Documentation

Core Modules

1. Control Unit: The control unit is for keeping all the modules functioning properly. The unit takes in the instruction opcode and decides which all control signals should be activated in accordance with the instructions. They modulate the control signals given below:
    - jump
    - beq
    - bne
    - mem_read
    - mem_write
    - alu_src
    - reg_dst
    - mem_to_reg
    - reg_write

    a) **mem_read** : This control signal enables the data to be read from the memory. This is needed only for taking the data from memory and storing it in the GPRs.
    b) **mem_write**: This helps in giving the memory write enable to write the data from register to memory.
    c) **alu_src**: For LOAD/STORE instructions, this is 1. This is because ,the **ext_im** signal line for the offset is selected for ALU operation(addition and subtraction). Whereas, a 0 gives read_data2 the value fetched from the second register(**reg_read_data_2**).
    d) **reg_dst**: It is used to determine whether the register address which is to be taken from the **instr**, is to be used. The register can only be written once every cycle. Thus, **reg_dst** decides which register to write to. In data processing instructions, where e3 registers are used, the **reg_dst** takes 1, for taking the address from 5->3 bit(3 bits), where for all other cases, it takes value 0 for bit 8->6(3 bits) and the offset bits(5->0) in the instructions are considered valuable.
    e) **mem_to_reg**: 1 indicates memory to register data transfer(LOAD/STORE) and 0 for ALU operation (result storage).
    f) **reg_write**: For indicating the the register data write enable. 1(set) for writing to register.
    g) **jump**: For indicating the jump instruction. This is needed to select between the instruction for jump or to continue the normal flow. The PC is altered with this signal.
    h) **beq**: Set to 1 for beq instructions.
    i) **bne :** Set for bne instructions.

2. ALU Control Unit: The ALU control unit decides which ALU operation have to be done for each instruction. It generates **ALU_Cnt** from **alu_op** and **opcode** from the instruction. The **ALU_Cnt** is fed into the alu unit for getting the required ALU operation.
3. ALU: The ALU unit looks at the **alu_control** and does the corresponding operation. The result is then pushed out of the unit.

4. Data Memory: The RISC CPU is based on the Harvard Architecture. The module have a memory array, as configured by the parameter file, and takes **mem_access_addr** line for accessing the memory element at that location. The data may be written to or taken from the location through the respective control lines. The memory array can we varied accordingly to our necessity.

5. Instruction Memory: This module contains the memory array for instructions. The PC register address is fed, from which the instruction address is taken and the instruction is taken as output.

6. GPR: General Purpose Registers are 8 in number. The are each 16 bits and can be accessed two at a time and can we written to one in every clock cycle. **reg_write_en** is the control signal for enabling writing to register.

Instruction Format

Memory Access: Load

Op	Rs1	Ws	offset
4	3	3	6

Memory Access: Store

Op	Rs1	Rs2	offset
4	3	3	6

Data Processing

Op	Rs1	Rs2	Ws	Useless
4	3	3	3	3

Branch

Op	Rs1	Rs2	offset
4	3	3	6

Jump

Op	offset
4	12

OP	
0000	Load Word
0001	Store Word
0010	Add
0011	Subtract
0100	Invert(1’s complement)
0101	Logical ShiftLeft
0110	Logical Shift Right
0111	Bitwise AND
1000	Bitwise OR
1001	Set on Less Than
1010	Hamming Distance
1011	Branch on equal
1100	Branch on Not equal
1101	Jump

Processor Control Unit Design

Control signals									
Instruction	Reg Dst	ALUSrc	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
Data-processing	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	10	0
SW	0	1	0	0	0	1	0	10	0
BEQ,BNE	0	0	0	0	0	0	1	01	0
J	0	0	0	0	0	0	0	00	1

ALU Control Unit

ALU Control				
ALU_Op	Opcode(hex)	ALU_cnt	ALU Operation	Instruction
10	xxxx	000	ADD	LW,SW
01	xxxx	001	SUB	BEQ,BNE
00	0002	000	ADD	D-type: ADD
00	0003	001	SUB	D-type: SUB
00	0004	010	INVERT	D-type: INVERT
00	0005	011	LSL	D-type: LSL
00	0006	100	LSR	D-type: LSR
00	0007	101	AND	D-type: AND
00	0008	110	OR	D-type: OR
00	0009	111	SLT	D-type: SLT

Instructions

1. LOAD instruction:
   Eg: 0000 010 000 000000
   Op R0 R1 offset
   (4) (3) (3) (6)
   - The load instruction takes the offset and load the value from the computed location to the respected register. Here, the R0 bits are taken as address and the data at R0 is added with the modified offset, to take the value from the obtained address from memory and place it in R1.
   - The reg_dst signal should be 0 to get the R0 [8:6].

- The order of transfer of information between register is from left to right, i.e, the information is taken from the register of the MSB to register of LSB.
- The address from 11->9 is given into the **reg_read_addr_1** of GPRs(always).
  reg_read_addr1 -> always
  reg_read_addr_2 -> reads from addr 8-> 6 by default.
- **reg_read_addr_1** gets the address, and fetches the data, which is passed onto the alu unit, along with **read_data2** ,which gives out the modified offset according to the control signal **alu_src** coming from the control unit.
- This is the same with the case of STORE instructions which need the offset to be subtracted from the reg data, to find the location to which the data is to be moved from the register.
- From the ALU, the result is given to **ALU_out**, which is then fed to the Data Memory unit, to retrieve the data and then given to **reg_wite_data**. This is then fed to the GPRs **reg_write_data**, and with the **reg_write_dest** taken from bits 8->6 of instruction, selected by using the **reg_dst**.
- The **ALU_out** goes into the **register_write_data**.

2. STORE instruction : Storing from register to memory.
   Eg. 0001 001 010 000000
   Op R0 R1 offset
   - This is the same as that of the LOAD, but the ALU performs subtraction.
   - The data in **reg_read_data_1** and the offset is used to calculate the location of in the Data Memory, for storing the data from the **reg_read_data_2.**
3. Branch on Equal: (BEQ)
   Eg. 1011 000 001 000001
   Op R0 R1 offset
   - BEQ takes the instruction from PC, adds 2 and then takes offset with left shift of 1, i.e, 7 bit offset.
   - We need to take the 11->9 bits and 8->6 bits as address and get **reg_read_addr_1 & reg_read_addr2.**
   - Now, we have the alu_src as 0,for making it to take the data from above registers.
   - The alu unit takes this and subtracts one from another. The ALU_out is then put through the logic which checks if the result is zero by referring to a zero flag.
   - S, in the current instruction, we take an offset of 000001 from the instruction[5:0].
   - To convert this into 16 bit address to be added with the program counter, we take,
     Offset = 00001
     ext_im = 16'd1
     for, **pc_beq = pc2 + {ext_im[14:0],1'b0}**
     , where **pc2 = pc_current + 2**
     Therefor, **pc_beq = pc2 + {16'h0002}** // This adds the offset in the bracket with pc2
   - **pc_beq** then becomes the pc_next, only when this instruction occurs.
   - **R0 &R1** subtract from each other and if the zero flag is reset, we can say that R0 and R1 are equal.
   - **beq_control** control signal is set to 1, which makes the **pc_2bne** get **pc_beq** , the address after processing with the offset.

- Now, the **pc_2bne** is combared with jump, and as **jump** is 0, the value is passed on to **pc_next** .

4. Branch on Not Equal (BNE): This is having the same procedure as the BEQ, but with the reverse logic.
5. Jump : Jump instructions take only the offset.
   Eg. 1101 000000000000
   Op offset(12 bits)
   - This offset is taken into **jump_shift,** added 0 at LSB, making it 13 bits.
   - Now, this added with the **pc2[15:13] (3 bits)** as MSG , to get the new 16 bit address which is given in the **pc_next**. The processor execution line then jumps to this address in the instruction memory.

Control Signals

1. **reg_dst** : It is used to determine whether the register address which is to be taken from the **instr**, is to be used. The register can only be written once every cycle. Thus, **reg_dst** decides which register to write to. In data processing instructions, where e3 registers are used, the **reg_dst** takes 1, for taking the address from 5->3 bit(3 bits), where for all other cases, it takes value 0 for bit 8->6(3 bits) and the offset bits(5->0) in the instructions are considered valuable.
2. **alu_src**: In LOAD/STORE instr, this is 1. This is because on 1, the ext_im is made from the offset [5:0] (6 bits) into [15:0] (16 bits), which is then fed into the **read_data2.** If 0, this takes the usual data from the second register, **reg_read_data_2,** and feeds it into the alu unit for respective operations.
3. **alu_op**: is fed directly into the control unit. This 2 bit [1:0] lines determine addition or subtraction of the LOAD/STORE instructions.
   alu_op : 10 (LOAD)
   : 01(STORE)
   : 00(For all other instructions)
4. **beq** : is set to 1 for BEQ instructions.
5. **bne :** is set to 1 for BNE instructions.
6. **jump:** is set to 1 for Jump instructions.
7. **beq_control:** This is to execute the BEQ logic. **beq** control signal is AND-ed with the zero flag to check for branching requirements. When 1, **pc_2beq** gets **pc_beq**. If 0 , then it gets pc2(the next instruction address).
8. **bne_control :** Same as **beq_control,** except ,this logi should execute when zero flag is not raised. When 1, **pc_2bne** gets pc_bne. If 0, it gets pc_2beq(as **pc_2beq** is opposite to **bne_control** and hence they have to co-exist).
9. **Mem_to_reg:** 1 indicates memory to register data transfer(LOAD/STORE) and 0 for ALU operation (result storage).
10. **reg_write :** To GPRs, for register write.
11. **mem_read:** Memory read enable.
12. **mem_write :** Memory write enable.

Future improvements: More instructions can be added by modifying the design pipeline.