Computer Organization 2018

Homework 2 ALU Decoder

Deadline: 2018/5/14 11:55PM

Overview

The goal of this homework is to help you understand how a single-cycle MIPS work and how to use Verilog hardware description language (Verilog HDL) to model electronic systems. In this homework, you need to implement ALU and decoder module and make your codes be able to execute *add*, *sub* and *slt* instructions. You need to follow the instruction table in this homework and satisfy all the homework requirements. In addition, you need to verify your CPU by using Modelsim.

General rules for deliverables

- Please use **Modelsim** as the simulation platform. No team work.
- Files TestBench.v, PC.v, CPU.v, IM.v, DM.v, Reg.v, ALU.v, ALU_ctrl.v, MUX_2_to_1.v, sign_extend.v are provided.
- You need to submit one report to answer questions and also the source codes.
- When submitting your homework, compress all files into a single **zip** file, and upload the compressed file to Moodle.
  - Please follow the file hierarchy shown in Figure 1.

F740XXXXX (your id)(folder)

SRC (folder) * Store your source code

F740XXXXX_Report.docx (Project Report. The report template is already included. Follow the template to complete the report.)

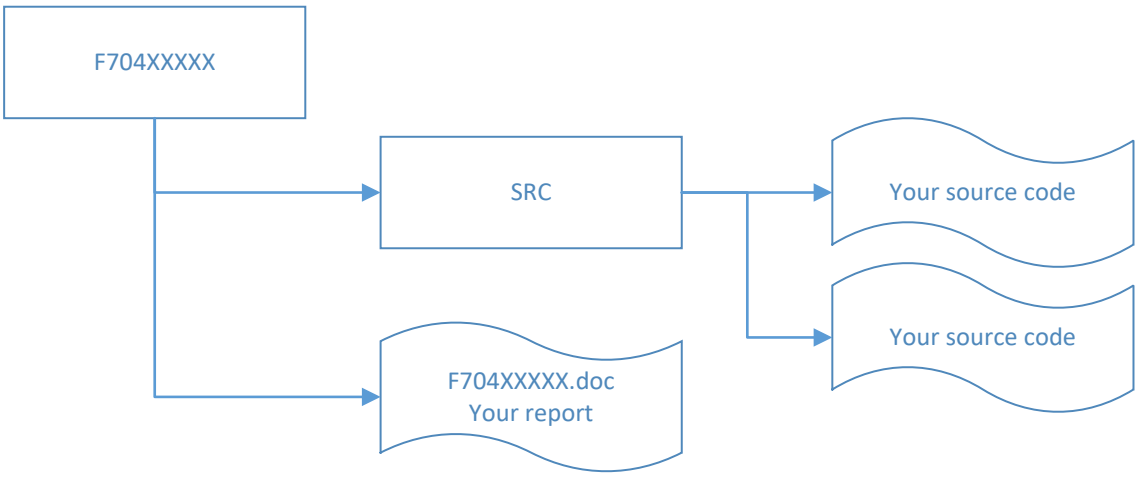


Figure 1. File hierarchy for homework submission

Instruction format:

instr	op [25:20]	rs [19:17]	rt [16:14]	rd[13:11]	Shamt[10:6]	Func[5:0]
add	6'b000000	SSS	TTT	DDD	5'b00000	6'b100000
sub	6'0000000	SSS	TTT	DDD	5'b00000	6'b100010
slt	6'b000000	SSS	TTT	DDD	5'b00000	6'b101010

add: do R[SSS]+R[TTT] and store in R[DDD]
sub: do R[SSS]-R[TTT] and store in R[DDD]
slt: do if(R[SSS]<R[TTT]) then R[DDD]=1 else R[DDD]=0

Homework Description

- **Module**

a .TestBench module

"TestBench" is not a part of CPU, it is a file that controls all the program and verify the correctness of our CPU. The main features are as follows: send periodical signal CLK to CPU, set the initial value of Reg and DM, print the value of Reg and DM at each cycle, end the program. The initial value are Reg[i]=i+1(0~7), DM[i]=i+1, (i=0~7).

※You don't need to modify this module

b. CPU module

"CPU"is the outmost module. It is responsible for connecting wires between modules.

Here are the wires:

- *PC* represents Program Counter
- *Instr* represents the instruction to be executed in this cycle
- *OP* represents the OP column of the instruction.
- *Reg_WE* represents the signal whether the data should be wrote in Reg.
- *RS_ID* represents RS column in MIPS instruction
- *RT_ID* represents RT column in MIPS instruction
- *Reg_RData1* represents the data read from Reg
- *Reg_RData2* represents the data read from Reg
- *Reg_WData* represents the data ready to be wrote in Reg
- *DM_RData* represents the data read from DM
- *DM_WData* represents the data ready to be wrote in DM
- DM_WE represents the signal whether the data should be wrote in DM.
- *address* represents the value out of Adder.
- *immediate_in* represents the immediate value read from instruction
- *func* represents the Func in MIPS instruction.
- *ALU_OP* represent the control signal from Decoder module to ALU_ctrl module.

- *ALU_CTRL* is the output signal of the ALU_ctrl module and the input signal of the ALU module in order to tell the ALU module which operation to do.
- *ALU_result* represent the result of the ALU module.
- *ALU_src* is used to choose which data should be the input of ALU(the sign_extend_module output or Reg_RData2).
- Mux_to_ALU is the MUX_2_to_1 module output. It is used to selected output from the sign_extend_module output or Reg_RData2 according to the input you provide.
- MEM_to_REG: If this signal is one then the data is from memory to register; if is zero, data from ALU will be written to register.
- REG_Dst is used to choose which register ID should be written(RT or RD)
- REG_W_ID is the register ID that should be written.
- ALU_src1, ALU_src2 are the inputs of ALU.
- Mux_Mem_to_reg_out are the MUX output which should be write to register

※You should modify this module and connect wires between each module.

c. PC module

"PC" is the abbreviation of "Program Counter". It is responsible for controlling next instruction ready to be executed. PC will plus one in the end of every cycle.

※You do not need to modify this module

d. IM module

"IM" is the abbreviation of "Instruction Memory". This module saves all the instructions and send instruction to CPU according to PC.

※You do not need to modify this module

d. Reg module

"Reg" is Register module. It will read or write data according to input signals. The name of wires please refer to CPU module.

※You need to change this module. You need to change the register ID that you need to write data in.

e. DM module

Same as Reg, but the target here is Data Memory.

Real memory is continuous spaces using byte as unit, but in order to easily implement this lab, we use what we use in Reg to simulate DM. That is using 16 bits as a unit to represent memory and use "grid" (DM[0] is a grid, DM[1] is a grid) as continuous spaces.

※You do not need to modify this module

f. Decoder module

Decoder is the module decode signal according to OP code that passed in. Signals then are send to Reg or DM for them to know whether this instruction should do read or write.

For example, if OP is 6'b100011, then Decoder should set Reg_WE=1, because this OP code

represents data will be wrote into Reg.

Hint 1.ALU_OP is use to distinguish the R-type(add sub slt) instruction and I-type instruction (load store)

2. MEM_to_REG is use to distinguish the data is form memory to register or from ALU
3. REG_Dst is use to choose the register ID which need to be written (RT or RD)
4. ALU_src is use to choose the input to ALU (sign_extend ouput or Reg_RData2)

※You need to modify this module.

Reset the Reg_WE and DM_WE according OP code and funct

And set ALU_src, MEM_to_REG, REG_Dst, ALU_OP according OP and funct

g. ALU_ctrl module

input ALU_OP and funct and use these two input to ouput ALU_CTRL.

※Things you need to modify: set ALU_CTRL according to ALU_OP and funct

h. MUX_2_to_1 module

choose the output according to select signal. This lab has three MUX.

MUX_REG_Dst, MUX_ALUsrc, MUX_MEM_to_REG

※Things you need to modify:

choose the output according to select signal. This lab has three MUX.

i. sign_extend module

extend the immediate_in(14 bits) to 16bits but don't change it's sign

※Things you need to modify

extend the immediate_in (14 bits) to 16bits but don't change its sign

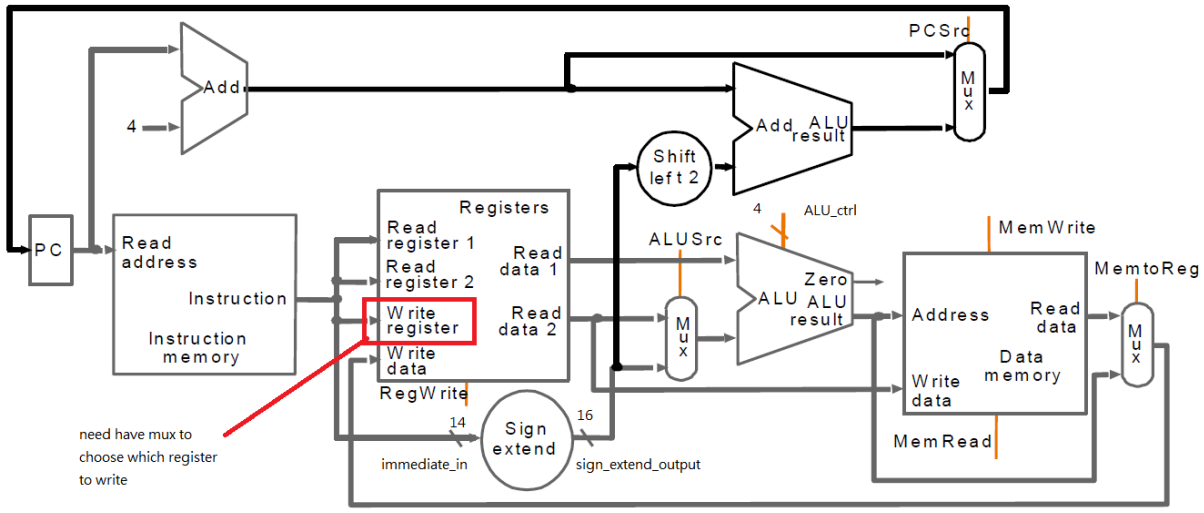
j. ALU module

do the operation according to ALU_CTRL from ALU_ctrl module

※Things you need to modify:

do the operation according to ALU_CTRL from ALU_ctrl module\

reference graph



test instruction(test1.txt)

DM[0]~DM[7] are initialized 1~8,R[0]~R[7] are initialized 1~8.

00000000101001100000100000	R[1]+R[2]->R[3]
00000010101011100000100010	R[5]-R[2]->R[7]
00000010011001000000101010	R[4]<R[6] R[2]=1

Simulation result:

After 1th cycle							
reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	**3**	5	5	6	7	8
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	4	5	6	7	8

After 2th cycle							
Reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	3	5	5	6	7	3
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	4	5	6	7	8

After 3th cycle							
Reg[0]	Reg[1]	Reg[2]	Reg[3]	Reg[4]	Reg[5]	Reg[6]	Reg[7]
1	2	1	5	5	6	7	3
DM[0]	DM[1]	DM[2]	DM[3]	DM[4]	DM[5]	DM[6]	DM[7]
1	2	3	**4**	5	6	7	8

Homework Requirements

1. Complete the Single cycle CPU that can execute add, sub, and slt instructions from the *MIPS ISA* section.
2. Verify your CPU with the benchmark and take a snapshot (e.g. Figure 6)

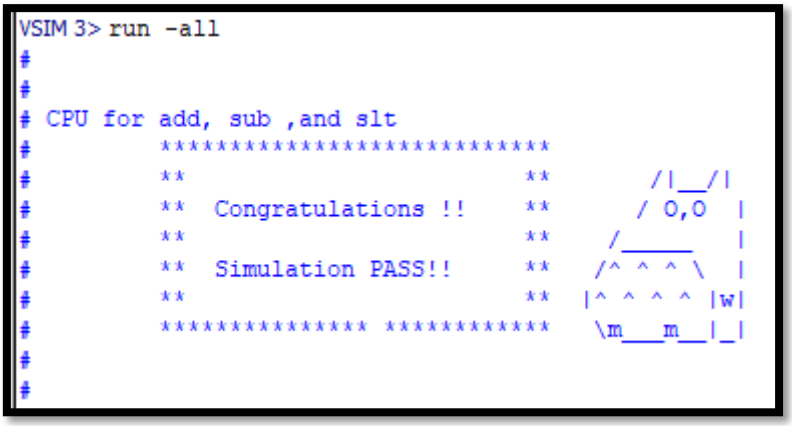


Figure 2. **S**napshot of correct simulation

   a. Using **waveform** to verify the execute results.
   b. Please annotate the waveform

3. Finish the Project Report.
   a. Complete the project report. The report template is provided.

> **Important**
>
> When you upload your file, please make sure you have satisfied all the homework requirements, including the **File hierarchy**, **Requirement file** and **Report format**.
>
> If you have any questions, please contact us.