

Lecture 6

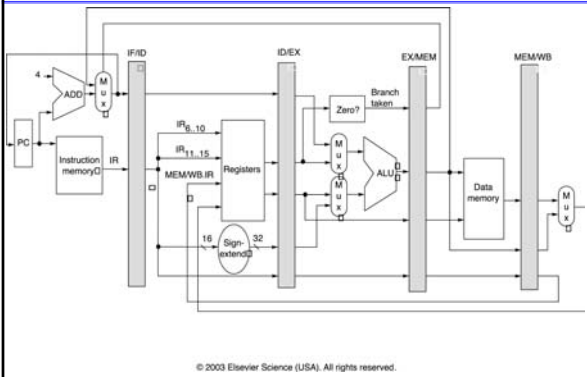
Pipeline Hazards, Exception Handling

Instructor: Jun Yang

Review

- Basic concepts of RISC pipeline
 - A simple 5-stage pipeline

Pipelined MIPS Datapath



Outline

- Pipeline hazards
 - Data hazard
 - Control hazard
- Difficulties in pipeline
 - Exception handling

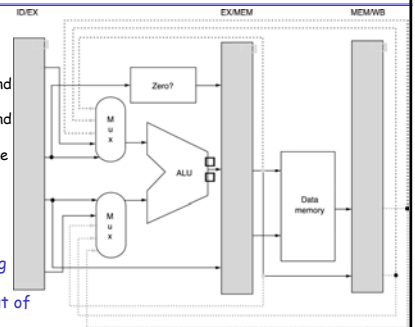
Dealing with Data Hazards

- **Forwarding (or bypassing):**
 - ❖ data values are forwarded "on the fly" to subsequent instructions in the pipeline.
- **Pipeline interlocks:**
 - ❖ detects hazards and stalls the pipeline until it is cleared: a no-op is dynamically inserted by the pipeline control.
- **Instruction scheduling.**
 - ❖ Delay slot filling: in the instruction slot immediately following a load put an independent instruction → no stalls.
 - ❖ Global optimizations reduce the effectiveness of delay slot filling.
 - ❖ FP programs have more parallelism → easier to schedule instructions.
- **Hazard detection.**
 - ❖ Instruction issue: transition from ID to EX. Hazards detected in ID by comparing source fields to destination fields of all issued instructions.
- **Forwarding logic in our simple pipeline:**
 - ❖ Detected at the beginning of the stage uses an operand (EX, MEM, etc.)
 - ❖ from ALU or data memory output to ALU, data memory or zero detection unit inputs;

Forwarding Implementation Example

- Forwarding of results to ALU from:
1. ALU output at the end of EX
 2. ALU output at the end of MEM
 3. Memory output at the end of MEM

What about forwarding result from the end of MEM stage to the input of memory?



Outline

- Pipeline hazards
 - Data hazard
 - **Control hazard**
- Difficulties in pipeline
 - Exception handling

Oct. 14, 2003

Lec. 6

7

Control Hazards

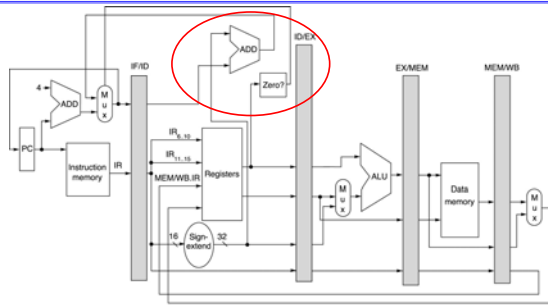
- **Branch problem:**
 - branches are resolved in MEM stage
 - 3 cycles penalty on taken branches and 2 cycles otherwise.
 - Ideal CPI = 1. Assuming 3 cycles for all branches and 32% branch instructions → new CPI = $1 + 0.32 \cdot 3 = 1.96$
- **Solutions:**
 - Reduce branch penalty: change the datapath.
 - New adder needed in ID stage.
 - Fixed branch prediction.
 - Static branch prediction.
 - Dynamic branch prediction.
 - Fill branch delay slot(s) with a useful instruction.

Oct. 14, 2003

Lec. 6

8

Change Datapath to reduce Branch Penalty



Oct. 14, 2003

© 2003 Elsevier Science B.V. All rights reserved.
Lec. 6

9

Control Hazards - 2

- **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.
- **Fixed branch prediction.**

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

 - No prediction: interlock pipeline. Very simple, one cycle stall.
 - 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.

Oct. 14, 2003

Lec. 6

10

Control Hazards - 3

-Predict taken: no advantage in our simple 5-stage pipeline; is only useful when target has to be computed before the condition is known (e.g. machines with implicitly set condition code).

- **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).

Oct. 14, 2003

Lec. 6

11

Control Hazards - 4

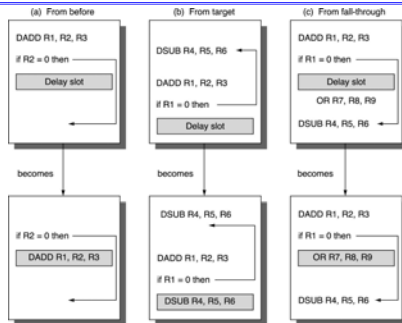
- **Dynamic branch prediction.**
 - Based on past history of same instruction.
 - In correlation to the behavior of prior branches.
 - More on this in chapter 3.
- **Branch delay slot filling:**
 - move an instruction (that is executed whether the branch is taken or not) into the branch delay slot. Three alternatives (next slide).
 - Filling >1 delay slots is difficult.
 - **Limitations:** restrictions on which instructions can be rescheduled, compile time prediction of taken or untaken branches.
 - ❖ **Canceling branches:** instruction indicates whether branch is predicted taken or not. If prediction wrong: instruction in delay slot is canceled.
- **Example** (read the ex. on page A-24)

Oct. 14, 2003

Lec. 6

12

Control Hazards - 5



Oct. 14, 2003

13

Outline

- Pipeline hazards
 - Data hazard
 - Control hazard
- Difficulties in pipeline
 - Exception handling

Oct. 14, 2003

Lec. 6

14

What Makes Pipelining Hard to Implement

- Exceptions (interrupts, faults)
 - ❖ I/O request
 - ❖ OS calls
 - ❖ Instruction tracing
 - ❖ Breakpoint
 - ❖ Arithmetic overflow or underflow
 - ❖ Page fault
 - ❖ Memory protection violation
 - ❖ Undefined/unimplemented operation
 - ❖ Hardware malfunctions
 - ❖ Power failure
- Instruction set complications

Oct. 14, 2003

Lec. 6

15

Interrupts

- **Characteristics of Interrupts** (see A.27 on Pg. A-42)
 - Synchronous & asynchronous: synchron. are from within a processor, reproducible; asynchronous are external.
 - User requested & coerced.
 - User maskable or not.
 - Within or between instructions.
 - Resume or terminate.
- Interrupts that occur **between** instructions are easy: hardware triggered procedure call.
- Interrupts that are **within** instructions are hard:
 - ❖ Control logic must check for exceptions at every state.
 - ❖ Save current state and invoke another program to handle it.
 - ❖ Restore previous state of instruction execution.
 - ❖ Resume instruction execution.

Oct. 14, 2003

Lec. 6

16

Interrupts (2)

- **The Complications**
 - Even though interrupts are rare, hardware must be designed to cope with them correctly and efficiently.
 - **Restartable computers:** recover from exceptions and continue to execute (many early microprocessors were not).
 - **Precise exception:** all instructions preceding interrupting one have completed, none of the following ones has changed the machine state.
 - Most new processors have a dual operation mode: precise and non (Alpha, MIPS R 8000 and R 10K, Power2).
 - Precise interrupts is needed for demand paging (often it is only in the integer pipeline) and for the IEEE Floating-Point standard compliance.

Oct. 14, 2003

Lec. 6

17

Interrupts (3)

- **Implementation issues:**
 - Difficult case - internal interrupt and must be restartable.
 - Actions:
 - ❖ Force a trap in IF in next cycle;
 - ❖ Disable all writes in pipeline following the faulting instruction;
 - ❖ Save restart state (PC of faulting instruction). Problem: if fault inst. is in branch delay slot and branch is taken! Must save PCs of delay slot size + 1.
 - Pipelined implementations: k concurrent interrupts:

IF	Page fault, misaligned access, memory protection.
ID	Illegal opcode.
EX	Arithmetic interrupt.
MEM	Same as in IF.

Oct. 14, 2003

Lec. 6

18

Interrupts (4)

Example:

	T+1	T+2	T+3	T+4	T+5	T+6
LW	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

- Easy case: LW page faults at MEM and ADD has exception at EX. Service page fault first, restart, exception will reappear.
- Difficult case: LW has page fault in MEM and ADD page faults in IF, out of order interrupts.
 - ❖ Precise approach: service interrupts in order
 - Create an interrupt status vector for each instruction showing stage of occurrence of interrupt.
 - Check for interrupts at MEM to WB transition, before update to state.
 - ❖ Less precise approach: service interrupts out-of order.

ISA can be hard for exceptions and pipelining

- Instructions that modify machine state in the middle of execution may cause imprecise exceptions.
 - VAX string copy inst. make the working storage in register.
 - In general, providing precise handling is costly and difficult.
- Complex instructions with memory addressing and multiple operands may cause multiple interrupts in a single instruction.
- Long running inst. make the pipeline design difficult because it introduces enormous number of hazards and forwarding conditions.
 - *Microinstruction*: a simple instruction used in sequences to implement a more complex instruction set (Intel, VAX 8800 etc.)
 - ❖ Easy for pipelining.
 - ❖ They look like MIPS!