CMSC 611: Advanced Computer Architecture

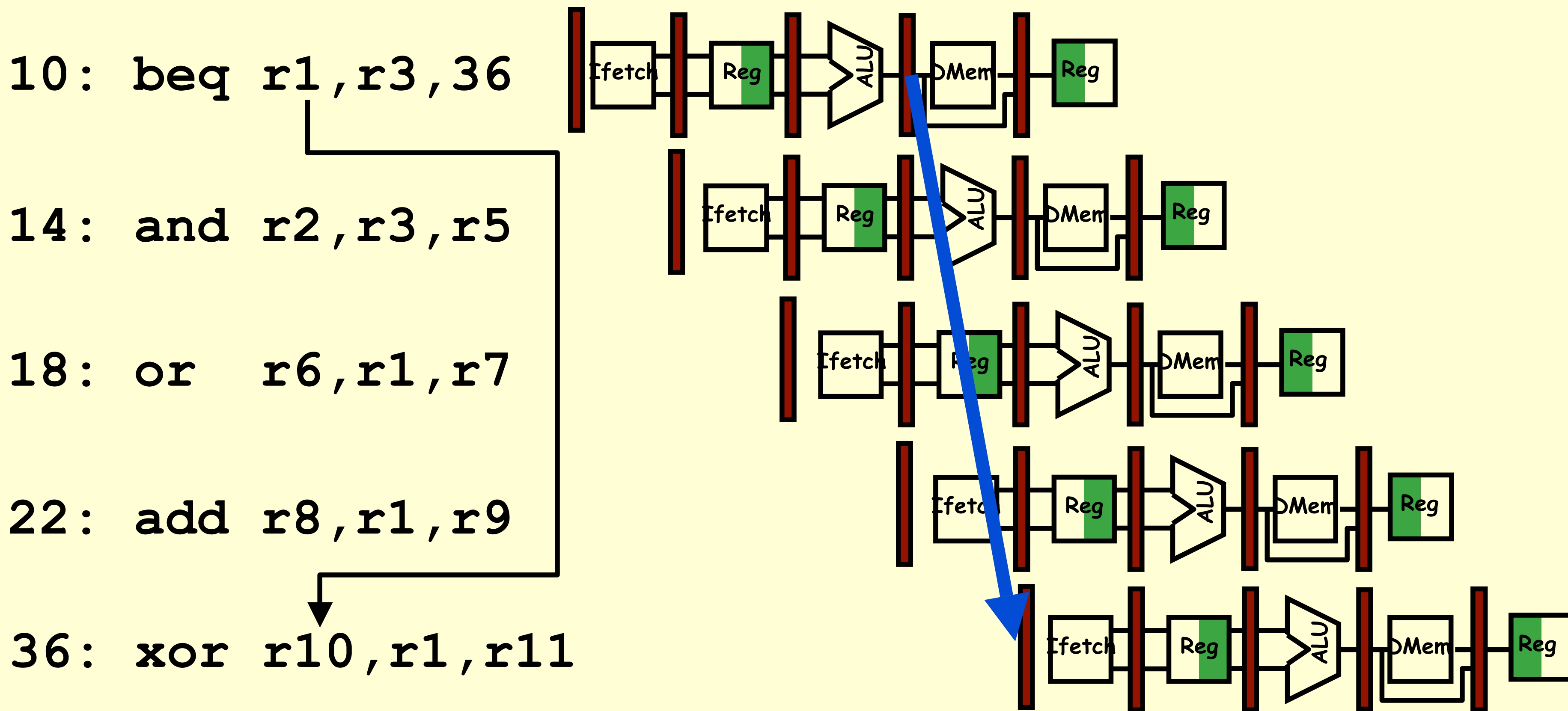
Pipelining & Instruction Level Parallelism

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

Pipeline Hazards

- Cases that affect instruction execution semantics and thus need to be detected and corrected
- Hazards types
  - Structural hazard: attempt to use a resource two different ways at same time
    - Single memory for instruction and data
  - Data hazard: attempt to use item before it is ready
    - Instruction depends on result of prior instruction still in the pipeline
  - Control hazard: attempt to make a decision before condition is evaluated
    - branch instructions
- Hazards can always be resolved by waiting

Control Hazard on Branches Three Stage Stall



Slide: David Culler

Example: Branch Stall Impact

- If 30% branch, 3-cycle stall significant!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or ≠ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Slide: David Culler

Pipelined MIPS Datapath

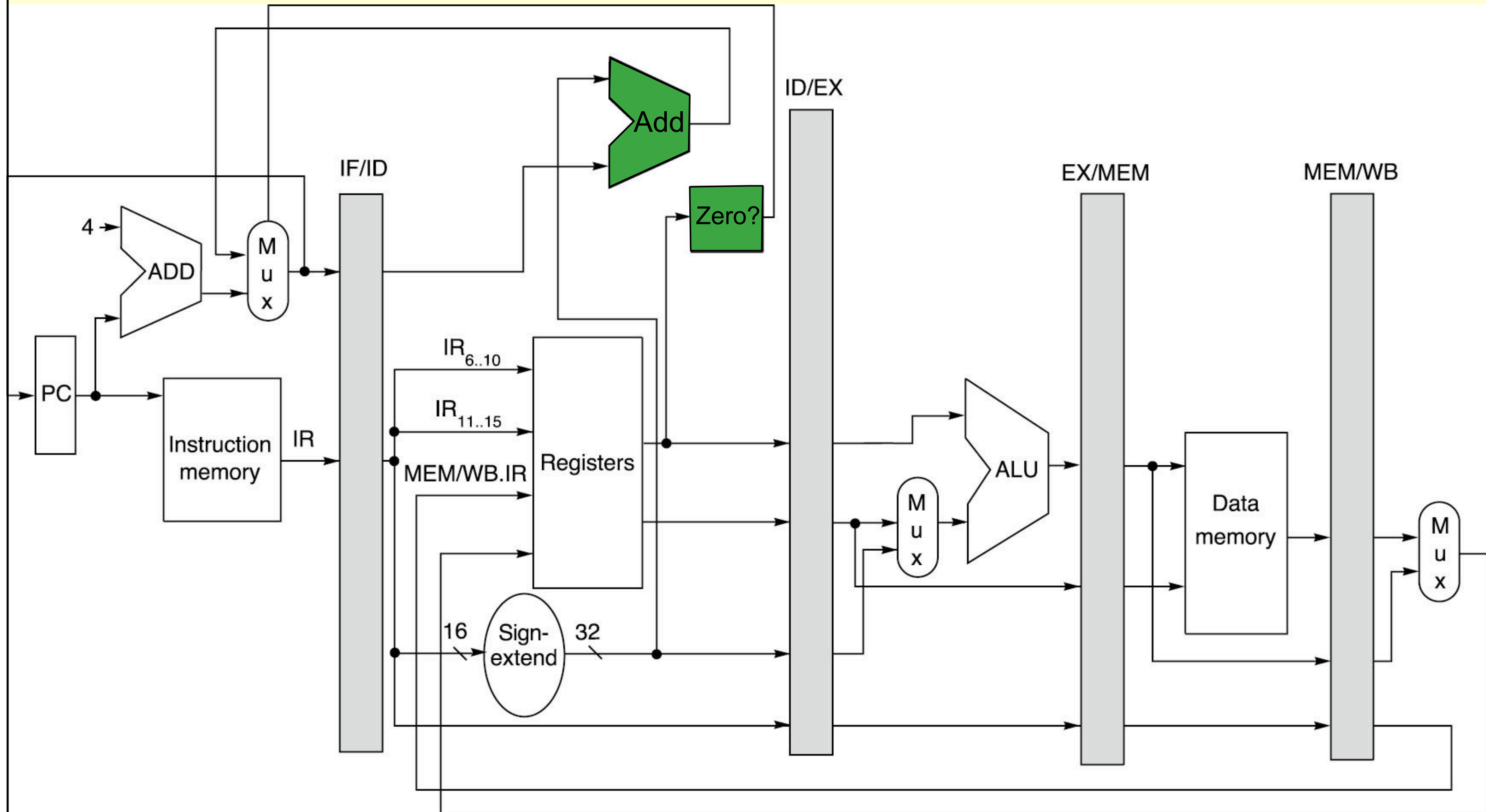


Figure: Dave Patterson

Four Branch Hazard Alternatives

1. Stall until branch direction is clear
2. Predict Branch Not Taken
   - Execute successor instructions in sequence
   - “Squash” instructions in pipeline if branch taken
   - Advantage of late pipeline state update
   - 47% MIPS branches not taken on average
   - PC+4 already calculated, so use it to get next instruction
3. Predict Branch Taken
   - 53% MIPS branches taken on average
   - But haven’t calculated branch target address in MIPS
     - MIPS still incurs 1 cycle branch penalty
     - Other machines: branch target known before outcome

Slide: David Culler

Four Branch Hazard Alternatives

4. Delayed Branch
   – Define branch to take place AFTER a following instruction

     branch instruction

     sequential successor$_1$
     sequential successor$_2$
     ....
     sequential successor$_n$

     (Branch delay of length *n*)

     ....

     branch target if taken

   – 1 slot delay allows proper decision and branch target address in 5 stage pipeline
   – MIPS uses this

Slide: David Culler

Delayed Branch

- Where to get branch delay slot instructions?
  - Before branch instruction
  - From the target address
    - only valuable when branch taken
  - From fall through
    - only valuable when branch not taken
  - Canceling branches allow more slots to be filled
- Compiler effectiveness for single delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - 48% (60% x 80%) of slots usefully filled
- Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)

Slide: David Culler

Example: Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}}$$

$$= \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Assume:

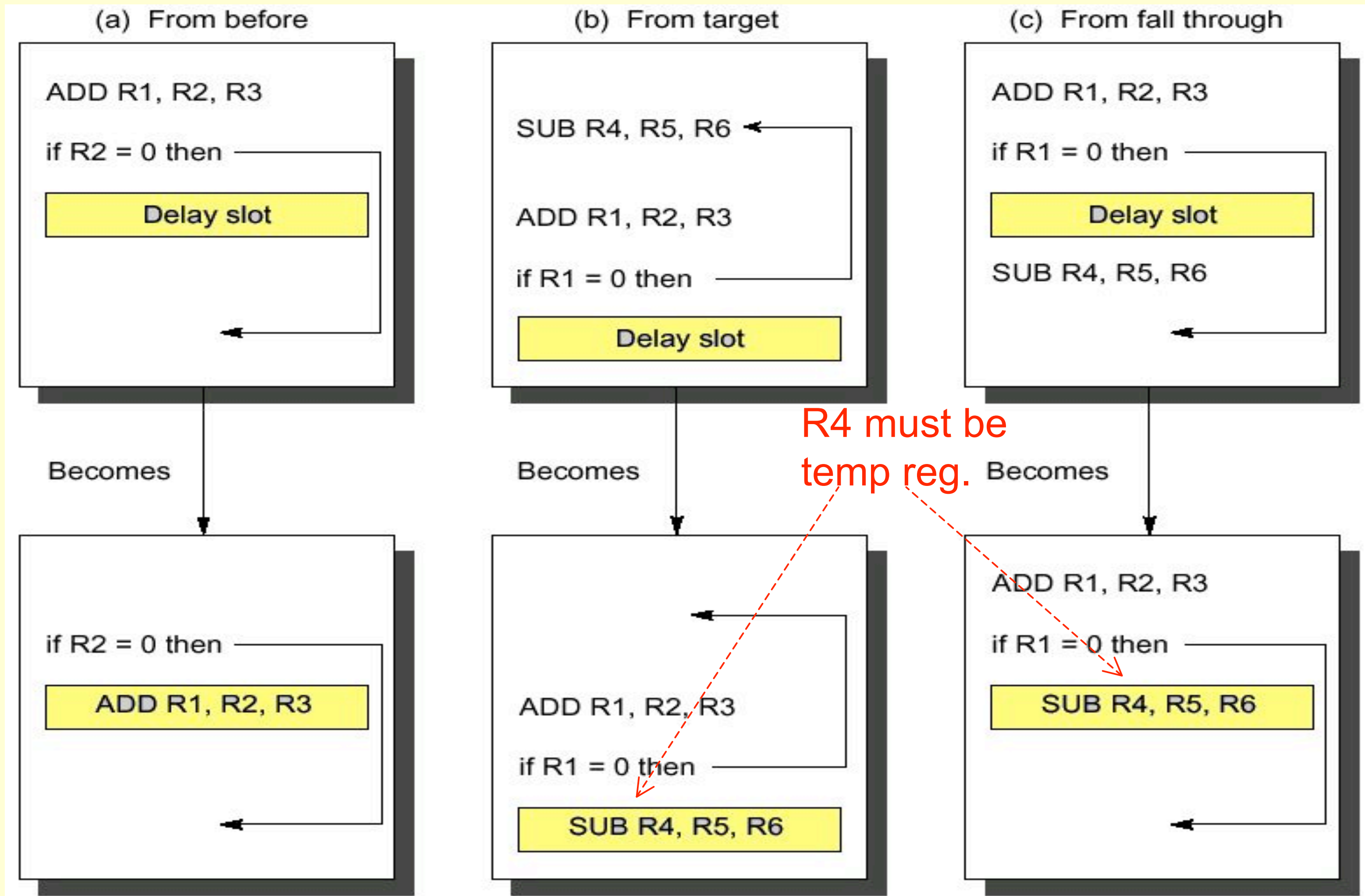
14% Conditional & Unconditional

65% Taken; 48% Delay slots not usefully filled

Scheduling Scheme	*Branch Penalty*	*CPI*	*Pipeline Speedup*	*Speedup vs stall*
Stall pipeline	3.00	1.42	3.52	1.00
Predict taken	1.00	1.14	4.39	1.25
Predict not taken	1.00	1.09	4.58	1.30
Delayed branch	0.48	1.07	4.69	1.39

Slide: David Culler

Scheduling Branch-Delay Slots



Best scenario **Good for loops** **Good taken strategy**

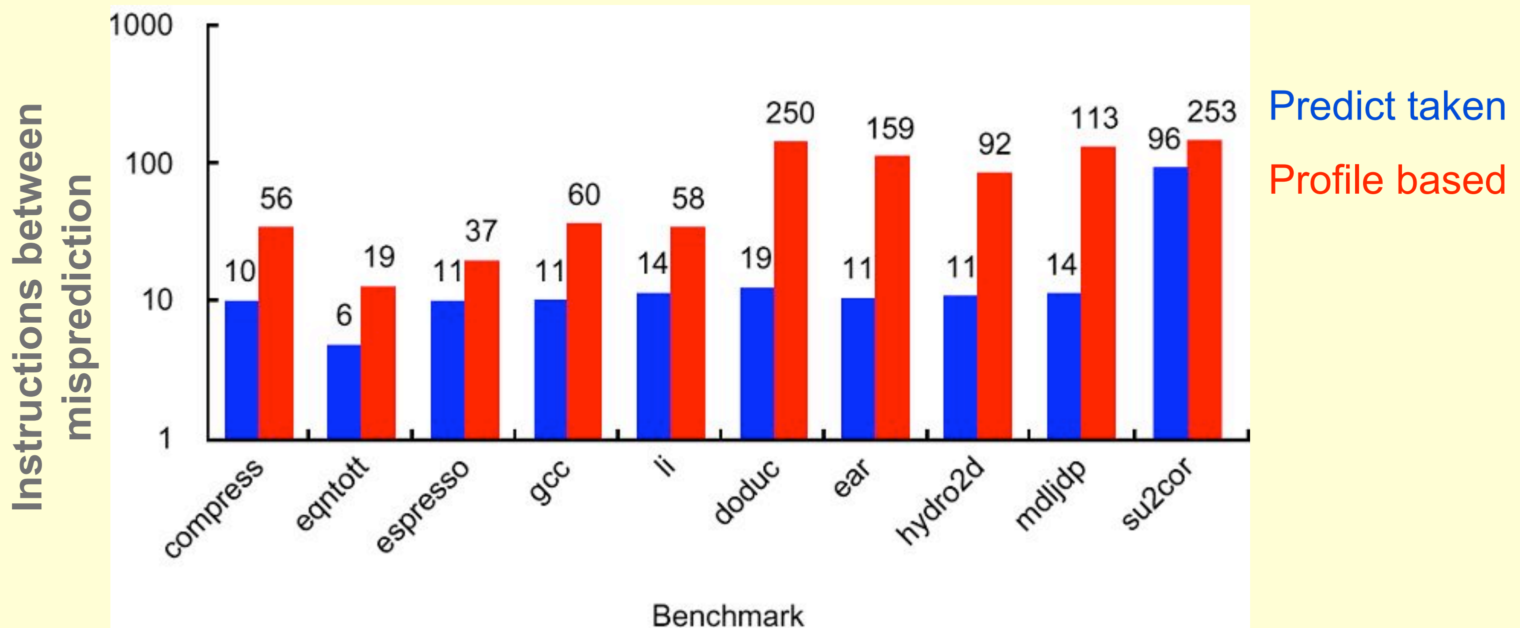
Branch-Delay Scheduling Requirements

Scheduling Strategy	Requirements	Improves performance when?
(a) From before	Branch must not depend on the rescheduled instructions	Always
(b) From target	Must be OK to execute rescheduled instructions if branch is not taken. May need to duplicate instructions.	When branch is taken. May enlarge programs if instructions are duplicated.
(c) From fall through	Must be okay to execute instructions if branch is taken.	When branch is not taken.

- Limitation on delayed-branch scheduling arise from:
  - Restrictions on instructions scheduled into the delay slots
  - Ability to predict at compile-time whether a branch is likely to be taken
- May have to fill with a no-op instruction
  - Average 30% wasted
- Additional PC is needed to allow safe operation in case of interrupts (more on this later)

Static Branch Prediction

- Examination of program behavior
  - Assume branch is usually taken based on statistics but misprediction rate still 9%-59%
- Predict on branch direction forward/backward based on statistics and code generation convention
  - Profile information from earlier program runs



Exception Types

- I/O device request
- Breakpoint
- Integer arithmetic overflow
- FP arithmetic anomaly
- Page fault
- Misaligned memory accesses
- Memory-protection violation
- Undefined instruction
- Privilege violation
- Hardware and power failure

Exception Requirements

- Synchronous vs. asynchronous
  - I/O exceptions: Asyncronous
    - Allow completion of current instruction
  - Exceptions within instruction: Synchronous
    - Harder to deal with
- User requested vs. coerced
  - Requested predictable and easier to handle
- User maskable vs. unmaskable
- Resume vs. terminate
  - Easier to implement exceptions that terminate program execution

Stopping & Restarting Execution

- Some exceptions require restart of instruction
  - e.g. Page fault in MEM stage
- When exception occurs, pipeline control can:
  - Force a trap instruction into next IF stage
  - Until the trap is taken, turn off all writes for the faulting (and later) instructions
  - OS exception-handling routine saves faulting instruction PC

Stopping & Restarting Execution

- Precise exceptions
  - Instructions before the faulting one complete
  - Instructions after it restart
  - As if execution were serial
- Exception handling complex if faulting instruction can change state before exception occurs
- Precise exceptions simplifies OS
- Required for demand paging

Exceptions in MIPS

Pipeline Stage	Problem exceptions occurring
IF	Page fault on instruction fetch; misaligned memory access; memory protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory protection violation
WB	None

- Multiple exceptions might occur since multiple instructions are executing
  - (LW followed by DIV might cause page fault and an arith. exceptions in same cycle)
- Exceptions can even occur out of order
  - IF page fault before preceeding MEM page fault

Pipeline exceptions must follow order of execution of faulting instructions not according to the time they occur

Precise Exception Handling

- The MIPS Approach:
  - Hardware posts all exceptions caused by a given instruction in a status vector associated with the instruction
  - The exception status vector is carried along as the instruction goes down the pipeline
  - Once an exception indication is set in the exception status vector, any control signal that may cause a data value to be written is turned off
  - Upon entering the WB stage the exception status vector is checked and the exceptions, if any, will be handled according the time they occurred
  - Allowing an instruction to continue execution till the WB stage is not a problem since all write operations for that instruction will be disallowed

Instruction Set Complications

- Early-Write Instructions
  - MIPS only writes late in pipeline
  - Machines with multiple writes usually require capability to rollback the effect of an instruction
    - e.g. VAX auto-increment,
  - Instructions that update memory state during execution, e.g. string copy, may need to save & restore temporary registers
- Branching mechanisms
  - Complications from condition codes, predictive execution for exceptions prior to branch
- Variable, multi-cycle operations
  - Instruction can make multiple writes