**Slide 2: Stalling**

| $0x100$ | $ADD\!W$ | $F0X MW$ |
| $0x104$ | $ADD\!W$ | $F0X MW$ |
| $0x108$ | $ADD\!W$ | $F0X MW$ |
| $0x10C$ | $J 0x200$ | $F0X MW$ |
| $0x110$ | $XOR$ | $F0X MW$ |

CPI = 2!

**Slide 3: Speculate**

| $0x100$ | $ADD\!W$ | $F0x MW$ |
| $0x104$ | $ADD\!W$ | $F0x MW$ |
| $0x108$ | $ADD\!W$ | $F0x MW$ |
| $0x10C$ | $J 0x200$ | $F0x MW$ |
| $0x110$ | $XOR$ | $F0x MW$ |

Correct Speculation

Wrong Speculate

Control Hazard!

| $0x100$ | $J 0x200$ | $F0x MW$ |
| $0x10C$ | $XOR$ | $F0x MW$ |
| $0x110$ | $ADD\!W$ | $F0x MW$ |

"kill" or "squash"

**Slide 4: Jump - Kill & Stall**

| $0x100$ | $ADD\!W$ | $F0x MW$ |
| $0x104$ | $J 0x200$ | $F0x MW$ |
| $0x108$ | $ADD\!W$ | $F0x MW$ |
| $0x110$ | $ADD\!W$ | $F0x MW$ |

Stalls for some reason

Fetch with $0x108$ multiple times?

Cannot overwrite FR!

| $0x100$ | $ADD\!W$ | $F0x MW$ |
| $0x104$ | $J 0x200$ | $F0x MW$ |
| $0x108$ | $ADD\!W$ | $F0x MW$ |
| $0x10C$ | $ADD\!W$ | $F0x MW$ |

Tricky to do, why can’t stall anyway?
Activity: Branch Resolution

Static with SEQ

Ox100 ADD I
Ox104 BEQ Ox200
Ox108 ADD I
Ox110 ADD I
Ox110 ...
Ox200 ADD I

Pipeline diagram assuming branch is taken.
Assume branch condition is resolved in X.
**Activity Branch Resolution**

- Ox100 ADDIU
- Ox104 BEQZ O,0,000
- Ox108 ADDIU
- Ox10c ADDIU
- Ox200 ADDIU

**Slide 8 - Kill + Stall**

- Ox100 ADDIU r1, r2, 1
- Ox104 BEQZ r3, O,000
- Ox108 ADDIU r4, r1
- Ox10c ADDIU
- Ox200 ADDIU

Stalled in D and F, but want to kill those same instructions.

If X waits for D and F to not be stalled, that might work, but what if D and F are stalled because of the inst in X? Can this happen? Jack, Jack?

> Deadlock

Kill takes precedence over stall

In general, later inst in pipeline (ie earlier instructions in program order) take precedence over instructions earlier in pipeline
Activity - Back to Back Branch/Jumps

Which of these instructions should actually execute and in what order? Assume branch is taken and that there is a one cycle branch delay slot.

Jumps are resolved in D, branches are resolved in X

I0: ADDIU
I1: J (LABEL-X)
I2: BRANCH LABEL-Y
I3: ADDIU
I4: ADDIU
...
LABEL-X:
I5: ADDIU
I6: J DONE
I7: ADDIU
...
LABEL-Y:
I8: ADDIU
DONE:

A) 0, 1, 2, 3, 4, 5, 6, 7, 8
B) 0, 1, 5, 6, 7
C) 0, 1, 2, 8
D) 0, 1, 2, 5, 6, 7
E) 0, 1, 2, 5, 8
LATENCY

**Transaction Latency**: How many cycles to execute a transaction assuming no hazards.

**Transaction Delay Latency**: How many cycles do I need to "delay" a transaction to avoid a specific kind of hazard with respect to an earlier transaction.

**X-Use Delay Latency**: Number of cycles I need to delay a transaction to avoid a structural hazard with respect to an earlier transaction that uses X.

**X-Use Data Delay Latency**: Number of cycles I need to delay a transaction that uses the result of transaction X.

**X Control Resolution Delay Latency**: Number of cycles I need to delay before I resolve the control hazard.

---

**Five-Stage MIPS R Pipeline**

- Transaction Latency: 5 cycles
- ALU-Use Delay Latency: 0 cycles
- Load-Use Delay Latency: 1 cycle
- Jump Resolved Latency: 1 cycle
- Branch Missed Latency: 2 cycles

With 8 cycle floating multiplier:

- Non-Multiply Transaction Latency: 5 cycles
- Multiply Transaction Latency: 12 cycles
- Multiply Structural Delay Latency: 4 cycles
- Multiply-Use Delay Latency: 4 cycles

With 8 cycle pipelined multiplier:

- Non-Multiply Transaction Latency: 5 cycles
- Multiply Transaction Latency: 12 cycles
- Multiply Structural Delay Latency: 9 cycles
- Multiply-Use Delay Latency: 4 cycles
8 cycle iterative multiplier

\[ \begin{align*}
\text{m} & \text{ul } r_1, r_2, r_3 & FD & T & Z & Z & Z & Z & Z & M & W \\
\text{m} & \text{ul } r_4, r_5, r_6 & FD & D & D & D & D & D & D & Z & Z & Z & Z & M & W \\
\hline
\end{align*} \]

8 cycle multiply-use, delay category

\[ \begin{align*}
\text{m} & \text{ul } r_1, r_2, r_3 & FD & T & Z & Z & Z & Z & Z & M & W \\
\text{add} & \text{w } r_4, r_1, 1 & FD & D & D & D & D & D & D & D & X & M & W \\
\end{align*} \]

8 cycle pipeline multiplier

\[ \begin{align*}
\text{m} & \text{ul } r_1, r_2, r_3 & FD & T & Z & Z & Z & Z & Z & M & W \\
\text{m} & \text{ul } r_4, r_5, r_6 & FD & D & D & D & D & D & D & Z & M & W \\
\hline
\end{align*} \]

8 cycle multiply-use, delay category

\[ \begin{align*}
\text{m} & \text{ul } r_1, r_2, r_3 & FD & D & T & Z & Z & Z & Z & Z & M & W \\
\text{add} & \text{w } r_4, r_1, 1 & FD & D & D & D & D & D & D & D & X & M & W \\
\end{align*} \]