b10902052 lab2 report

Modules Explaination

Adder.v
Add two 32-bit input, and output the sum.

ALU_Control.v
`ALU_Control` module is an ALU controller. It takes control signals `ALUop_i`, `func3`, and `func7`, and outputs a three-bit signal `ALUfunc_o` that specifies the ALU operation.

ALU.v
Here I simply copy it from `lab1`, you can find the explain from `lab1`.

AndUnit.v
AND two 32-bit, out is a 32-bit value after and.

Control.v
I copy it from `lab1` too, and add other control bit
`Branch_o` : Control the branch.
`MemtoReg` : Control the RD data from memory or aluresult.
`MemRead` : Control the memory read 1 if it allow read.
`MemWrite` : Control the memory write 1 if it allow write.

EqualUnit.v
Test equality of two 32-bit, out is a 1-bit and it is `1` if equal.

IDEX.v EXMEM.v MEMWB.v
Registers, read left-side as input, store on posedge of `clk_i`, and output to right-side.

FwdUnit.v
Impelement as the pseudo-code from spec.

HarzardUnit.v
Detect if it need an `no-op` insertion.

ImmGen.v
`ImmGen` takes a 32-bit instruction `Inst_i` and generates a 32-bit immediate value `ImmGen_o`. The immediate value is extracted based on the opcode of the instruction, with specific bit positions considered for different opcodes. If the opcode doesn't match predefined cases, the output is set to zero.

LeftShifter.v
Read an 32-bit input and output another 32-bit with shift 1-bit left.

MUX32.v , MUX32_4.v
With two or four 32-bit input, a select bit, and a 32-bit output. Here I use the `?` operator to complete this.

CPU.v
`CPU` module represents a processor. It consists of various sub-modules, each responsible for specific tasks within the processor's pipeline schedule. The main functionalities include instruction fetch `IF` , instruction decode `ID` , execute `EX` , memory access `MEM` , and write-back `WB` . The `CPU` module integrates these components and manages data flow between them.

Here's a brief overview of the main components:

- IF (Instruction Fetch):
    - Fetches instructions from memory based on the program counter (PC).
    - Includes a PC incrementer `IF_adder` and a multiplexer `IF_mux` .
- ID (Instruction Decode):
    - Decodes instructions, extracts immediate values, and manages hazard detection.
    - Controls the pipeline flush `ID_FlushIF` and stall `ID_stallout` .
    - Utilizes sub-modules like `Control` , `Registers` , `ImmGen` , etc.
- EX (Execute):
    - Performs ALU operations based on the ALU control unit `EX_ALUcon` .
    - Handles forwarding logic `EX_fwd` and multiplexing.
    - Manages forwarding control signals `EX_fwdA` and `EX_fwdB` .
- MEM (Memory Access):
    - Accesses data memory `Data_Memory` for load and store operations.
    - Controls memory operations and manages data flow.
- WB (Write-Back):
    - Writes results back to the register file.

The overall `CPU` module orchestrates the interactions between these components based on the clock signal `clk_i` and reset signal `rst_i` .

Difficulties Encountered and Solutions in This Lab

- Difficulties

1. I found that there seemed to be an issue with the register assignment. Specifically,

```
assign  RS1data_o = (RS1addr_i == RDaddr_i && RegWrite_i)?
        RDdata_i : register[RS1addr_i];
assign  RS2data_o = (RS2addr_i == RDaddr_i && RegWrite_i)?
        RDdata_i : register[RS2addr_i];
```

If `(RS1addr_i == RDaddr_i && RegWrite_i)` and `RDaddr_i == 5'b0`, it would directly assign `RDdata_i` to `RS1addr_i`, leading to an error.

- Solutions

1. Later, I resolved this issue by encapsulating other model's conditional check within a process block and using `always @(*)`.

Development Environment

- OS :
  - Distributor ID: Ubuntu
  - Description: Ubuntu 22.04.2 LTS
  - Release: 22.04
  - Codename: jammy
- compiler :
  - iverilog :
    - Icarus Verilog version 11.0 (stable)
- IDE :
  - vscode
    - 1.83.1
    - f1b07bd25dfad64b0167beb15359ae573aecd2cc
    - x64