Additional pipelining topics

• Why pipelining is so hard: exception handling
• Advanced compiler techniques: loop unrolling

Exceptions * interrupts * traps * faults

• Unusual conditions that change the normal execution sequence of the instructions
• Most common types:
  – I/O interrupt
  – timer interrupt
  – request for OS services
  – arithmetic overflow or underflow
  – page fault
  – misaligned memory reference
  – memory protection violation
  – undefined instruction
  – hardware malfunction, power failure
Exceptions * interrupts * traps * faults

• Taxonomy:
  – synchronous vs. asynchronous
  – user requested vs. coerced
  – user maskable vs. non-maskable
  – within vs. between instructions
  – resume vs. terminate

Exception handling without pipelining

<table>
<thead>
<tr>
<th>instruction i</th>
<th>interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC --&gt; instruction i+1</td>
<td></td>
</tr>
<tr>
<td>instruction i+2</td>
<td></td>
</tr>
<tr>
<td>. . .</td>
<td></td>
</tr>
</tbody>
</table>

HW: saves PC and other CPU state
causes a branch to exception handler code
OS: handles the exception
calls the OS scheduler which (re)starts process
Exceptions with pipelining

Egads!! exception affects all instructions in the pipeline
Sacre Bleu!! there can be more than one exception in a single cycle
Ai yah!! in what order should they be handled??
Yikes!! How to shut down the pipeline and restart it correctly??

Exceptions with pipelining (cont)

- Force a TRAP instruction into the pipeline on the next IF.
- Turn off all writes for the faulting instruction and all that follow it in the pipeline (zero out the pipeline latches).
- TRAP will cause state to be saved and branch to OS exception handler.
Multiple exceptions with pipelining

- Multiple exceptions can occur in an order that is different from instruction order!
- One instruction can cause multiple exceptions!
- HW posts all exceptions for a given instruction in an exception status vector associated with that instruction.
- When an instruction enters WB, its vector is checked and exceptions are handled in the order they would have occurred on unpipelined machine.

Restart under pipelining

- Each posted exception is handled by the appropriate exception handler
- If pipeline can be stopped and restarted correctly, the pipeline has what are called precise exceptions.

This is just a small look at the complications of pipelining. There are many more issues that must be addressed.
MIPS64 Exceptions

- Fig. A.28

Loop Unrolling

- A compiler technique whose goal is to increase parallelism.
- Loop unrolling can decrease penalties due to control hazards
- Loop unrolling can reduce the number of instructions executed (but decrease the number of instructions in memory)
- ILP (instruction level parallelism) - unrolling loops on machine instructions
- Higher level parallelism - unrolling loops on high level code (C and Fortran)
Example:

C code: for \(i=100; \ i>=1; \ i--) \x
x[i] = x[i] + 23;

Machine code: assume R1 has addr of X[100] R2 has the value 23. R3 has the value 100.

```
LOOP  LW  R4,0(R1)
       ADD R4,R4,R2
       SW  0(R1),R4
       SUBI R1,R1,#4  decr ptr to array elmt
       SUBI R3,R3,#1  decr loop counter
       BNEZ R3,LOOP
```

Example (cont): hazards

Machine code: assume R1 has addr of X[100] R2 has the value 23. R3 has the value 100.

```
LOOP  LW  R4,0(R1)
stall
       ADD R4,R4,R2
       SW  0(R1),R4
       SUBI R1,R1,#4  decr ptr to array elmt
       SUBI R3,R3,#1  decr loop counter
       BNEZ R3,LOOP
       stall
```
Example (cont): get rid of RAW

Machine code: assume R1 has addr of X[100]  
R2 has the value 23. R3 has the value 100.

```
LOOP  LW  R4,0(R1)
       SUBI R3,R3,#1      decr loop counter
       ADD  R4,R4,R2
       SW   0(R1),R4
       SUBI R1,R1,#4      decr ptr to array elmt
       BNEZ R3,LOOP
       stall
```

Example (cont): Unroll the loop using replication

```
Example (cont): Unroll the loop using replication

LOOP  LW  R4,0(R1)
       SUBI R3,R3,#1
       ADD  R4,R4,R2
       SW   0(R1),R4
       LW   R4,-4(R1)
       SUBI R3,R3,#1
       ADD  R4,R4,R2
       SW   -4(R1),R4
       LW   R4,-8(R1)
       SUBI R3,R3,#1
       ADD  R4,R4,R2
       SW   -8(R1),R4
       LW   R4,-12(R1)
       SUBI R3,R3,#1
       ADD  R4,R4,R2
       SW   -12(R1),R4
       SUBI R1,R1,#16     decr ptr to array elmt
       BNEZ R3,LOOP
       stall <- 25 stalls instead of 100
```
Example (cont): fill the branch delay slot

```
LOOP
  LW    R4,0(R1)
  SUBI  R3,R3,#1
  ADD   R4,R4,R2
  SW    0(R1),R4
  LW    R4,-4(R1)
  SUBI  R3,R3,#1
  ADD   R4,R4,R2
  SW    -4(R1),R4
  LW    R4,-8(R1)
  SUBI  R3,R3,#1
  ADD   R4,R4,R2
  SW    -8(R1),R4
  LW    R4,0-12(R1)
  SUBI  R3,R3,#1
  ADD   R4,R4,R2
  SW    -12(R1),R4
  SUBI  R1,R1,#16      decr ptr to array elmt
  BNEZ  R3,LOOP
  SW    4(R1),R4          adjust the last SW
```
Each replication saves two branch control instructions.