Assignment No. 8
Computer Organisation - CS220

Problem Statement: We want to create a new processor IITK-Mini-MIPS that has the instruction set architecture as shown in Tab.1 below. We assume that the processor word size and instruction size are 32 bits. The processor has two different memories: a) Instruction Memory (for storing instructions), b) Data Memory (for storing data). You have to make sure that these two sections do not overlap. We also assume that IITK-Mini-MIPS has a total of 32 general purpose registers and 32 floating point registers, of which certain registers follow the same roles as in MIPS-32 ISA. For example, Program Counter (PC) register holds the address of the next instruction. Note that movement from floating point registers to general purpose registers is only allowed through $mfc1$ and $mtc1$ instruction.

The target is to create the op-code formats for the given ISA, shown in Table 1, decide upon the data path elements (such as addition, subtraction, shift, jump) and then develop different verilog modules for the same along with the finite state machine for the control path.

Finally we shall build a single-cycle instruction execution unit for IITK-MIni-MIPS.

Processor Development Strategy:

Now to do the project, we shall follow the below-mentioned steps and every step should be documented in the project report.

1. **[PDS1]** Decide the registers and their usage protocol.
2. **[PDS2]** Decide upon the size for instruction and data memory.
3. **[PDS3]** Design the instruction layout for R-, I- and J-type instructions and their respective encoding methodologies.
4. **[PDS4]** Now design and implement an instruction fetch phase where the instruction next to be executed will be stored in the instruction register.
5. **[PDS5]** Next, design and implement a module for instruction decode to identify which data path element to execute given the opcode of the instruction.
6. **[PDS6]** Design and implement the Arithmetic Logic Unit (ALU) in top-down approach to develop different modules for different types of instructions. There might be some hardware parts in the ALU that can be shared by different modules if required. This will reduce the footprint of the hardware.
7. **[PDS7]** Design and implement the branching operation along the ALU implementation.
8. **[PDS8]** Design the finite state machine for the control signals to execute the processor. Please ensure that every instruction should be executed in single clock cycle. Finally write the test benches to simulate the IITK-Mini-MIPS.
9. **[PDS9]** Develop the MIPS code for *Bucket Sort*. Then convert it into machine code following the ISA given above.
10. **[PDS10]** Store the machine code in the instruction memory and execute IITK-Mini-MIPS. Store the output of the bucket sort in the data memory.

Project Progress Plan:

- **Week of 24th March and 26th March:** PDS1, PDS2, PDS3, PDS4, PDS5.
- **Week of 31st March and 2nd April:** PDS6, PDS7

- **Week of 7th April and 9th April:** PDS8
- **Week of 14th Arpil and 16th April:** PDS9 and PDS10.

Submission Instruction:

- Create a zip file that will contain the verilog modules, test benches and the documentation or report in PDF format for the assignment. The name of the zip file will be $Assignment7\_RollNo1\_RollNo2.zip$.
- In every verilog source files and the testbench you should clearly mention your name and roll no.
- No hard-coding is allowed.
- It is recommended that you add comments in the code to increase readability.
- The TAs will be checking the progress of the project every week.
- If any group does not upload their codes, they will not get any marks even if they show the output in the lab.
- Do not copy from other group. You will get zero.
- The deadline to upload the code in HelloIITK is: 19th April, 2025.

Table 1: Instruction Set for CSE-BUBBLE

<table>
<tr><th>Class</th><th>Instruction</th><th>Meaning</th></tr>
<tr><td rowspan="9">Arithmetic</td><td>add r0, r1, r2</td><td>r0=r1+r2</td></tr>
<tr><td>sub r0, r1, r2</td><td>r0=r1-r2</td></tr>
<tr><td>addu r0, r1, r2</td><td>r0=r1+r2 (unsigned addition, not 2's complement)</td></tr>
<tr><td>subu r0,r1,r2</td><td>r0=r1-r2 (unsigned addition, not 2's complement)</td></tr>
<tr><td>addi r0,r1,1000</td><td>r0=r1+1000</td></tr>
<tr><td>addiu r0,r1, 1000</td><td>r0=r1+1000 (unsigned additon, not 2's complement)</td></tr>
<tr><td>madd r0,r1</td><td>r0*r1 added with the value in the concatenated registers lo and hi.</td></tr>
<tr><td>maddu r0,r1</td><td>unsigned version of madd</td></tr>
<tr><td>mul r0,r1</td><td>hi=z[63:32],lo=z[31:0], Z=r0*r1</td></tr>
<tr><td rowspan="11"></td><td>and r0,r1,r2</td><td>r0= r1 & r2</td></tr>
<tr><td>or r0,r1,r2</td><td>r0=r1 | r2</td></tr>
<tr><td>andi r0,r1, 1000</td><td>r0= r1 & 1000</td></tr>
<tr><td>ori r0,r1, 1000</td><td>r0= r1 | 1000</td></tr>
<tr><td>not r0,r1</td><td>ro=$r1$</td></tr>
<tr><td>xori, r0,r1,1000</td><td>r0=r1 xor 1000</td></tr>
<tr><td>xor, r0,r1,r2</td><td>r= r1 xor r2</td></tr>
<tr><td>sll r0, r1, 10</td><td>r0=r1<<10 (shift left logical)</td></tr>
<tr><td>srl r0, r1, 10</td><td>r0=r1>>10 (shift right logical)</td></tr>
<tr><td>sla r0, r1, 10</td><td>r0=r1<<10 (shift left arithmetic)</td></tr>
<tr><td>sra r0, r1, 10</td><td>r0=r1>>10 (shift right arithmetic)</td></tr>
<tr><td rowspan="3">Data transfer</td><td>lw r0,10(r1)</td><td>r0=Memory[r1+10] (load word)</td></tr>
<tr><td>sw r0,10(r1)</td><td>Memory[r1+10]=r0 (store word)</td></tr>
<tr><td>lui r0, 1000</td><td>r0[31:16]=1000</td></tr>
<tr><td rowspan="8">Conditional Branch</td><td>beq r0,r1,10</td><td>if(r0==r1) go to PC+4+10 (branch on equal)</td></tr>
<tr><td>bne r0,r1,10</td><td>if(r0!=r1) go to PC+4+10 (branch on not equal)</td></tr>
<tr><td>bgt r0,r1,10</td><td>if(r0>r1) go to PC+4+10 (branch if greater than)</td></tr>
<tr><td>bgte r0,r1, 10</td><td>if(r0>=r1) go to PC+4+10 (branch if greter than or equal)</td></tr>
<tr><td>ble r0,r1, 10</td><td>if(r0<r1) go to PC+4+10 (branch if less than)</td></tr>
<tr><td>bleq r0,r1, 10</td><td>if(r0<=r1) go to PC+4+10 (branch if less than or equal)</td></tr>
<tr><td>bleu r0,r1, 10</td><td>unsigned version of ble</td></tr>
<tr><td>bgtu r0,r1, 10</td><td>unsigned version of bgt</td></tr>
<tr><td rowspan="3">Unconditional Branch</td><td>j 10</td><td>jump to address 10</td></tr>
<tr><td>jr r0</td><td>jump to address stored in register r0</td></tr>
<tr><td>jal 10</td><td>ra=PC+4 and jump to address 10</td></tr>
<tr><td rowspan="3">Comparison</td><td>slt r0,r1,r2</td><td>if(r1<r2) r0=1 else r0=0</td></tr>
<tr><td>slti r0,r1,100</td><td>if(r1<100) r0=1 else r0=0</td></tr>
<tr><td>seq r0,r1,100</td><td>if(r1=100) r0=1 else r0=0</td></tr>
<tr><td rowspan="10">Floating point</td><td>mfcl r0, f0</td><td>r0=f0</td></tr>
<tr><td>mtc1 f0,r0</td><td>f0=r0</td></tr>
<tr><td>add.s f0,f1, f2</td><td>f0=f1+f2</td></tr>
<tr><td>sub.s f0,f1, f2</td><td>f0=f1-f2</td></tr>
<tr><td>c.eq.s cc f0, f1</td><td>the flag cc is set to one if f0=f1</td></tr>
<tr><td>c.le.s cc f0, f1</td><td>the flag cc is set to one if f0<=f1</td></tr>
<tr><td>c.lt.s cc f0, f1</td><td>the flag cc is set to one if f0<f1</td></tr>
<tr><td>c.ge.s cc f0, f1</td><td>the flag cc is set to one if f0>=f1</td></tr>
<tr><td>c.gt.s cc f0, f1</td><td>the flag cc is set to one if f0>f1</td></tr>
<tr><td>mov.s cc f0, f1</td><td>f0=f1</td></tr>
</table>