CA Lab 1 Report

B08902055 李英華

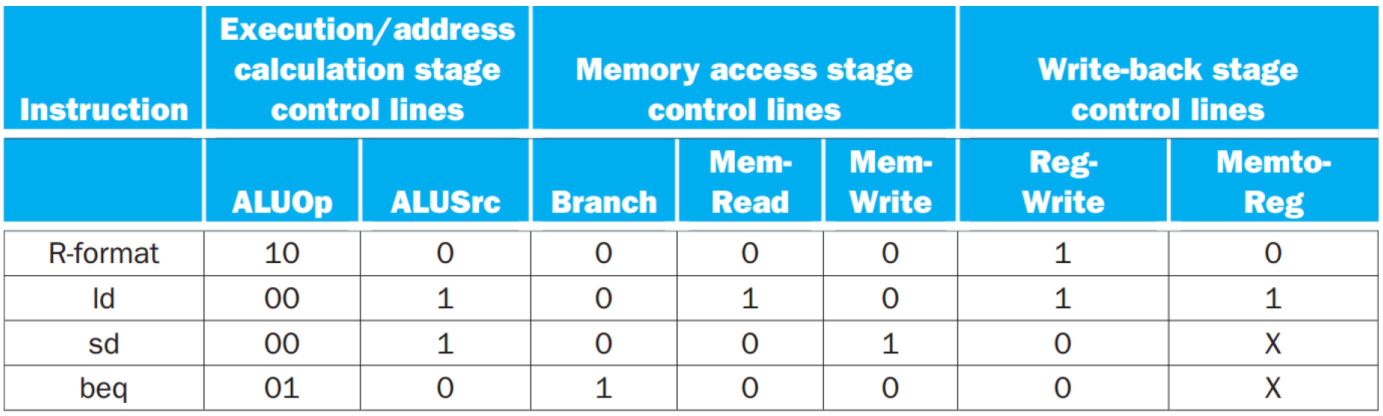
Modules Explanation

Control

```
// Input
Op_i[6:0], NoOp_i

//Output
RegWrite_o, MemtoReg_o, MemRead_o, MemWrite_o, ALUSrc_o, Branch_o, ALUOp_o[1:0]
```

If `NoOp_i` is 1, simply set all output value to 0. If not, use `Op_i[6:0]` to determine the outputs according to the following table.



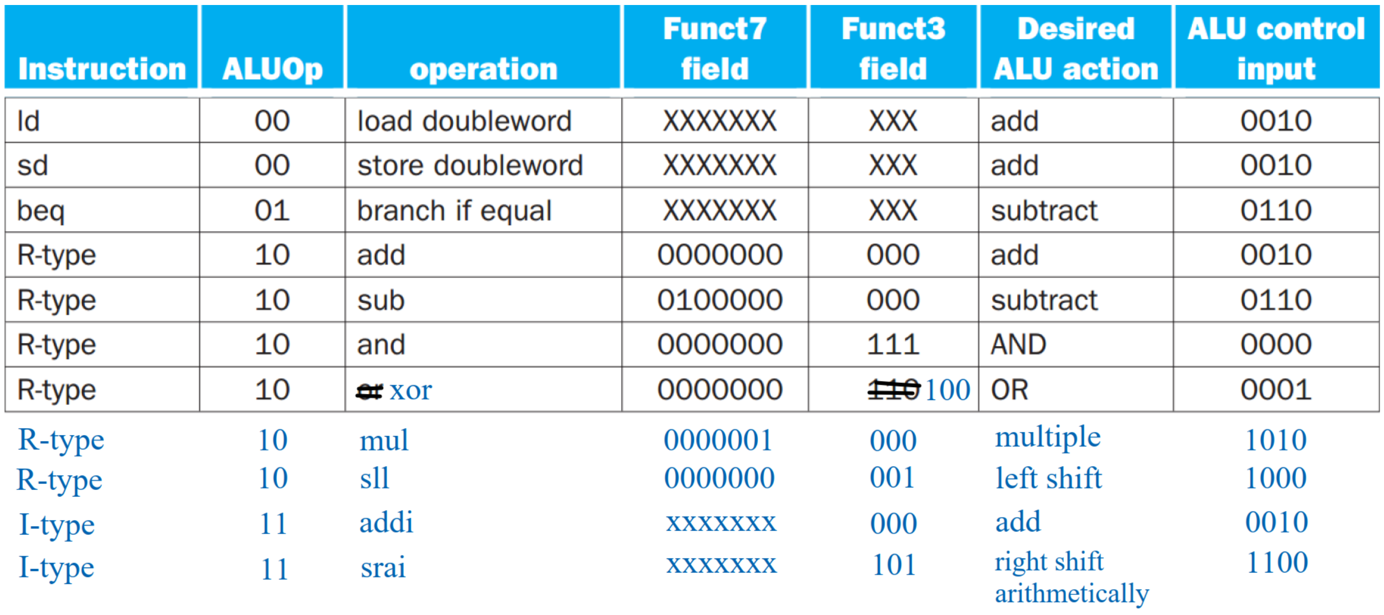
Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
ld	00	1	0	1	0	1	1
sd	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X
I-type	11	1	0	0	0	1	0

ALU_Control

```
// Input
funct_i[9:0], ALUOp_i[1:0]

// Output
ALUCtrl_o[3:0]
```

With `funct_i` (`{funct7, funct3}`) and `ALUOp_i` , decide `ALUCtrl_o` according to below table.



Instruction	ALUOp	operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	~~or~~ xor	0000000	~~110~~ 100	OR	0001
R-type	10	mul	0000001	000	multiple	1010
R-type	10	sll	0000000	001	left shift	1000
I-type	11	addi	xxxxxxx	000	add	0010
I-type	11	srai	xxxxxxx	101	right shift arithmetically	1100

ALU

```
// Input
data1_i[31:0], data2_i[31:0], ALUCtrl_i[3:0]

// Output
data_o[31:0]
```

According to different `ALUCtrl_i`, perform different arithmetic on `data1_i`, `data2_i`, and output `data_o`.

Forward_Control

We detect EX or MEM hazard here by logic provided in spec (use the notation in spec here for explanation)

- In both case, `RegisterRd` should not be 0 to prevent write to `x0`, and only check for hazard if `RegWrite` is set.
- If the `MEM/WB.RegisterRd` or `EX/MEM.RegisterRd` is the same as one of the `ID/EX.RegisterRs`, set `ForwardA` and `ForwardB` accordingly. Otherwise, all set to 0.

Hazard_Detection

```
// Input
RS1addr_i[4:0], RS2addr_i[4:0], MemRead_i, RDaddr_i[4:0]

//Output
stall_o, PCWrite_o, NoOp_o
```

Check for load-use hazard:

- When `MemRead_i` is set
- When `RDaddr_i` (ID/EX) equals `RS1addr_i` or `RS2addr_i` (IF/ID)
    - set `stall_o` as 1 for IF_ID
    - set `PCWrite_o` as 0 for PC
    - set `NoOp_o` as 1 for Control

Sign_Extend

As the input contains the whole instruction, we can check `Opcode` and `funct` to decide which sets of bits we want to use as `imm`, and extend its sign.

imm[11:0]		rs1	000	rd	0010011	addi
0100000	imm[4:0]	rs1	101	rd	0010011	srai
imm[11:0]		**rs1**	**010**	**rd**	**0000011**	**lw**
imm[11:5]	**rs2**	**rs1**	**010**	**imm[4:0]**	**0100011**	**sw**
imm[12,10:5]	**rs2**	**rs1**	**000**	**imm[4:1,11]**	**1100011**	**beq**

Pipeline Latches

- For all latches, their outputs are also registers.
- Initialize all the pipeline registers to 0 if `negedge start_i`.
- If `posedge clk_i`, pass all inputs to output registers. (Except for IF_ID, see below.)

IF_ID

```
// Input
clk_i, start_i, flush_i, stall_i, PC_i[31:0], instr_i[31:0]

// Output & registers
PC_o[31:0], instr_o[31:0]
```

- When `posedge clk_i` and if input `stall_i` or `flush_i` not set, pass `PC_i`, `instr_i` to `PC_o`, `instr_o`
- Else, set them to 0

CPU

```
// Input
clk_i,  rst_i, start_i
```

Connect the above all components together as the final path provided in spec Fig 5. `clk_i` indicate the clock cycle of the pipelined registers. `start_i` is used for initializing pipeline registers and PC.

$\mathrm{testbanch}$

Initializing `CPU.PCWrite_reg` , `CPU.Stall_reg` , `CPU.NoOp_reg` , `CPU.PC_select_reg` (for IF/ID.flush) here. Also, initializing pipeline registers here (as required).

Difficulties Encountered and Solutions in This Lab

- Hard to debug:
  - `iverilog` would not report error if pass the wrong thing (typo for the wire name or something else) into module.
  - [Solution]: One can find this bug out using the gtkWave to see what is exactly feed into each module. It'll regard the non-exist name as wire with value 'Z'.
- Initialization:
  - At first I initialize pipeline registers and some of the module inputs (e.g., `stall` , `flush` , `NoOp`) in the initial block of $\mathrm{testbanch}$, but this doesn't work and results in many xxx value at the beginning.
  - [Solution]: Pass `start_i` into each pipeline latche and do initialization when `negedge start_i` .

Development Environment

- Operating System: Windows 10
- Compiler: iverilog