CIS 429/529
Hazard Resolution

Prof. Michel A. Kinsky
Resolving Data Hazards

• Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  ▸ interlocks

• Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  ▸ bypass
Resolving Data Hazards

• Strategy 3: Speculate on the dependence

  » Two cases:

    • Guessed correctly
     » do nothing
    • Guessed incorrectly
     » kill and restart
Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can stall (or kill) instructions.
- Controlling a pipeline in this manner works provided the instruction at stage $i+1$ can complete without any interference from instructions in stages 1 to $i$.
  - Otherwise deadlocks may occur.

![Diagram showing feedback from stages 1 to 4 with FB1, FB2, FB3, and FB4]
Interlocks to resolve Data Hazards

Stall Condition

... $r1 \leftarrow r0 + 10$

... $r4 \leftarrow r1 + 17$
Stalled Stages and Pipeline

\[
(I_1) \ r1 \leftarrow (r0) + 10 \\
(I_2) \ r4 \leftarrow (r1) + 17 \\
(I_3) \\
(I_4) \\
(I_5)
\]

\[
\begin{align*}
\text{time} & \quad \text{t0} & \text{t1} & \text{t2} & \text{t3} & \text{t4} & \text{t5} & \text{t6} & \text{t7} \\
(I_1) \ r1 & \leftarrow (r0) + 10 & \text{IF}_1 & \text{ID}_1 & \text{EX}_1 & \text{MA}_1 & \text{WB}_1 & \text{EX}_2 & \text{MA}_2 \\
(I_2) \ r4 & \leftarrow (r1) + 17 & \text{IF}_2 & \text{ID}_2 & \text{IF}_3 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_3 & \text{MA}_2 \\
(I_3) & & \text{IF}_3 & \text{ID}_2 & \text{IF}_3 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_3 & \text{MA}_2 \\
(I_4) & & \text{IF}_3 & \text{ID}_2 & \text{IF}_3 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_3 & \text{MA}_2 \\
(I_5) & & \text{IF}_3 & \text{ID}_2 & \text{IF}_3 & \text{ID}_2 & \text{ID}_2 & \text{ID}_2 & \text{EX}_3 & \text{MA}_2 \\
\end{align*}
\]

\[
\begin{array}{cccccccc}
\text{Resource} & \text{IF} & \text{ID} & \text{EX} & \text{MA} & \text{WB} \\
\text{Usage} & I_1 & I_2 & I_3 & I_4 & I_5 \\
\end{array}
\]

\[
\begin{align*}
\text{time} & \quad \text{t0} & \text{t1} & \text{t2} & \text{t3} & \text{t4} & \text{t5} & \text{t6} & \text{t7} \\
\text{IF} & I_1 & I_2 & I_3 & \text{nop} & \text{nop} & \text{nop} & \text{nop} & \text{nop} \\
\text{ID} & I_1 & I_2 & I_3 & \text{nop} & \text{nop} & \text{nop} & \text{nop} & \text{nop} \\
\text{EX} & I_1 & I_2 & I_3 & \text{nop} & \text{nop} & \text{nop} & \text{nop} & \text{nop} \\
\text{MA} & I_1 & I_2 & I_3 & \text{nop} & \text{nop} & \text{nop} & \text{nop} & \text{nop} \\
\text{WB} & I_1 & I_2 & I_3 & \text{nop} & \text{nop} & \text{nop} & \text{nop} & \text{nop} \\
\end{align*}
\]
Interlock Control Logic

Diagram of computer architecture and embedded systems laboratory (CAES Lab)
Resolving Data Hazards

• Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  ‣ interlocks

• Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  ‣ bypass
Interlocks to resolve Data Hazards

Stall Condition

... r1 ← r0 + 10
... r4 ← r1 + 17
...
Source & Destination Registers

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{R-type:} & \text{op} & \text{rs} & \text{rt} & \text{rd} \\
\hline
\text{I-type:} & \text{op} & \text{rs} & \text{rt} & \text{immediate16} \\
\hline
\text{J-type:} & \text{op} & \text{immediate26} \\
\hline
\end{array}
\]

source(s) destination

- **ALU**: \( \text{rd} \leftarrow (\text{rs}) \text{ func (rt)} \) \( \text{rs, rt} \rightarrow \text{rd} \)
- **ALUi**: \( \text{rt} \leftarrow (\text{rs}) \text{ op imm} \) \( \text{rs} \rightarrow \text{rt} \)
- **LW**: \( \text{rt} \leftarrow \text{M [(rs) + imm]} \) \( \text{rs} \rightarrow \text{rt} \)
- **SW**: \( \text{M [(rs) + imm]} \leftarrow (\text{rt}) \) \( \text{rs, rt} \)
- **BZ**: \( \text{cond (rs)} \)
  - \( \text{true: PC} \leftarrow (\text{PC}) + \text{imm} \) \( \text{rs} \)
  - \( \text{false: PC} \leftarrow (\text{PC}) + 4 \) \( \text{rs} \)
- **J**: \( \text{PC} \leftarrow (\text{PC}) + \text{imm} \)
- **JAL**: \( \text{r31} \leftarrow (\text{PC}), \text{PC} \leftarrow (\text{PC}) + \text{imm} \) \( \text{r31} \)
- **JR**: \( \text{PC} \leftarrow (\text{rs}) \) \( \text{rs} \)
- **JALR**: \( \text{r31} \leftarrow (\text{PC}), \text{PC} \leftarrow (\text{rs}) \) \( \text{rs} \rightarrow \text{r31} \)
Deriving the Stall Signal

\( C_{\text{dest}} \)
\[
ws = \text{Case opcode} \\
\text{ALU} \rightarrow rd \\
\text{ALUi, LW} \rightarrow rt \\
\text{JAL, JALR} \rightarrow R31
\]
\[
we = \text{Case opcode} \\
\text{ALU, ALUi, LW} \rightarrow (ws \neq 0) \\
\text{JAL, JALR} \rightarrow \text{on} \\
\ldots \rightarrow \text{off}
\]

\( C_{\text{re}} \)
\[
re1 = \text{Case opcode} \\
\text{ALU, ALUi,} \\
\text{LW, SW, BZ,} \\
\text{JR, JALR} \rightarrow \text{on} \\
\text{J, JAL} \rightarrow \text{off}
\]
\[
re2 = \text{Case opcode} \\
\text{ALU, SW} \rightarrow \text{on} \\
\ldots \rightarrow \text{off}
\]

\( C_{\text{stall}} \)
\[
stall = ((rs_D = ws_E).we_E + \)
\quad (rs_D = ws_M).we_M + \)
\quad (rs_D = ws_W).we_W).re1_D + \)
\quad ((rt_D = ws_E).we_E + \)
\quad (rt_D = ws_M).we_M + \)
\quad (rt_D = ws_W).we_W).re2_D
\]
Resolving Data Hazards

• Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  ‣ interlocks

• Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  ‣ bypass
Bypassing

Each stall or kill introduces a bubble in the pipeline $\rightarrow$ CPI $>$ 1

- Each stall or kill introduces a bubble in the pipeline $\rightarrow$ CPI $>$ 1
Adding a Bypass

When does this bypass help?

\((I_1) \ r_1 \leftarrow r_0 + 10\)  \(r_1 \leftarrow M[r_0 + 10]\)  \(\text{JAL 500}\)

\((I_2) \ r_4 \leftarrow r_1 + 17\)  \(r_4 \leftarrow r_1 + 17\)  \(r_4 \leftarrow r_31 + 17\)
The Bypass Signal

\[ C_{\text{dest}} \]

\[ ws = \text{Case opcode} \]
- ALU \rightarrow rd
- ALUi, LW \rightarrow rt
- JAL, JALR \rightarrow R31

\[ we = \text{Case opcode} \]
- ALU, ALUi, LW \rightarrow (ws \neq 0)
- JAL, JALR \rightarrow \text{on}
- ... \rightarrow \text{off}

\[ C_{\text{re}} \]

\[ re1 = \text{Case opcode} \]
- ALU, ALUi,
- LW, SW, BZ,
- JR, JALR \rightarrow \text{on}
- J, JAL \rightarrow \text{off}

\[ re2 = \text{Case opcode} \]
- ALU, SW \rightarrow \text{on}
- ... \rightarrow \text{off}

\[ C_{\text{stall}} \]

\[ \text{stall} = ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W) \cdot re_{1D} \]
\[ + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W) \cdot re_{2D} \]

\[ \text{ASrc} = (rs_D = ws_E).we_E . re_{1D} \]

Is this correct?
Bypass and Stall Signals

- Split \( w_{E} \) into two components: \( w_{E} \)-bypass, \( w_{E} \)-stall

\[
\begin{align*}
\text{we-bypass}_{E} &= \text{Case opcode}_{E} \\
\text{ALU, ALU}_{i} &\rightarrow (ws \neq 0) \\
... &\rightarrow \text{off}
\end{align*}
\]

\[
\begin{align*}
\text{we-stall}_{E} &= \text{Case opcode}_{E} \\
\text{LW} &\rightarrow (ws \neq 0) \\
\text{JAL, JALR} &\rightarrow \text{on} \\
... &\rightarrow \text{off}
\end{align*}
\]

\[
\text{ASrc} = (rs_{D} = ws_{E}).\text{we-bypass}_{E} \cdot \text{re1}_{D}
\]

\[
\text{stall} = ((rs_{D} = ws_{E}).\text{we-stall}_{E} + (rs_{D} = ws_{M}).\text{we}_{M} + \\
(rs_{D} = ws_{W}).\text{we}_{W}). \text{re1}_{D} + ((rt_{D} = ws_{E}).\text{we}_{E} + \\
(rt_{D} = ws_{M}).\text{we}_{M} + (rt_{D} = ws_{W}).\text{we}_{W}). \text{re2}_{D}
\]
Is there still a need for the stall signal?

\[
\text{stall} = (\text{rs}_D = \text{ws}_E). (\text{opcode}_E = \text{LW}_E). (\text{ws}_E \neq 0). \text{re1}_D \\
+ (\text{rt}_D = \text{ws}_E). (\text{opcode}_E = \text{LW}_E). (\text{ws}_E \neq 0). \text{re2}_D
\]
Why a program may have CPI > 1

• Why an Instruction may not be dispatched every cycle (CPI>1)?
  ‣ Full bypassing may be too expensive to implement
    • typically all frequently used paths are provided
    • some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
  ‣ Loads have two cycle latency
    • Instruction after load cannot use load result
    • MIPS-I ISA defined load delay slots, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard) - Removed in MIPS-II
Why a program may have CPI > 1

- Why an Instruction may not be dispatched every cycle (CPI>1)?
  - Conditional branches may cause bubbles
    - kill following instruction(s) if no delay slots
**Branch or Load Delay Slots**

- Expose control hazard to software:
  - Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  - Gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted

```
| I_1 | 096 | ADD |
| I_2 | 100 | BEQZ r1 200 |
| I_3 | 104 | ADD |
| I_4 | 304 | ADD |
```

- Delay slot instruction executed regardless of branch outcome

- Other techniques include branch prediction, which can dramatically reduce the branch penalty... to come later
Critical paths and clocks

- Hazard resolution, bypassing and other complex speculation strategies can make the control circuitry determine the length of the critical path because of communication between various pipeline stages.

- We will discuss techniques to address this problem in future ...
Next Class

• Complex Pipelining