**Computer Organization 2018**

**Homework 2 ALU Decoder**

**Deadline:**

**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

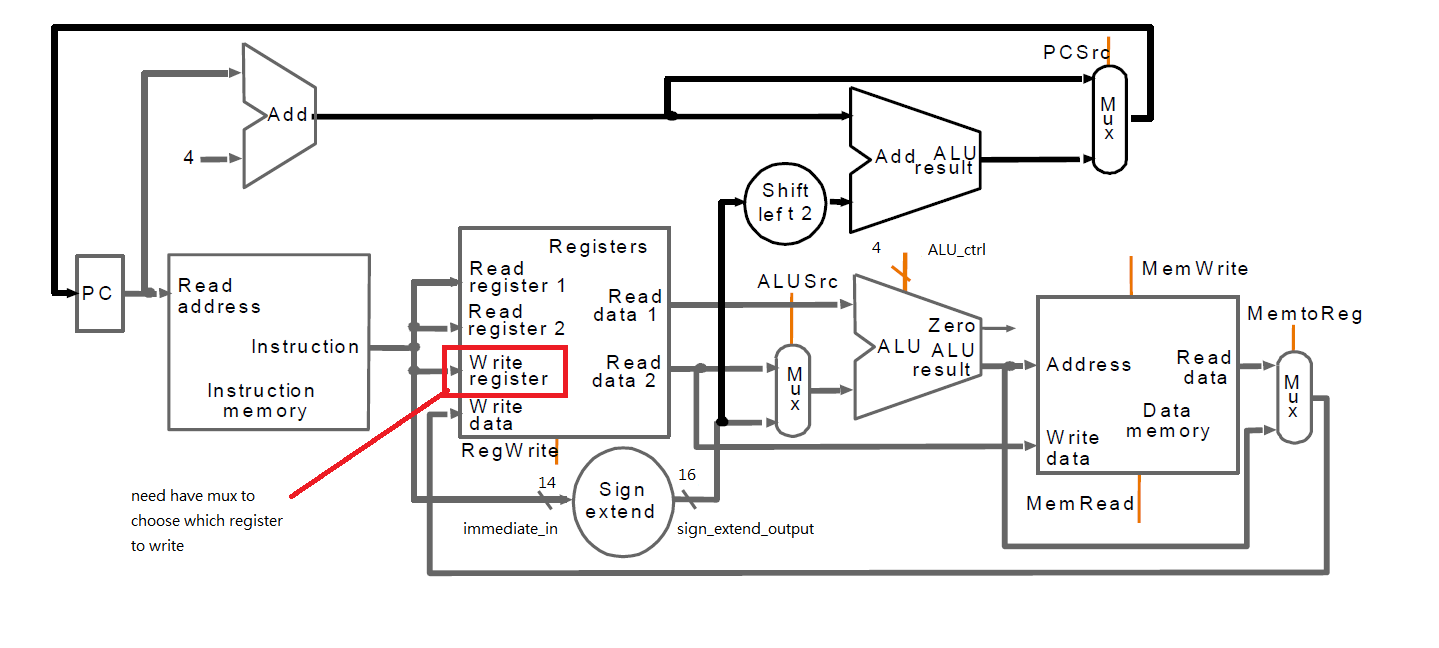
**j. ALU module**

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 6. **S**napshot of correct simulation |

* 1. Using **waveform** to verify the execute results.
  2. Please annotate the waveform

1. Finish the Project Report.
   1. 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. |