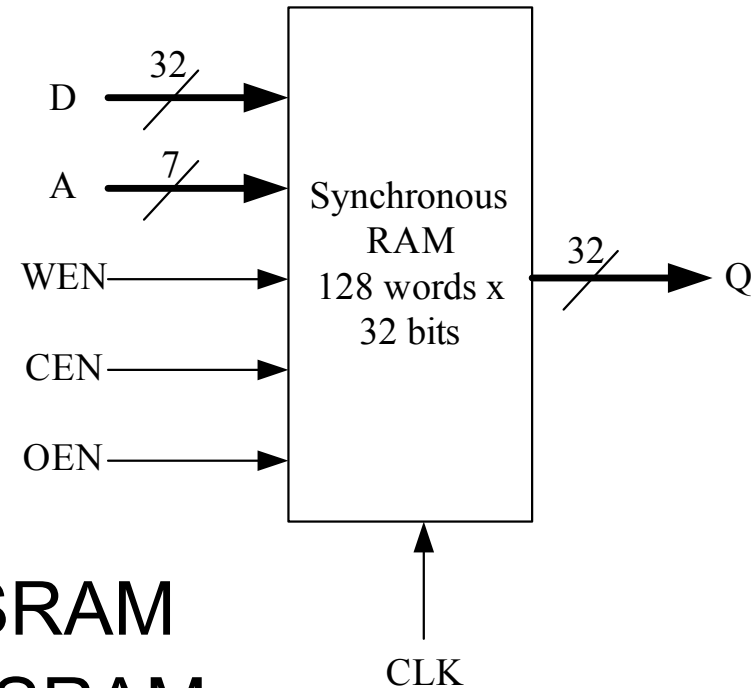
Memory

- ❖ Instruction memory and data memory are included in the testbench
- ❖ For the data memory, a memory module is provided in HSs18n_128x32.v
 - ❖ 128 words x 32 bits
 - ❖ Create an instance of HSs18n_128x32 as your data memory
 - ❖ The spec of the memory is described in HSs18n_128x32.pdf



HSs18n_128x32

- ❖ CLK: clock input
- ❖ D: data inputs
- ❖ A: address
- ❖ CEN: chip_enable,
0 when you read/write data
- ❖ WEN: write_enable,
0 when you write data into SRAM
1 when you read data from SRAM
- ❖ OEN: output_enable,
Always 0 in this case
- ❖ Q: data outputs





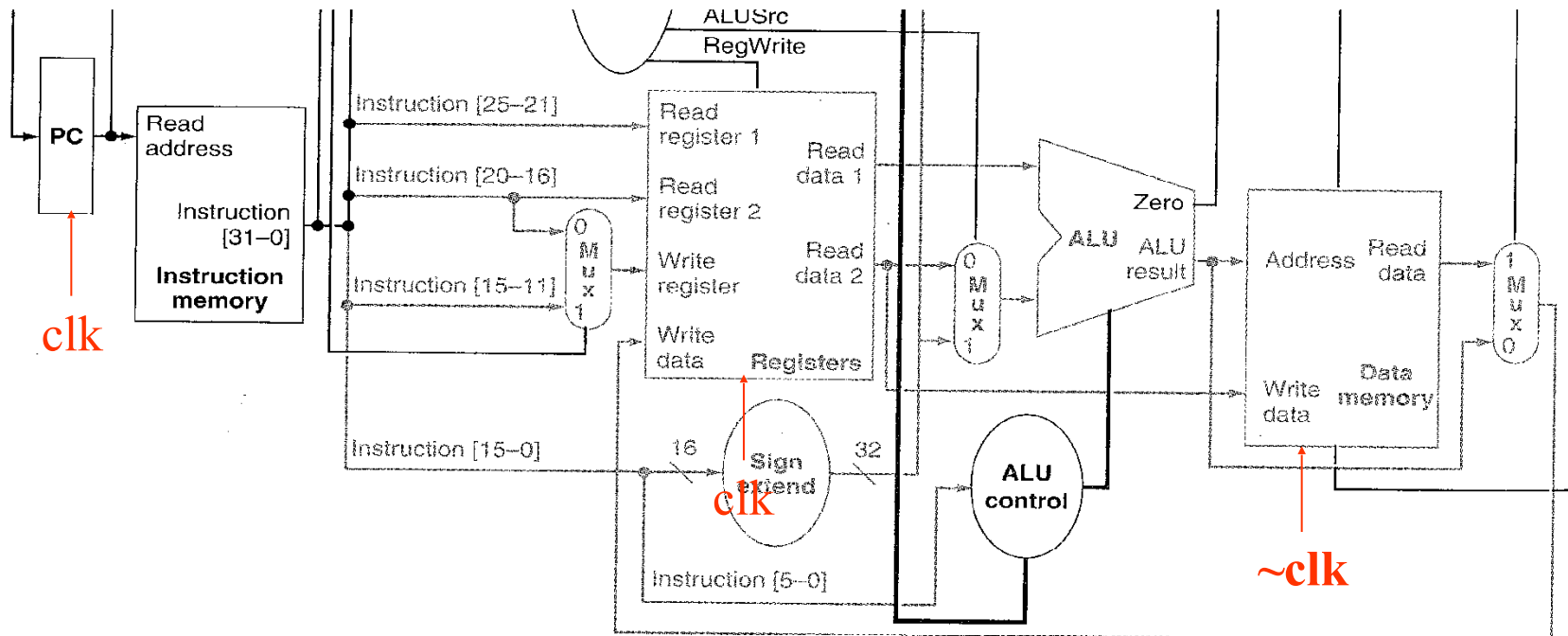
Memory Addressing

- ❖ In MIPS, the memory address is byte address.
- ❖ In HSs18n_128x32, the memory address is word address.
 - ❖ We can just discard the least significant two bits of the address input
- ❖ The address input of HSs18n_128x32 is 7-bit wide.
 - ❖ More significant bits of address inputs are also discarded



Memory Clock

- ❖ To accomplish a load in a cycle, you need to connect the clock of the memory to the inverse of the main clock





Simulation & Synthesis

- ❖ Check “`how_to_invoke_tools.txt`”
- ❖ 3 Major Things
 - ❖ RTL coding & simulation
 - ❖ Logic Synthesis
 - ❖ Gate-level simulation & debugging/refinement
- ❖ Files needed for simulation
 - ❖ RTL code: `DSDHW3.v`
 - ❖ Gate-level code: `DSDHW3.vg`
 - ❖ Timing information (SDF file): `DSDHW3.sdf`
 - ❖ Design library (DDS file): `DSDHW3.ddc`



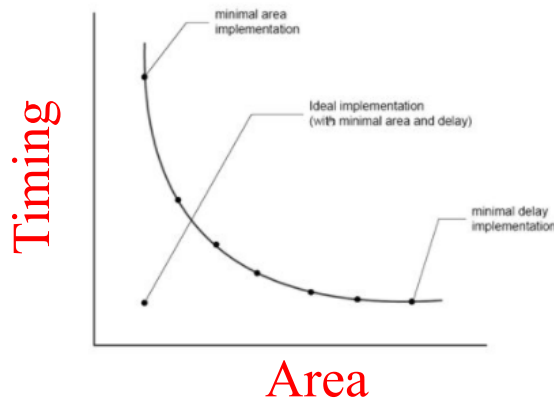
Grading Policy

- [illegible]



Report

- ❖ ***Simulated*** timing (ns)
 - ❖ Gate-level simulation clock cycle
 - ❖ Not specified synthesis clock cycle
- ❖ Area(μm^2)
 - ❖ report_area
- ❖ Cost($A \cdot T$)



```
Number of ports:      172
Number of nets:       367
Number of cells:      130
Number of combinational cells: 125
Number of sequential cells: 0
Number of macros:     0
Number of buf/inv:    39
Number of references: 22

Combinational area:   43665.613947
Noncombinational area: 32960.112083
Net Interconnect area: undefined (No wire load specified)

Total cell area:      76625.726031
Total area:           undefined
```



Submission

- ❖ Please submit 5 files (follow the name)
 - ❖ RTL code: DSDHW3.v
 - ❖ Synthesis: DSDHW3.vg, DSDHW3.sdf, DSDHW3.ddc
 - ❖ Report: DSDHW3.pdf
- ❖ Compress all the files into one ZIP or RAR file
 - ❖ File name: DSD_HW3_學號_姓名.zip
 - ❖ EX: DSD_HW3_B03901001_王小明.zip
- ❖ Upload the file to Ceiba
- ❖ Deadline: 2018/05/10 24:00
 - ❖ Late submission not allowed