Design Report

Han Zhang u7235649, Jinchao He u7230788

26/04/2022

Overview:

On the original CPU circuit, we expanded 8 new instructions: ILDR, an I-Mode memory instruction to directly load from a specified memory location; MAX-instruction, to assist our CPU to compare two numbers and load the larger one; and 6 ALU instructions: XOR, NOT, DEC (decrement), INC (increment), LSFT (left bit shift) and RSFT (right bit shift). We change the ALUOP signal to 4 bits wide because we have more ALU instructions. All the test assembly codes are in the folder ***asm***.

Implementation:

1) ILDR instruction:

The machine code is *0011 z<rd> <imm8>* and the assembly syntax is *ILDR? <RD:reg>, <IMM8:uint8>.* This is a I-Mode memory instruction. In the control unit most of the circuits for generating LDR signals are used for ILDR, and an IMM8 output is added to output an 8-bit immediate value. In the CPU I use a 2-way selector, IMM8 is selected as the 1A signal of the memory during the ILDR instruction, and the value obtained from the register is used at other times. The machine code starts with 0011 but not 0010 because this can simplify the circuits. The LDR starts with 0101, so for the LDR signal in the control unit, the first and the last bits of the two instructions are the same, and only the middle two bits need to be judged by a XOR gate.

2) MAX instruction:

The machine code of the instruction looks like: *0111 z<rd> 0<ra> 0<rb>*; the assembly syntax should look like: MAX? {<RD:reg?RA>, } <RA:reg>, <RB:reg>.

Max instruction will assist the CPU to compare the values of two registers (ra & rb) and store the value of the bigger one into another register(rd). It should be noticed that the instruction should be excluded after the sub-alu-instruction to write the right value into rd. An example of the assembly code of the instructions on the right side:

If the input 4-bit op-code is equal to 7, a HIGH signal will be outputted. It will be used as control signal to control the multiplexers to input the 4-bit aluop code of 1(sub) to alu-unit to do the subtraction between two registers and stor FLAG into FL register and 2-bit control-code of 0 will be send to the DMUX. Based on the FLAG signals produced before, the control unit will send option code to alu-unit to ask alu to write the value of ra or rb into register rd. If the zero and negative signal are both 0, ra will be stored in rd. If one of the two signals is 1, then rb will be stored in rd.

3) XOR instruction:

The machine code is *1100 z<rd> 0<ra> 0<rb>*, the assembly syntax is *XOR? {<RD:reg?RA>, } <RA:reg>, <RB:reg>* and the ALUOP is *0100*. This is an ALU instruction to bitwise XORs the values in RA and RB and stores the result in RD. This is similar to AND and ORR instructions. For the RESULT of the ALU, we previously calculated the result of each instruction and the logical conditions of their occurrence, but this will make the circuit very complicated when there are more ALU instructions. In the end we decided to use a multiplexer to decide the final output and use ALUOP as the selection signal. But we use two instead of one to determine the result, which will be explained later in the INC and DEC instructions.

4) LSFT instruction and RSFT instruction:

The machine code for LSFT is *1101 z<rd> 0<ra> m imm*, the assembly syntax is LSFT {<RD:reg?RA>, } <RA:reg>, <M:unit1>, <IMM:unit3> and the ALUOP is *0101*. The machine code for RSFT is *1110 z<rd> 0<ra> m imm*, the assembly syntax is RSFT {<RD:reg?RA>, } <RA:reg>,

<M:unit1>, <IMM:unit3> and the ALUOP is *0110*. The LSFT can left bit shift the value in RA for IMM bits where the RSFT can right bit shift for IMM bits. We add SFT and SFTM signals in the control unit and ALU to indicate the bits to shift and the shift mode. Use SFTM to set the mode, set it to 0 for logical mode and 1 for rotate mode. The value can be shifted up to 7 bits. We use a barrel bit shifter and make it a custom component. We use two 8-way multiplexers to select the number of bits to shift and another 2-way multiplexer to select the direction. Use an AND gate on the shifted out bits and SETM in logical mode to zero out the shifted out bits.

5) **NOT instruction:**

The machine code is *1111 z<rd> 0<ra> 0000,* the assembly syntax is *NOT? {<RD:reg?RA>, } <RA:reg>* and the ALUOP is *0111*. It bitwise inverts the value in RA and stores it in RD. It treats the value as an unsigned value.

6) **INC instruction and DEC instruction:**

The machine code for INC is *1111 z<rd> 0<ra> 0010,* the assembly syntax is *INC? {<RD:reg?RA>, } <RA:reg>* and the ALUOP is *1000.* The machine code for DEC is *1111 z<rd> 0<ra> 0011,* the assembly syntax is *DEC? {<RD:reg?RA>, } <RA:reg>* and the ALUOP is *1001.* These two instructions can increase and decrease the value by 1 respectively. For NOT instruction and these two instructions, we use the last 4 bits as additional as extra flags to indicate instructions, since their last 4 bits are all free. We make the most significant bit of ALUOP for these two instructions to be 1, so that the remaining 3 bits correspond to the ALUOP bits of addition and subtraction, and the value to be added or subtracted is selected by the multiplexer. We use an 8-way multiplexer for the other ALU instructions, use the last three bits of ALUOP as selection bits, and input this output with the increment and decrement values into a 2-way multiplexer, using the highest bit of ALUOP as the selection bit, thus avoiding the waste caused by directly using one 4-bit selection bits multiplexer.

Reflection:

1) The initial plan for max-instruction is to compare two registers by using a single instruction. However, we found that a sub-instruction is required before the max-instruction to achieve the task. Based on the knowledge support from one of my electric engineer friends who graduated from UNSW, we realized that it is due to the limitation of the QuAC ISA (it uses RISC architecture): a RISC architecture which has a simple circuit often requires multiple simple instructions to complete a specific task.
2) The order of the machine code of the instructions affects the complexity of the circuit. Like ILDR instruction starts with 0011 but not 0010, circuits can be simplified by adjusting the instructions represented by machine code.