Control Hazards

- Branches (conditional, unconditional, call-return)
- Interrupts: asynchronous event (e.g., I/O)
  - Occurrence of an interrupt checked at every cycle
  - If an interrupt has been raised, don’t fetch next instruction, flush the pipe, handle the interrupt (see later in the quarter)
- Exceptions (e.g., arithmetic overflow, page fault etc.)
  - Program and data dependent (repeatable), hence “synchronous”

Exceptions

- Occur “within” an instruction, for example:
  - During IF: page fault
  - During ID: illegal opcode
  - During EX: division by 0
  - During MEM: page fault; protection violation
- Handling exceptions
  - A pipeline is *restartable* if the exception can be handled and the program restarted w/o affecting execution

Precise exceptions

- If exception at instruction \( i \) then
  - Instructions \( i-1, i-2 \) etc complete normally (flush the pipe)
  - Instructions \( i+1, i+2 \) etc. already in the pipeline will be “no-oped” and will be restarted from scratch after the exception has been handled
- Handling precise exceptions: Basic idea
  - Force a *trap* instruction on the next IF
  - Turn off writes for all instructions \( i \) and following
  - When the target of the trap instruction receives control, it saves the PC of the instruction having the exception
  - After the exception has been handled, an instruction “return from trap” will restore the PC.

Precise exceptions (cont’d)

- Relatively simple for integer pipeline
  - All current machines have precise exceptions for integer and load-store operations
- Can lead to loss of performance for pipes with multiple cycles execution stage (f-p see later)

Integer pipeline (RISC) precise exceptions

- Recall that exceptions can occur in all stages but WB
- Exceptions must be treated in instruction order
  - Instruction \( i \) starts at time \( t \)
  - Exception in MEM stage at time \( t + 3 \) (treat it first)
  - Instruction \( i + 1 \) starts at time \( t + 2 \)
  - Exception in IF stage at time \( t + 3 \) (occurs earlier but treat in 2nd)

Treating exceptions in order

- Use pipeline registers
  - Status vector registers: possible exceptions carried on with the instruction.
  - Once an exception is posted, no writing (no change of state; easy in integer pipeline – just prevent store in memory)
  - When an instruction leaves MEM stage, check for exception.
Difficulties in less RISCy environments

- Due to instruction set ("long" instructions")
  - String instructions (but use of general registers to keep state)
  - Instructions that change state before last stage (e.g., autoincrement mode in Vax and update addressing in Power PC) and these changes are needed to complete inst. (require ability to back up)
- Condition codes
  - Must remember when last changed
- Multiple cycle stages (see later)