Computer Architecture
DAT105
Exercise Session 2

M. WAQAR AZHAR
waqarm@chalmers.se

Chalmers University of Technology, Sweden

September 11, 2019

Agenda

- Problem 3.8
- Problem 3.10
- Problem 3.11

Agenda

- Problem 3.8
- Problem 3.10
- Problem 3.11

Problem 3.8

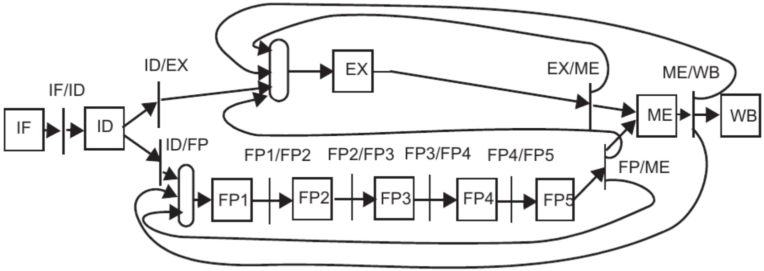


Figure 3.8. Pipeline with out-of-order execution completion

In this problem we evaluate the hardware needed to detect hazards in various static pipelines with out-of-order instruction execution completion. We will consider the floating point extension to 5-stage pipeline, displayed in figure 3.8. Each pipeline register carries its destination register number, either floating-point or integer. ME/WB carries two instructions, one from the integer pipeline and one from the floating-point arithmetic pipeline.

Problem 3.8

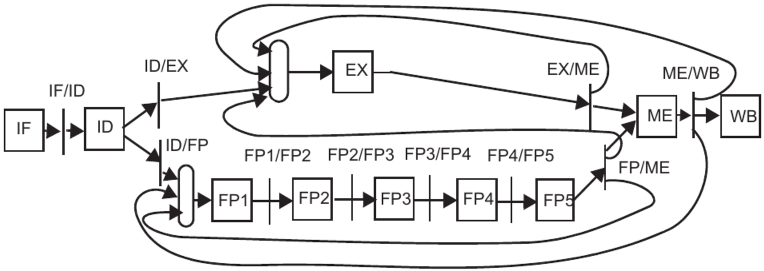


Figure 3.8. Pipeline with out-of-order execution completion

Consider the following type of of instructions consecutively

- Integer arithmetic/logic/Store instructions (inputs: two integer registers) and all Load instructions (input: one integer register)
- Floating-point arithmetic instructions (inputs: two floating-point registers)
- Floating-point Stores (inputs: one integer and one floating-point registers).

Problem 3.8 (part - a)

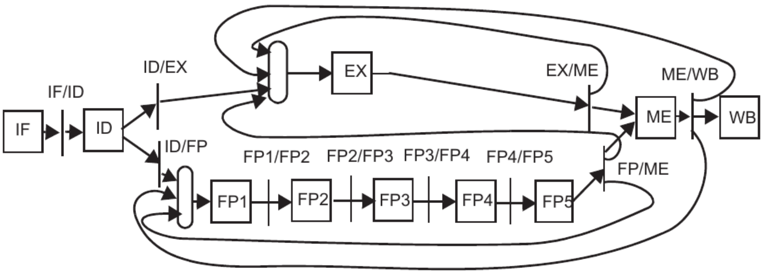


Figure 3.8. Pipeline with out-of-order execution completion

To solve RAW (Read after write) data hazards on registers (integer and/or floating point), hardware checks (interlocks) between the current instructions in ID and instructions in pipeline may stall instruction in ID. List first all pipeline registers that must be checked in ID. Since ME/WB may have two destination registers, list them as ME/WB (int) or ME/WB (fp). Do not list pipeline stages, list pipeline registers, and make sure that the set of checks in minimum.

Problem 3.8 (Part - a) Solution

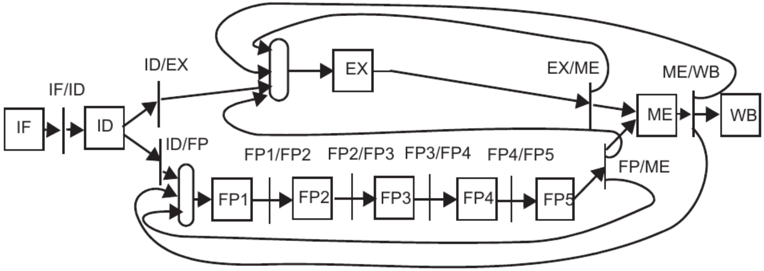


Figure 3.8. Pipeline with out-of-order execution completion

- If an integer arithmetic/logic/store instruction or load is in ID, check ID/EX for a load
- If an FP arithmetic instruction is in ID, check ID/EX (for FP loads), ID/FP, FP1/FP2, FP2/FP3, and FP3/FP4.
- If an FP store is in ID, check ID/EX (for loads returning address register or FP value), FP1/FP2, FP2/FP3, and FP3/FP4.

Problem 3.8 (part - b)

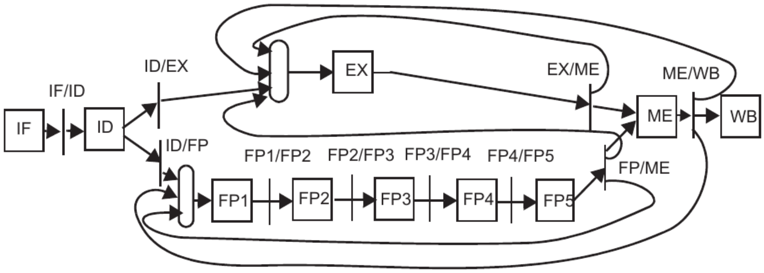


Figure 3.8. Pipeline with out-of-order execution completion

To solve WAW hazards on registers, we check the destination register in ID with the destination register of instructions in various pipeline stages. Please list the pipeline registers that must be checked. Make sure that the set of checks is minimum. IMPORTANT: remember that there is a mechanism in ID to avoid structural hazards on the write register ports of both register files.

Problem 3.8 (Part - b) Solution

.

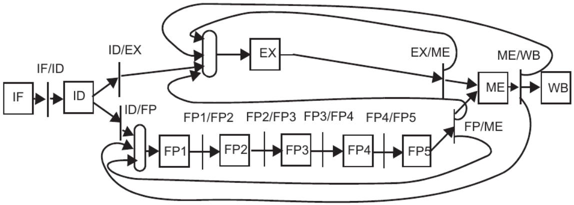


Figure 3.8. Pipeline with out-of-order execution completion

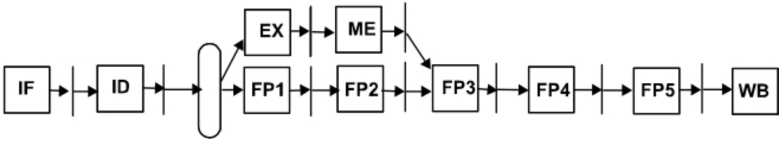
- If an integer arithmetic/logic/store or integer load is in ID: No check is necessary, since these instructions update integer registers only and follow the integer pipeline path in process order.
- If an FP load is in ID, check ID/FP, and FP1/FP2 and FP2/FP3 (no need to check FP3/FP4 because an FP load and an FP arithmetic instruction cannot reach the FP register file in the same cycle. This is done in ID to prevent structural hazards on the write port of the FP register file.)
- If an FP arithmetic instruction is in ID, there is no need to check any pipeline stage because FP arithmetic instructions cannot bypass a previous instruction.
- If an FP store is in ID, there is no need to check any pipeline stage because stores do not write value in registers.

Agenda

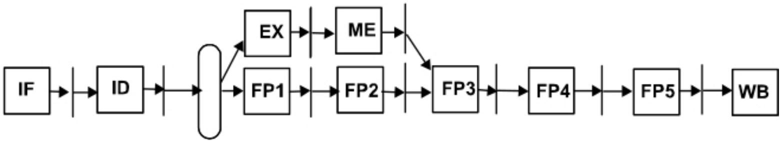
- Problem 3.8
- Problem 3.10
- Problem 3.11

Problem 3.10 (Part-a)

In the pipeline of Figure 3.9 WAW data hazards on registers are eliminated and exceptions can be handled in the WB stage where instructions complete in process order as in the classic 5-stage pipeline. As always values are forwarded to the input of the execution units.
a) List all required forwarding paths from pipeline registers to either EX or FP1 to fully forward values for all instructions. List them as
source $--$ > *destination* (e.g, $FP2/FP1 -- > FP1$)



Problem 3.10 (Part-a) Solution



We consider values forwarded to EX first.

$$EX/ME --> EX$$

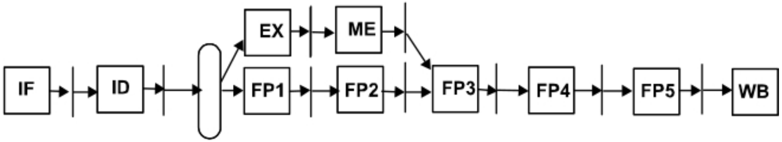
$$ME/FP3 --> EX$$

$$FP3/FP4 --> EX$$

$$FP4/FP5 --> EX$$

$$FP5/WB --> EX$$

Problem 3.10 (Part-a) Solution

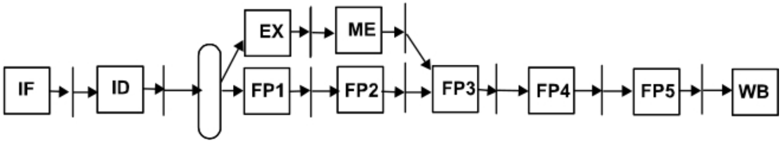


Consider values forwarded to FP1.

$$ME/FP3 --> FP1$$

$$FP5/WB --> FP1$$

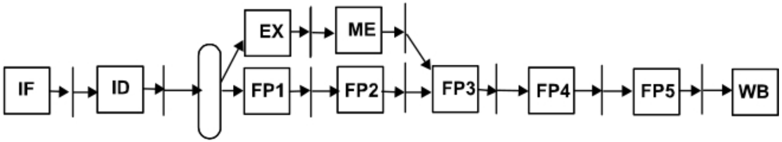
Problem 3.10 (Part b)



Given those forwarding paths, indicate all checks that must be done in the hazard detection unit associated with ID to solve RAW hazards. Your solutions specifying the hazard detection logic should be written as follows for RAW hazards on registers:

- If Integer arithmetic/logic/Store or Load instruction in ID check $<$ pipeline registers $>$
- If FP arithmetic instruction in ID check $<$ pipeline registers $>$
- If FP Store instruction in ID check $<$ pipeline registers $>$

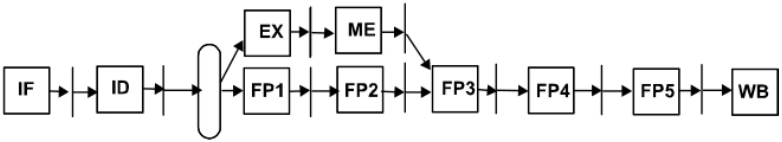
Problem 3.10 (Part b) Solution



If Integer arithmetic/logic/Store or Load instruction in ID

- check ID/EX(for preceding loads)

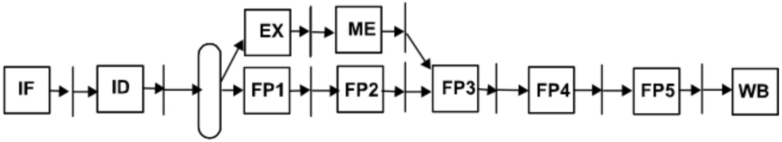
Problem 3.10 (Part b) Solution



If FP arithmetic instruction in ID

- check ID/EX (for preceding loads)
- check ID/FP1
- check FP1/FP2
- check FP2/FP3
- check FP3/FP4

Problem 3.10 (Part b) Solution



Remaining RAW hazards.

If FP Store instruction in ID check

- check ID/EX (for loads returning address)
- check ID/FP1
- check FP1/FP2
- check FP2/FP3
- check FP3/FP4

Problem 3.10 (Part c)

This architecture still has a subtle problem with respect to exception handling. Namely Stores are executed early and modify memory before they retire in the write-back stage. What is the problem.
Can you propose a solution to this problem? (Please do not propose the solution of saving the memory value and then restoring it upon an exception.)

Problem 3.10 (Part c) Solution

- Stores update machine state early
- One simple solution is to stall stores in ID until it is determined that no previous instruction currently in the pipeline can trigger an exception.

Agenda

- Problem 3.8
- Problem 3.10
- Problem 3.11

Problem 3.11 (a)

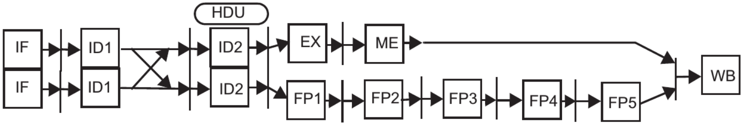


Figure 3.45. Two-way superscalar CPU

Consider the super-scalar architecture of Figure 3.45. Two consecutive instructions are fetched at a time, incrementing PC by 8. To simplify pipeline interlocks, we split the decode stage into two stages ID1 and ID2. A switch with two settings (straight and across) separates ID1 and ID2. Upper ID2 must be an integer/branch instruction or an FP Load/Store. Lower ID2 must be an arithmetic FP instruction.
Let I1 be the upper instruction and I2 be the lower instruction in ID1. I1 must proceed to ID2 before or at the same time as I2 is allowed to proceed in order to adhere to process order. The following is done in ID1:

Problem 3.11 (a)

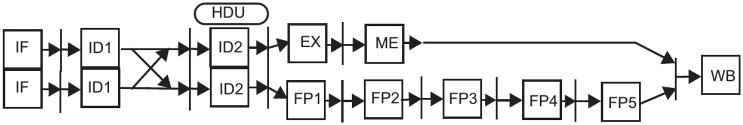


Figure 3.45. Two-way superscalar CPU

- If I1 is an integer branch instruction or an FP Load Store or a NOOP and I2 does not depend on I1 then set switch to straight.
- If I1 is an FP Load and I2 is an instruction using the value returned by the Load, stall I2 in ID1 and move I1 to ID2 with switch set to straight (Lower ID2 is NOOPed).
- If I1 is an FP arithmetic instruction, stall I2, and move I1 to ID2 with switch set to across (Upper ID2 is NOOPed).
- If I1 and I2 are both an integer/branch instruction or an FP Load/Store, stall I2 in ID1 and move I1 to ID2 with switch set to straight.(Lower ID2 is NOOPed)
- If I2 is an integer/branch instruction or an FP Load/Store and I1 is a NOOP, move I1, I2 to ID2 with switch set to across.

Problem 3.11 (a)

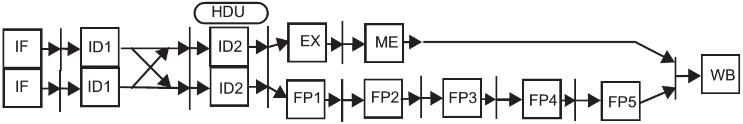


Figure 3.45. Two-way superscalar CPU

Thus if the two fetched instructions are dependent or are the wrong pair, they are serialized in ID1. Instructions in ID2 are subject to stalls due to pipeline hazard as in the single issue processor and proceed if they have no data hazard with previous instructions still in the pipeline. We deploy the same forwarding paths as in Figure 3.8. When instruction(s) are stalled in ID2, then instructions in IF and ID1 are stalled as well.

Describe briefly the function of the HDU associated with ID2 ?

Problem 3.11 (a)

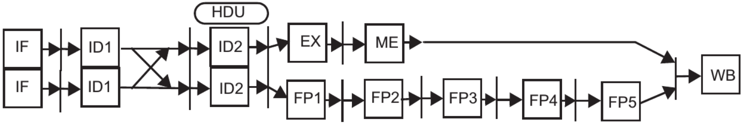


Figure 3.45. Two-way superscalar CPU

Question: Describe briefly the function of the HDU associated with ID2 ?

Awnser: The HDU associated with ID2 simply checks for hazards with instructions currently in the pipeline for both instructions in ID2.

Problem 3.11 (b)

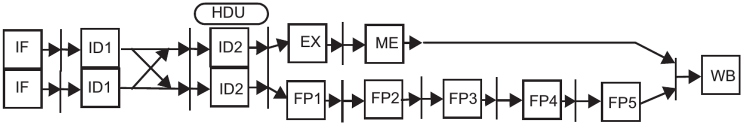


Figure 3.45. Two-way superscalar CPU

Question : Explain how a branch is processed (consider both cases when the branch is upper or lower in ID1), assuming that branches are always predicted untaken by the hardware. ?

Problem 3.11 (b)

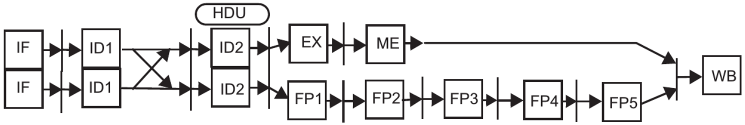


Figure 3.45. Two-way superscalar CPU

Question : Explain how a branch is processed (consider both cases when the branch is upper or lower in ID1), assuming that branches are always predicted untaken by the hardware. ?

Awnser :

- A branch is resolved in EX
- Branches are always predicted untaken so if its evaluates to be taken we need to flush pipeline
- Following instructions are in IF, ID1, ID2 and FP1
- Total 7 instructions

Problem 3.11 (c)

```
LOOP:
L.D F2,0(R1)
ADD.D F4,F2,F4
L.D F6, -8(R1)
ADD.D F8,F6,F4
S.D F8, 0(R1)
SUBI R1,R1,16
BNEZ R1, LOOP
```

Compare the execution times of one iteration of this loop (not the last iteration) on this machine and the machines of Problem 3.8 and 3.9. For the machine of Problem 3.9 assume that branches can be resolved in EX1.

Problem 3.11 (c)

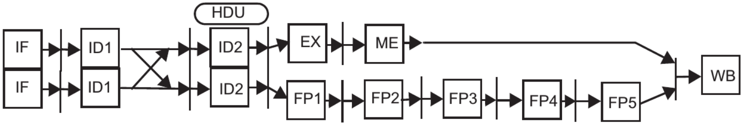


Figure 3.45. Two-way superscalar CPU

Upper ID2	Lower ID2	Cycles	Comments
L.D F2,0(R1)	(NOOP)	1	
(NOOP)	ADD.D F4,F2,F4	2	Rule2: Serialize in ID1,Wait for F2 in ID2
L.D F6, -8(R1)	(NOOP)	1	
(NOOP)	ADD.D F8,F6,F4	4	Rule2: Serialize in ID1,Waiting for F4 & F6
S.D F8, 0(R1)	(NOOP)	5	Rule 4,5: Serialize in ID1,Waiting for F8
SUBI R1,R1,16	(NOOP)	1	
BNEZ R1, LOOP	(NOOP)	1+3	Waiting for R1,+3 cycles for miss-prediction
		18	

Problem 3.11 (c) Pipeline from problem 3.8

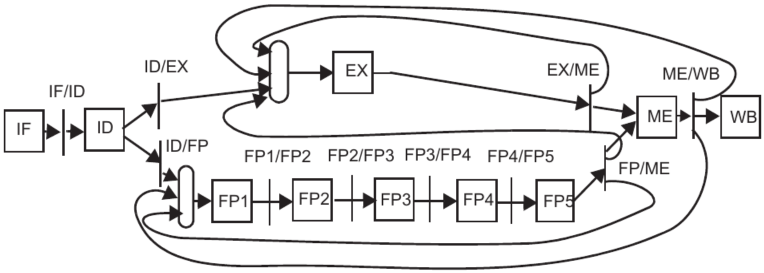


Figure 3.8. Pipeline with out-of-order execution completion

Instruction	Cycles	Comments
L.D F2,0(R1)	1	
ADD.D F4,F2,F4	2	Waiting for F2
L.D F6, -8(R1)	1	
ADD.D F8,F6,F4	4	Waiting for F4 & F6
S.D F8, 0(R1)	5	Waiting for F8, Avoiding Structural hazard on ME, WB
SUBI R1,R1,16	1	
BNEZ R1, LOOP	1+2	+2 cycles for miss-prediction
	17	

Problem 3.11 (c) Pipeline from problem 3.9

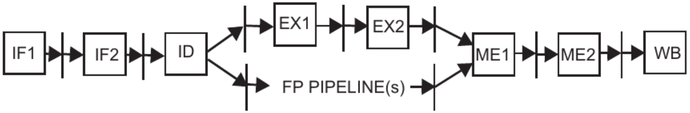


Figure 3.10. Superpipelined CPU

Instruction	Cycles	Comments
L.D F2,0(R1)	1	
ADD.D F4,F2,F4	4	Waiting for F2
L.D F6, -8(R1)	1	
ADD.D F8,F6,F4	4	Waiting for F4 & F6
S.D F8, 0(R1)	5	Waiting for F8
SUBI R1,R1,16	1	
BNEZ R1, LOOP	2+3	Waiting for R1, +3 cycles for miss-prediction
	21	