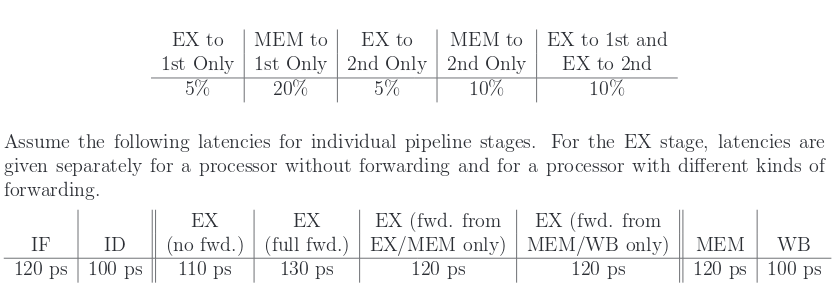
6. This exercise is intended to help you understand the cost/complexity/performance trade offs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from Figure 7.53. These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions have a particular type of RAW data dependence. The type of RAW data dependence is identified by the stage that produces the result (EX or MEM) and the next instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We assume that the register write is done in the first half of the clock cycle and that register reads are done in the second half of the cycle, so “EX to 3rd” and “MEM to 3rd” dependences are not counted because they cannot result in data hazards. We also assume that branches are resolved in the EX stage (as opposed to the ID stage), and that the CPI of the processor is 1 if there are no data hazards.



(a) For each RAW dependency listed above, give a sequence of at least 3 assembly statements

that exhibits that dependency.

EX to 1st

add $6, $5, $4

add $7, $6, $5

add $8, $9, $5

MEM to 1st

lw $6, 0($1)

add $7, $6, $5

lui $2, 0x0001

EX to 2nd

add $6, $5, $4

sub $1, $2, $3

add $7, $6, $5

MEM to 2nd

lw $5, 0($4)

add $1, $2, $3

add $6, $5, $4

EX to 1st&& EX to 2nd

add $1, $2, $3

add $5, $1, $6

add $7, $1, $6

(b) For each RAW dependency above, how many nops would need to be inserted to allow your

code from 6a to run correctly on a pipeline with no forwarding or hazard detection? Show where the

nops could be inserted.

EX to 1st

add $6, $5, $4

nop

nop

add $7, $6, $5

add $8, $9, $5

MEM to 1st

lw $6, 0($1)

nop

nop

add $7, $6, $5

lui $2, 0x0001

EX to 2nd

add $6, $5, $4

nop

sub $1, $2, $3

(or nop here)

add $7, $6, $5

MEM to 2nd

lw $5, 0($4)

nop

add $1, $2, $3

(or nop here)

add $6, $5, $4

EX to 1st&& EX to 2nd

add $1, $2, $3

nop

nop

add $5, $1, $6

add $7, $1, $6

(c) Analyzing each instruction independently will over-count the number of nops needed to

run a program on a pipeline with no forwarding or hazard detection. Write a sequence of three assembly instructions so that, when you consider each instruction in the sequence independently, the sum of the stalls is larger than the number of stalls the sequence actually needs to avoid data hazards.

Add $1, $2, $3

Add $4, $1, $2

Add $5, $4, $1

(d) Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.)

CPI = (0.05 \* 3) + (0.20 \* 3) + (0.05 \* 2) + (0.10 \* 2) + (0.10 \* 3) + (0.50 \* 1) = 1.85

stalls = .85 / 1.85 = .459 = 46% of instruction are stalls/nops

(e) What is the CPI if we use full forwarding (forward all results that can be forwarded)? What percent of cycles are stalls?

CPI = (0.05 \* 1) + (0.20 \* 2) + (0.05 \* 1) + (0.10 \* 1) + (0.10 \* 1) + (0.50 \* 1)

= (0.20 \* 2) + (0.80 \* 1) = 1.20

stalls = .20 / 1.20 = .166 = 16.6% of instruction are stalls/nops

(f) Let us assume that we cannot afford to have three-input multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the EX/MEM pipeline register (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

IF ID EX MEM WB

IF ID EX

CPI = (0.05 \* 3) + (0.20 \* 3) + (0.05 \* 2) + (0.10 \* 2) + (0.10 \* 3) + (0.50 \* 1) = 1.85

CPI EX → MEM = (0.05 \* 1) + (0.20 \* 3) + (0.05 \* 1) + (0.10 \* 2) + (0.10 \* 1) + (0.50 \* 1)

= (0.20 \* 3) + (0.10 \* 2) + (.70 \* 1)

= 1.5 CPI

CPI MEM → WB = (0.05 \* 3) + (0.20 \* 2) + (0.05 \* 2) + (0.10 \* 1) + (0.10 \* 3) + (0.50 \* 1)

= (0.15 \* 3) + (0.25 \* 2) + (0.60 \* 1)

= 1.55 CPI

So the EX → MEM forwarding would be more effective.

(g) For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by adding each type of forwarding (EX/MEM, MEM/WB, or full) to a pipeline that has no forwarding?

IF EX ID MEM WB

No forawrding = 120 + 110 + 100 + 120 + 100 = 550

speed = 550 \* 1.85 n

EX/MEM only = 120 + 110 + 120 + 120 + 100 = 570

speed = 570 \* 1.5 n

% faster = 1 - ((570 \* 1.5 n) / (550 \* 1.85 n))

= 1 – 0.8403

= 15.97% faster

MEM/WB only = 120 + 110 + 120 + 120 + 100 = 570

speed = 570 \* 1.55 n

% faster = 1 – ((570 \* 1.55 n) / (550 \* 1.85 n))

= 1 – 0.8683

= 13.17% faster

Full forwarding = 120 + 110 + 130 + 120 + 100 = 580

speed = 580 \* 1.20 n

% faster = 1 – ((580 \* 1.20 n) / (550 \* 1.85 n))

= 1 – 0.684

= 31.16 %faster

(h) What would be the additional speedup (relative to the fastest processor from part 6g) if we added time-travel forwarding that eliminates all data hazards? Assume that the yet-to-be-invented time-travel circuitry adds 100 ps to the latency of the full-forwarding EX stage.

Time Travel Magic CPU = 120 + 110 + 230 + 120 + 100 = 680

speed = 680 \* 1 n

% faster than full forwarding = 1 – ((680 \* 1 n) / (580 \* 1.20 n))

= 1 – 0.9770

= 2.35 faster than full forwarding.

(i) The table of hazard types has separate entries for “EX to 1st” and “EX to 1st and EX to 2nd”. Why is there no entry for “MEM to 1st and MEM to 2nd”?

Because by the time the memory step has has written to register for the 1st instruction it will be ready for use by the 2nd.