16-bit pipelined RISC Microprocessor

Write the verilog RTL description for a 16-bit pipelined RISC Microprocessor. The microprocessor is based on the MIPS architecture. Refer to the following for more information if required:


The design should use a five stage pipeline. Each pipeline stage operates over 2 clock cycles. You can clock your pipeline registers directly using the clock or a divide by 2 version of the clock. The interface to the chip is given in the following diagram.

- **IF**: Instruction Fetch: Send the program counter (PC) address to the instruction memory and read the instruction. Compute the next value of PC=PC +2. Determine the next PC value either using the increment above or using forwarded data from a branch or jump instruction. Timing diagram for the read access is shown below in Figure 2.
- **ID**: Instruction Decode: The register file is located in this stage. It consists of 8 16-bit registers R0-R7. This stage also contains the sign extension unit for sign-extending immediate fields in the instruction. Data is written from the writeback stage on the first rising edge of clock and read on the second rising edge of the clock. Branch decoding and address calculation are also done here. Data can be forwarded to this stage.
- **EX**: Execute: This is where all the functional units are located. Data can be forwarded to this stage. Data can be forwarded from this stage.
- **MEM**: Memory Access: Used to access the data memory for load/store instructions. Timing diagram for read is similar to IF read timing. For write, dump address and data on first rising edge. Turn on write on the falling edge, check ready on the next falling edge. If ready is 1, turn off write and remove address and data on next rising edge. Data can be forwarded from this stage.
- **WB**: Write Back: Write the data back into the register file on the first rising edge of clock.

![Figure 1](image-url)
The process should assume that a branch is always not taken, i.e., you can fetch the next instruction after a branch instruction. If you find in the decode stage that the branch is taken, then you will flush the IF stage, i.e., insert a NOP instead of the fetched instruction. Keep in mind the setup and hold restrictions of flip-flops and latches during the design phase. Insert delay elements when there is no logic between two flip-flops. This arises in situations when something is just passed from one pipeline register to the next without any processing e.g., the store data read in ID from registers is passed directly in the EX stage without any processing. There could be possible timing violation between the ID/EX pipeline register and EX/MEM pipeline register. As explained before your processor should be able to forward data as required between various stages to avoid data hazards. The pipeline should be able to stall for hazards that cannot be solved using data forwarding.

Figure 2

The process should assume that a branch is always not take, i.e. you can fetch the next instruction after a branch instruction. If you find in the decode stage that the branch is taken, then you will flush the IF stage, i.e. insert a NOP instead of the fetched instruction. Keep in mind the setup and hold restrictions of flip-flops and latches during the design phase. Insert delay elements when there is no logic between two flip-flops. This arises in situations when something is just passed from one pipeline register to the next without any processing e.g. the store data read in ID from registers is passed directly in the EX stage without any processing. There could be possible timing violation between the ID/EX pipeline register and EX/MEM pipeline register. As explained before your processor should be able to forward data as required between various stages to avoid data hazards. The pipeline should be able to stall for hazards that cannot be solved using data forwarding.

Instruction Formats:
5 bit opcode, plus last 2 bits in the opcode for different variants.
3 bit operand fields (Rd, Rs1, Rs2) to specify one of the 8 general purpose registers.
Immediates are restricted in length to the number of bit positions remaining in a 16 bit instruction word.
ALL instructions are 16 bits in length.
Shift immediates can be restricted to 2 bits.
<table>
<thead>
<tr>
<th>Name</th>
<th>Opcode 5:2 bits</th>
<th>Format</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>00000:00</td>
<td>SEQ Rd Rs1 Rs2 00</td>
<td>Set if Equal: Set register Rd to 1 if the values in Rs1 and Rs2 are equal, else set to 0.</td>
</tr>
<tr>
<td>SGT</td>
<td>00000:01</td>
<td>SGT Rd Rs1 Rs2 01</td>
<td>Set if Greater then: Set register Rd to 1 if Rs1 &gt; Rs2, else set to 0.</td>
</tr>
<tr>
<td>SLT</td>
<td>00000:10</td>
<td>SLT Rd Rs1 Rs2 10</td>
<td>Set if Less then: Set register Rd to 1 if Rs1 &lt; Rs2, else set to 0.</td>
</tr>
<tr>
<td>SNE</td>
<td>00000:11</td>
<td>SNE Rd Rs1 Rs2 11</td>
<td>Set if Not Equal: Set register Rd to 0 if the values in Rs1 and Rs2 are equal, else set to 1.</td>
</tr>
<tr>
<td>ADD</td>
<td>00001:00</td>
<td>ADD Rd Rs1 Rs2 00</td>
<td>Add registers Rs1 and Rs2 and store the result in Rd.</td>
</tr>
<tr>
<td>SUB</td>
<td>00001:01</td>
<td>SUB Rd Rs1 Rs2 10</td>
<td>Compute Rs1 - Rs2 and store the result in Rd.</td>
</tr>
<tr>
<td>NEG</td>
<td>00001:11</td>
<td>NEG Rd Rs1 000 11</td>
<td>Put 2’s complement of Rs1 into Rd</td>
</tr>
<tr>
<td>ADDI</td>
<td>00010</td>
<td>ADDI Rd Rs #</td>
<td>Store in register Rd the value Rs + the 5-bit sign-extended immediate.</td>
</tr>
<tr>
<td>SUBI</td>
<td>00011</td>
<td>SUBI Rd Rs #</td>
<td>Store in register Rd the value Rs - the 5-bit sign-extended immediate.</td>
</tr>
<tr>
<td>AND</td>
<td>00100:00</td>
<td>AND Rd Rs1 Rs2 00</td>
<td>Store in register Rd the bitwise AND of Rs1 and Rs2.</td>
</tr>
<tr>
<td>OR</td>
<td>00100:01</td>
<td>OR Rd Rs1 Rs2 01</td>
<td>Store in register Rd the bitwise OR of Rs1 and Rs2.</td>
</tr>
<tr>
<td>XOR</td>
<td>00100:10</td>
<td>XOR Rd Rs1 Rs2 10</td>
<td>Store in register Rd the bitwise XOR of Rs1 and Rs2.</td>
</tr>
<tr>
<td>NOT</td>
<td>00100:11</td>
<td>NOT Rd Rs 000 11</td>
<td>Store in register Rd the bitwise complement Rs.</td>
</tr>
<tr>
<td>ANDI</td>
<td>00101</td>
<td>ANDI Rd Rs #</td>
<td>Store in register Rd the bitwise AND of Rs and the 5-bit sign-extended immediate.</td>
</tr>
<tr>
<td>ORI</td>
<td>00110</td>
<td>ORI Rd Rs #</td>
<td>Store in register Rd the bitwise OR of Rs and the 5-bit zero-extended immediate.</td>
</tr>
<tr>
<td>SRA</td>
<td>00111</td>
<td>SRA Rd Rs 000 #</td>
<td>Shift Right Arith: Store in register Rd the sign-extended value of Rs shifted to the right by the 2-bit immediate. (An immediate of 0 does not shift the operand).</td>
</tr>
<tr>
<td>Name</td>
<td>Opcode</td>
<td>Format</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>SRL</td>
<td>01000</td>
<td>SRL Rd Rs 000 #</td>
<td>Shift Right Logic: Store in register Rd the zero-extended value of Rs shifted to the right by the 2-bit immediate.</td>
</tr>
<tr>
<td>SLL</td>
<td>01001</td>
<td>RLL Rd Rs 000 #</td>
<td>Shift Left Logic: Store in register Rd the zero-extended value of Rs shifted to the left by the 2-bit immediate.</td>
</tr>
<tr>
<td>LW</td>
<td>01010</td>
<td>LD Rd Rs #</td>
<td>Load the 16-bit value at memory address Rs + sign extended immediate into Rd. (You can assume that address is an even number).</td>
</tr>
<tr>
<td>SW</td>
<td>01011</td>
<td>SW Rd Rs #</td>
<td>Store to memory at address Rd+ sign extended immediate the value in Rs. (You can assume that address is an even number).</td>
</tr>
<tr>
<td>LBI</td>
<td>01100</td>
<td>LBI Rd #</td>
<td>Load Byte Immediate: Store the 8-bit sign-extended immediate into register Rd after sign extending it.</td>
</tr>
<tr>
<td>LBIU</td>
<td>01101</td>
<td>LBIU Rd #</td>
<td>Load Byte Immediate Unsigned: Write the 8-bit zero-extended immediate into register Rd.</td>
</tr>
<tr>
<td>LHI</td>
<td>01110</td>
<td>LHI Rd #</td>
<td>Load High Immediate: Write the 8-bit immediate into the upper 8 bits of register Rd and preserve the low order 8 bits.</td>
</tr>
<tr>
<td>LLI</td>
<td>01111</td>
<td>LLI Rd #</td>
<td>Load Low Immediate: Write the 8-bit immediate into the lower 8 bits of register Rd and preserve the higher order 8 bits.</td>
</tr>
<tr>
<td>BEQZ</td>
<td>10000</td>
<td>BEQZ Rs #</td>
<td>Branch on Equal to Zero: Set PC to PC + 8-bit sign-extended immediate if the value in register Rs is zero.</td>
</tr>
<tr>
<td>BNEZ</td>
<td>10001</td>
<td>BNEZ Rs #</td>
<td>Branch if Not Zero: Set PC to PC + 8-bit sign-extended immediate if the value in register Rs is non-zero.</td>
</tr>
<tr>
<td>SGE</td>
<td>10010:00</td>
<td>SGE Rd, Rs1, Rs2 00</td>
<td>Set if Greater than Equal: Set register Rd to 1 if Rs1 &gt;= Rs2, else set to 0.</td>
</tr>
<tr>
<td>SLE</td>
<td>10010:01</td>
<td>SLE Rd, Rs1, Rs2 00</td>
<td>Set if Less than Equal: Set register Rd to 1 if Rs1 &lt;= Rs2, else set to 0.</td>
</tr>
<tr>
<td>J</td>
<td>10011</td>
<td>J #</td>
<td>Jump: Set the PC to PC + 11-bit sign-extended immediate.</td>
</tr>
<tr>
<td>Name</td>
<td>Opcode 5:2 bits</td>
<td>Format</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>--------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>JR</td>
<td>10100</td>
<td>JR 000 Rs</td>
<td>Jump to Register: Set the PC to the value in register Rs.</td>
</tr>
<tr>
<td>JAL</td>
<td>10101</td>
<td>JAL #</td>
<td>Jump and link: Set the PC to PC + 11-bit sign-extended immediate. Save next PC i.e. PC+4 value at R0.</td>
</tr>
<tr>
<td>JALR</td>
<td>10110</td>
<td>JALR Rd Rs</td>
<td>Jump and Link Register: Save the current value of PC i.e. PC+4 to register Rd and set PC to value in register Rs</td>
</tr>
<tr>
<td>MOVE</td>
<td>10111</td>
<td>MOVE Rd Rs</td>
<td>Copy value in Rs into Rd</td>
</tr>
<tr>
<td>Unused</td>
<td>11000-11110</td>
<td>Expandable instructions</td>
<td>Required for 641 students. Should include 8-bit multiply and 8-bit divide</td>
</tr>
<tr>
<td>NOP</td>
<td>11111</td>
<td>NOP</td>
<td>Do nothing.</td>
</tr>
</tbody>
</table>