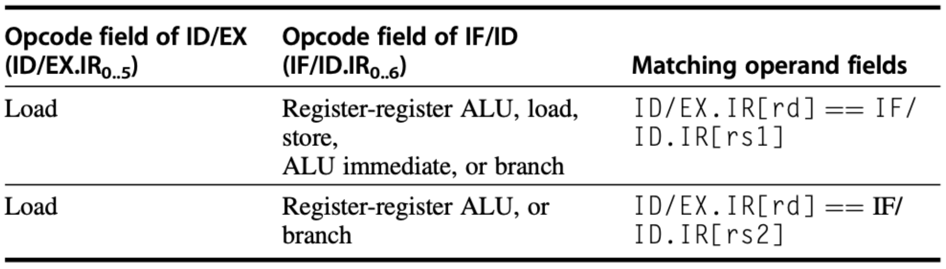
ECE 786 Advanced Computer Architecture: Data Parallel Processors
Spring 2023 Huiyang Zhou HW 1

1. C.8

[15] <C.5> Construct a table like that shown in Figure C.22 to check for WAW stalls in the RISC V FP pipeline of Figure C.30. Do not consider FP divides.



Opcode field of ID/EX ($ID/EX.IR_{0..5}$)	Opcode field of IF/ID ($IF/ID.IR_{0..6}$)	Matching operand fields
Load	Register-register ALU, load, store, ALU immediate, or branch	`ID/EX.IR[rd] == IF/ID.IR[rs1]`
Load	Register-register ALU, or branch	`ID/EX.IR[rd] == IF/ID.IR[rs2]`

Figure C.22 The logic to detect the need for load interlocks during the ID stage of an instruction requires two comparisons, one for each possible source. Remember that the IF/ID register holds the state of the instruction in ID, which potentially uses the load result, while ID/EX holds the state of the instruction in EX, which is the load instruction.

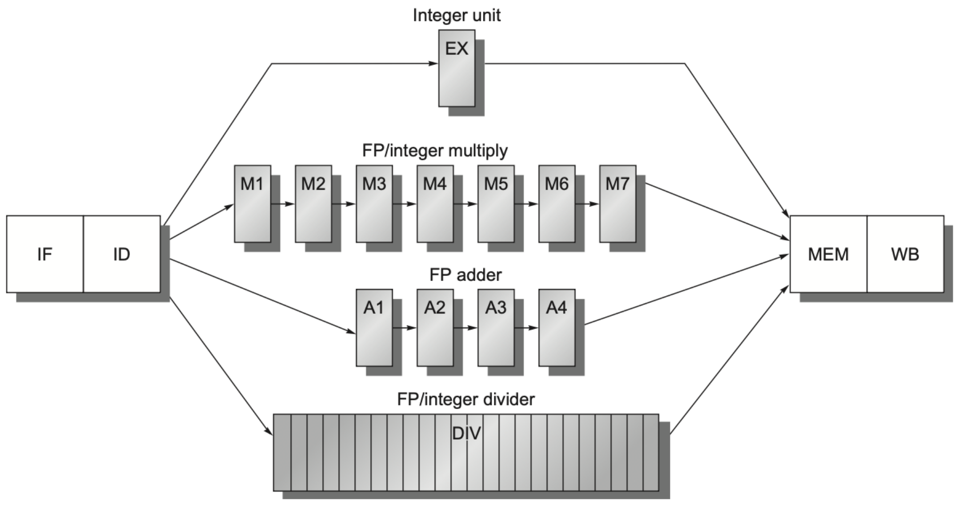


Figure C.30 A pipeline that supports multiple outstanding FP operations. The FP multiplier and adder are fully pipelined and have a depth of seven and four stages, respectively. The FP divider is not pipelined, but requires 24 clock cycles to complete. The latency in instructions between the issue of an FP operation and the use of the result of that operation without incurring a RAW stall is determined by the number of cycles spent in the execution stages. For example, the fourth instruction after an FP add can use the result of the FP add. For integer ALU operations, the depth of the execution pipeline is always one and the next instruction can use the results.

If integer, nothing really has to be done.
NOTES:
WAW hazard is also know as Output-dependence, where an example is as follows:

Table 1: New CPI for Benchmarks

Opcode field of ID/EX	Opcode field of IF/ID	Matching operand fields
FP OP	multiply, multiply	Nothing to be done
FP OP	multiply, multiply	Nothing to be done
FP OP	multiply, multiply	Nothing to be done
FP OP	multiply, multiply	Nothing to be done

```
add r3, r1, r2
add r3, r6, r7
```

2. C.9.a

[20/22/22] <C.4, C.6> In this exercise, we will look at how a common vector loop runs on statically and dynamically scheduled versions of the RISC V pipeline. The loop is the so-called DAXPY loop (discussed extensively in Appendix G) and the central operation in Gaussian elimination. The loop implements the vector operation $Y = a * X + Y$ for a vector of length 100. Here is the MIPS code for the loop:

```
foo:    fld    f2, 0(x1)    ; load X(i)
        fmul.d f4, f2, f0   ; multiply a*X(i)
        fld    f6, 0(x2)    ; load Y(i)
        fadd.d f6, f4, f6   ; add a*X(i) + Y(i)
        fsd    0(x2), f6    ; store Y(i)
        addi   x1, x1, 8    ; increment X index
        addi   x2, x2, 8    ; increment Y index
        sltiu  x3, x1, done ; test if done
        bnez   x3, foo      ; loop if not done
```

For parts (a) to (c), assume that integer operations issue and complete in 1 clock cycle (including loads) and that their results are fully bypassed. You will use the FP latencies (only) shown in Figure C.29, but assume that the FP unit is fully pipelined. For scoreboards below, assume that an instruction waiting for a result from another function unit can pass through read operands at the same time the result is written. Also assume that an instruction in WB completing will allow a currently active instruction that is waiting on the same functional unit to issue in the same clock cycle in which the first instruction completes WB.

a. [20]<C.5> For this problem, use the RISCV pipeline of Section C.5 with the pipeline latencies from Figure C.29, but a fully pipelined FP unit, so the initiation interval is 1. Draw a timing diagram, similar to Figure C.32, showing the timing of each instruction's execution. How many clock cycles does each loop iteration take, counting from when the first instruction enters the WB stage to when the last instruction enters the WB stage?

Instruction	1	2	3	4	5	6	7	8	9	10	11
1. fld f2, 0(x1)	IF	ID	EX	MEM	WB						
2. fmul.d f4, f2, f0		IF	ID	FMUL1	FMUL2	FMUL2	FMUL3	FMUL4	FMUL5	FMUL6	FMU
3. fld f6, 0(x2)			IF	ID	EX						
4. fadd.d f6, f4, f6				IF	ID	FADD1	FADD2	FADD3	FADD4		
5. fsd 0(x2), f6					IF	ID				EX	
6. addi x1, x1, 8						IF				ID	EX
7. addi x2, x2, 8										IF	ID
8. sltiu x3, x1, done											IF
9. bnez x3, foo											