

Control Hazards - branch delay slots

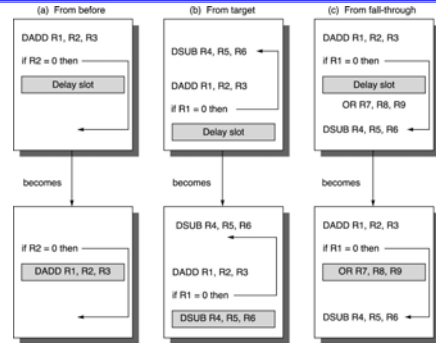
- **Reduced branch penalty:**
 - Compute condition and target address in the ID stage: 1 cycle stall.
 - Target and condition computed even when instruction is not a branch.
- **Branch delay slot filling:**

move an instruction into the slot right after the branch, hoping that its execution is necessary. Three alternatives (next slide)

Limitations: restrictions on which instructions can be rescheduled, compile time prediction of taken or untaken branches.

5

What to Put in Delay Slots?



© 2003 Elsevier Science (USA). All rights reserved.

Control Hazards: Branch Prediction

- **Idea: doing something is better than waiting around doing nothing**
 - o Guess branch target, start executing at guessed position
 - o Execute branch, verify (check) your guess
 - + minimize penalty if guess is right (to zero)
 - May increase penalty for wrong guesses
 - o Heavily researched area in the last 15 years
- **Fixed branch prediction.**

Each of these strategies must be applied to all branch instructions indiscriminately.

 - Predict not-taken:
 - continue to fetch instruction without stalling;
 - do not change any state (no register write);
 - if branch is taken turn the fetched instruction into no-op, restart fetch at target address: 1 cycle penalty.

7

Control Hazards: Branch Prediction

- Predict taken: more difficult, must know target before branch is decoded. no advantage in our simple 5-stage pipeline.
- **Static branch prediction.**
 - Opcode-based: prediction based on opcode itself and related condition. Examples: MC 88110, PowerPC 601/603.
 - Displacement based prediction: if $d < 0$ predict taken, if $d \geq 0$ predict not taken. Examples: Alpha 21064 (as option), PowerPC 601/603 for regular conditional branches.
 - Compiler-directed prediction: compiler sets or clears a predict bit in the instruction itself. Examples: AT&T 9210 Hobbit, PowerPC 601/603 (predict bit reverses opcode or displacement predictions), HP PA 8000 (as option).

8

Control Hazards: Branch Prediction

- **Dynamic branch prediction**
 - Later
- **Multi-cycle operations**

9

Dealing with Multi-Cycle Operations

- So far our simplified MIPS pipeline assume "single-cycle operations"
- Fact: not all operations complete in one cycle, e.g.
 - 4-cycle FP add, 7-cycle FP multiply
 - 24-cycle FP divide
 - Cache misses
- Alternative: slow clock cycle to slowest operation
 - Not going to happen - huge performance loss

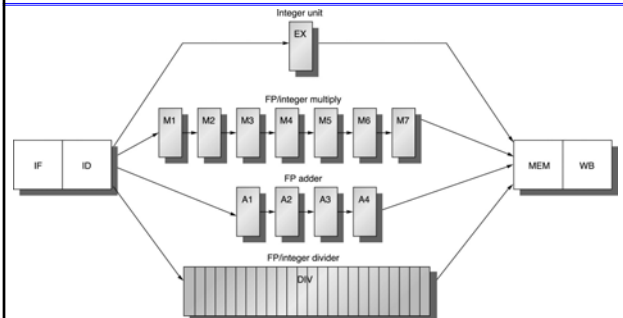
10

Multi-Cycle Operations

- Repeat EX stage multiple times
 - Multiple, parallel functional units
 - ❖ E: integer (1 cycle)
 - ❖ E+: FP adder (e.g. 2 cycles)
 - ❖ E*: FP/integer multiplier (e.g. 4 cycles)
 - ❖ E/: FP/integer divider (e.g. 20 cycles)
 - Separate integer/FP pipelines, registers
 - ❖ Load/store in integer pipeline (why?)

11

A pipeline with multi-cycle FP operations



© 2003 Elsevier Science (USA). All rights reserved.

12

Problems with Multi-Cycle Operations

Recall:

Output dependence: Write After Write (WAW) hazard (both I and J try to write same location, writes must be in-order even with no intervening reads). Did not occur in MIPS pipeline. Can occur in multicycle operation pipelines.

13

Implications of Multi-cycle Operations

B. WAW Hazards: (see example A-34)if LD is a cycle earlier its results would be overwritten by ADDD!

- If an inst between ADDD and LD uses F2 then it's a RAW and no WAW. Most WAWs occur when useless inst are executed! But, such sequences *do* occur in reasonable code.
- Options:
 - delay LD until ADDD is in MEM
 - detect hazard, make ADDD a no-op and issue LD right away.

Inst. No.	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0, F4, F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...											
ADD.D F2, F4, F6			IF	ID	A1	A2	A3	A4	MEM	WB	
...											
LD F2, 0(R2)							IF	ID	EX	MEM	WB

14

Implications of Multi-cycle Operations

• Effects on Hazards and Forwarding

- Structural hazards because of FP Divide.
- More than one register write per cycle: structural hazard.
- WAW possible: not all inst. reach WB stage in order. WAR not possible since all inst. are in ID in-order and before WB.
- Out-of-order completion of instructions: problem with interrupts (explained later).
- More frequent RAW stalls because of longer latencies.

• Solutions and Design options:

A. Register writes:

- Increase no. of write ports: costly, rarely used.
- Detect structural hazard in ID using a register to indicate reserved cycles for reg. writes (i.e. collision vectors).
- Detect collisions at the end of MEM stage: can select which inst to stall, longest latency one has higher probability of RAW hazard. Complex pipeline control.

15

Implications

B. Hazard detection: made easier by separate FP and int reg files

- Check for structural hazards: wait for DIV operation to complete before issuing a new one, make sure reg write port is available before issuing instruction.
- Check for RAW hazards: wait until all src registers are not listed as dest registers of pending instructions.
- Check for WAW hazards: any inst in A1, ... A4, D, M1, ... M7 that has same dest as current inst: stall current inst.

C. Forwarding: from EX/MEM, A4/MEM, M7/MEM, D/MEM, or MEM/WB to source registers.

16