HW5 solution

October 16, 2019

Problem 1: Hazards

a) To implement `add rd, rs1, rs2, rs3: rd = rs1 + rs2 + rs3`, we must add a new input address port and a new output data port to the register file. The third address would be coded in the instruction at unknown address. A new input port also needs to be added to the ALU. This is shown in figure 1.

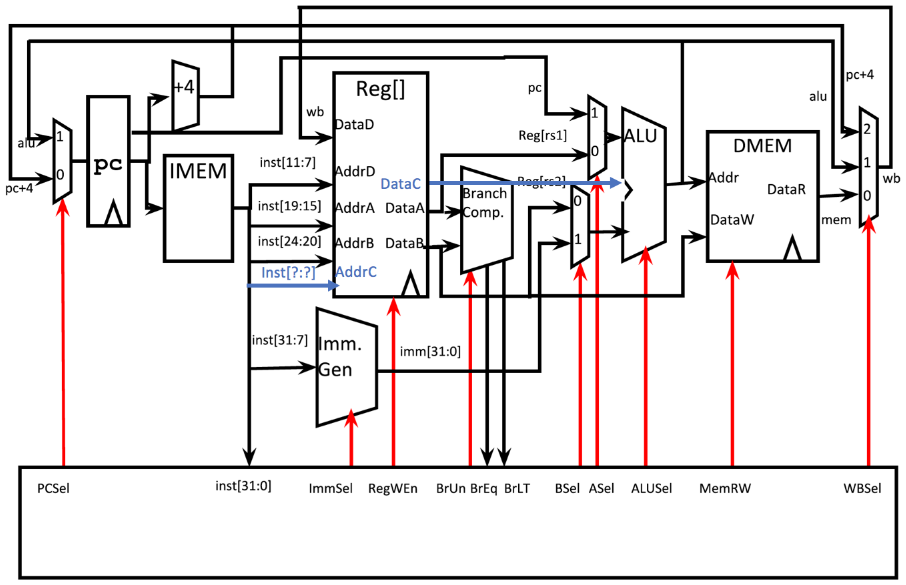

Figure 1: Updated CPU diagram for 1.(a)(i)

To implement `swadd rs1, rs2, imm: M[rs1] = rs2 + imm`, we have to add three extra muxes. The first one is in front of the first input of the ALU to make rs2 value a possible input to the ALU. Then we add a mux in front of the addr to DMEM to select between rs1 and the output of the ALU, and another mux in front of DataW of DMEM to allow ALU output to be written into the memory. The select signals of the two muxes in front of DMEM are not shown.

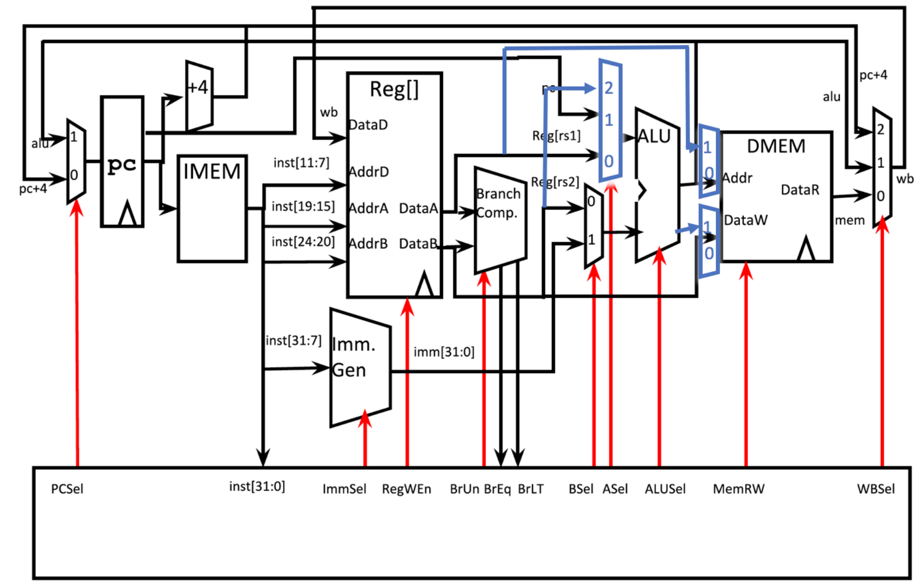

Figure 2: Updated CPU diagram for 1.(a)(ii)

b) (a) When adder is broken, all addition operations cannot be performed. beq wouldn't work because pc = pc + offset needs to be computed sw wouldn't work because reg[rs1] + offset needs to be computed jal wouldn't work because pc = pc + offset needs to be computed

(b) "lw" and "add" wouldn't work because they needs to store memory data to destination register, which requires RegWEn to be 1.

(c) When Asel is stuck at 0, pc cannot be added with immediate values so the next instruction is stuck at pc+4. Therefore, any instruction that requires computation of new pc would not work. Here that would be "beq" and "auipc".

Problem 2: Pipelines

(a) Because we can make the assumption that while data is being written back into the register file, it can be read directly (it has been bypassed through internal hardware in the regfile for example), there should be no need for stalling, the timing diagram is simply:

Cycle	IF	EX	WB
1	add	-	-
2	sub	add	-
3	add	sub	add
4	or	add	sub
5	and	or	add
6	xor	and	or
7	add	xor	and
8	-	add	xor
9	-	-	add

It takes 9 cycles to implement.

(b) The following diagram shows the forwarding necessary for a 5-stage pipeline. Notice that forwarding should be done in both memory read and write back stage. Because in a case like this :

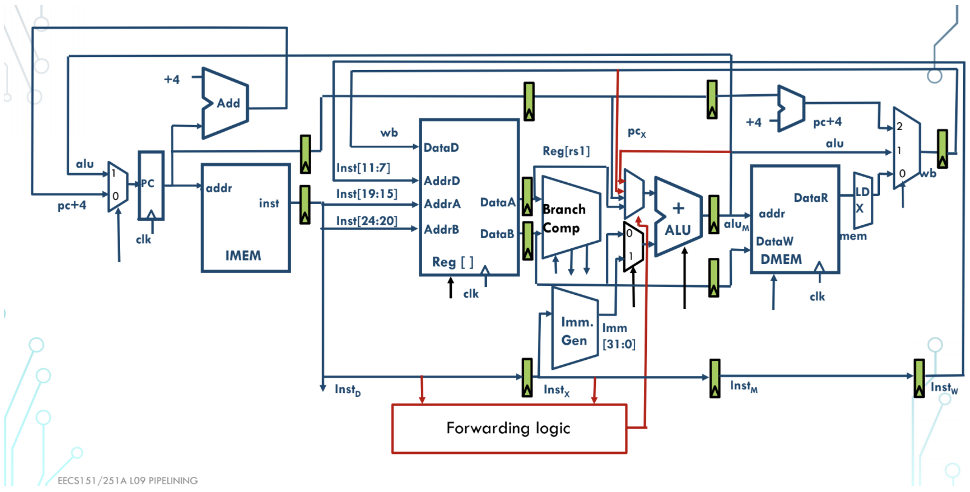

Figure 3: Forwarding in 5-stage pipeline

```
add x1, x2, x3
nop
add x4, x1, x2
```

the forwarding should be able to handle x1 properly.

(c) The new timing table now becomes:

Cycle	IF	ID	EX	MR	WB
1	add	-	-	-	-
2	sub	add	-	-	-
3	add	sub	add	-	-
4	or	add	sub	add	-
5	and	or	add	sub	add
6	xor	and	or	add	sub
7	add	xor	and	or	add
8	-	add	xor	and	or
9	-	-	add	xor	and
10	-	-	-	add	xor
11	-	-	-	-	add

Now it takes 11 cycles. However, this doesn't mean this will take longer than the 3-stage pipeline. Actually with more pipeline stages, the critical path of each stage is shorter so the CPU can run at a higher clock frequency, so optimization should be done on number of pipeline stages vs critical path delay.

Problem 3: Branch Prediction

(a) When pc+4 is always taken:

Cycle	IF	ID	EX	MR	WB
1	li (nop)	-	-	-	-
2	li	li (nop)	-	-	-
3	add	li	li (nop)	-	-
4	li	add	li	li (nop)	-
5	bne	li	add	li	li (nop)
6	addi (0x14)	bne	li	add	li
7	subi	addi (0x14)	bne	li	add
8	addi (0x1c)	subi	addi (0x14)	bne	li
9	addi (0x1c)	-	-	-	bne
10	nop	addi (0x1c)	-	-	-
11	-	nop	addi(0x1c)	-	-
12	-	-	nop	addi(0x1c)	-
13	-	-	-	nop	addi(0x1c)
14	-	-	-	-	nop

Notice here that it takes 3 extra cycles to take the wrong prediction because the new pc value is fed back to the mux before PC register after one flip flop, which means it gets delayed by one more cycle.

(b) In the case where branch always taken is predicted, there will be extra hardware that directly calculates the value of the new PC, so it should exist immediately in the next cycle and doesn't need to wait for ALU to compute it. The new tablethen becomes:

Cycle	IF	ID	EX	MR	WB
1	li (nop)	-	-	-	-
2	li	li (nop)	-	-	-
3	add	li	li (nop)	-	-
4	li	add	li	li (nop)	-
5	bne	li	add	li	li (nop)
6	addi (0x1c)	bne	li	add	li
7	nop	addi (0x1c)	bne	li	add
8	-	nop	addi (0x1c)	bne	li
9	-	-	nop	addi (0x1c)	bne
10	-	-	-	nop	addi (0x1c)
11	-	-	-	-	nop