Data Hazards

Philipp Koehn

9 October 2019
**Data Hazard**

- **Definition**: instruction waits on result from prior instruction

- **Example**

```
add $s0, $t0, $t1
sub $t0, $s0, $t3
```

  - add instruction writes result to register $s0 in stage 5
  - sub instruction reads $s0 in stage 2

⇒ Stage 2 of sub has to be delayed

- **We overcome this in hardware**
Graphical Representation

- **IF**: instruction fetch
- **ID**: instruction decode
- **EX**: execution
- **MEM**: memory access
- **WB**: write-back

add $s0,$t0,$t1
Add and Subtract

- Add wiring to circuit to directly connect output of ALU for next instruction
Load and Subtract

- Add wiring from memory lookup to ALU
- Still 1 cycle unused: "pipeline stall" or "bubble"
forwarding
Add and Subtract

• Example

```plaintext
add $s0, $t0, $t1
sub $t0, $s0, $t3
```

• Plan

![Diagram of the pipeline stages for add and subtract operations.](image)
Add (Stage 1)

IF: Instruction Fetch
ID: Instruction decoder 
MEM: Memory access
WB: Write Back

Selector
PC
Address
Instruction Memory
Instruction

Add

Select

4

Add

Instruction Fetch

Instruction decoder

Memory access

Write Back

Sign extended
Shift Left

ADD

Zero

ALU Result

Address
Data Memory

Selector

Read data

Write data

Read register 1 data 1
Write register
Read register 2 data 2
Write data

Selector

Write Back
Subtract (Stage 1), Add (Stage 2)
Subtract (Stage 2), Add (Stage 3)
Subtract (Stage 3), Add (Stage 4)
Data Hazard

IF: Instruction Fetch
- Address
- Instruction
- Memory
- Selector
- PC

ID: Instruction decoder
- register file read
- ALU
- Add
- selector
- Sign extended
- Zero
- Sign extended
- Shift Left
- Registers
- Read register 1
- Read data 1
- Write register
- Write data
- Read register 2

EX: Execute / address calculate
- ADD
- ALU
- Result
- Selector
- Address
- Data Memory
- Write data

MEM: Memory access
- Memory
- Write data

WB: Write Back
- Selector
Forwarding Data

IF: Instruction Fetch
ID: Instruction decoder
MEM: Memory access
WB: Write Back

Selector
PC

Address Memory
Instruction

Instruction
Memory

Add

Sign extended
Shift Left
Zero

Selector

ALU

Add

Instruction
Registers

Read register 1
Read data 1

Write register
Read data 2

Write data

Read register 2

Address Memory

Data Memory
Write data

Selector

Selector

Selector
Forwarding Unit

- Forwarding Unit must
  - detect if there is a data hazard
  - forward the right register values

- Relevant information for decision
  - identity of input registers used in instruction currently in EX (either first or second operand)
  - identity of output register used in instruction currently in MEM
  - value of output register used in instruction currently in MEM

- Format of decision
  - Register value
  - Control lines for selectors for input to ALU
Formal Names

• Relevant information for decision
  – EX.Rs and EX.Rt
    identify of input registers used in instruction currently in EX
    (either first or second operand)
  – MEM.Rd
    identity of output register used in instruction currently in MEM
  – MEM.RdValue
    value of output register used in instruction currently in MEM

• Format of decision
  – Forward.Rs and Forward.Rt
    Register value
  – Hazard.Rs and Hazard.Rt
    Control lines for selectors for input to ALU
Forwarding Logic

- Logic in forwarding unit

```python
if (MEM.Rd == EX.Rs)
    Forward.Rs = MEM.RdValue
    Hazard.Rs = 1
else
    Hazard.Rs = 0

if (MEM.Rd == EX.Rt)
    Forward.Rt = MEM.RdValue
    Hazard.Rt = 1
else
    Hazard.Rt = 0
```

- Must also check if "RegisterWrite" for instruction in MEM stage

- Relevant information must be passed through stages
Forwarding Unit

IF: Instruction Fetch
- Address
- Instruction Memory
- Instruction

ID: Instruction decoder
- register file read
- Read register 1
- Read register 2
- Registers
- Write register data
- Write data

EX: Execute/
- address calculate
- Selector
- Zero
- ALU
- Result
- ForwardA
- ForwardB
- Hazard A
- Hazard B

MEM: Memory access
- Forwarding Unit
stalling
Load and Subtract

• Example
  
  \[
  \text{load } s0, 20(t0) \\
  \text{sub } t0, s0, t3
  \]

• Plan
Load (Stage 1)

IF: Instruction Fetch
- Address
- Instruction Memory
- PC
- Selector

ID: Instruction decoder
- Read register 1
- Read register 2
- Write register
- Read data 1
- Read data 2
- Write data

EX: Execute / address calculate
- Sign extended
- Shift Left
- ADD

MEM: Memory access
- Zero

WB: Write Back
- Address
- Data Memory
- Write data
Subtract (Stage 1), Load (Stage 2)
Subtract (Stage 2), Load (Stage 3)
Subtract (Stage 3), Load (Stage 4)
Data Hazard

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Selector
PC

Add

4

Address
Instruction Memory
Instruction

Sign extended

Shift Left

Zero

ALU Result

Selector

Read register 1
Read data 1

Read register 2
Read data 2

Write register

Write data

Address
Data Memory
Write data

Selector

Data
Memory
ALU
Add

Instruction

Register

Read

Write
Data Hazard

- Our example
  
  ```
  load $s0, 20($t0)
  sub $t0, $s0, $t3
  ```

- Worse than add/sub hazard
  
  - we need operand value in $s0
  
  - we have not even retrieved it at this stage

- Stalling
  
  - load instruction processing has to move to stage 5
  
  - sub instruction processing has to stall
Stalling

- Hazard condition between 2 instructions

- Second instruction has to be delayed

- Technical solution: insert a "nop" operation ("no operation")

- Resets program counter
• Fetch of load instruction
Load and Sub Processing

- Fetch of sub instruction
Load and Sub Processing

• Next stage
  - load: address calculation
  - sub: instruction decode
Load and Sub Processing

• Next stage
  - load: address calculation
  - sub: instruction decode

• Registers are known now → hazard detected
Load and Sub Processing

lw $s0,20($t0)
sub $t0,$s0,$t3

• Insertion of nop instruction
Load and Sub Processing

lw $s0,20($t0)

Next stage
- load: memory retrieve
- sub: instruction decode
Load and Sub Processing

- Next stage
  - load: write to register
  - sub: ALU operation execution
- Operand for sub forwarded from load instruction execution
Hazard Detection (Stalling) Unit

- Stalling unit must
  - detect if there is a data hazard
  - insert a "nop" instruction into pipeline

- Relevant information for decision
  - identify of input registers used in instruction currently in ID (either first or second operand)
  - identity of load register used in instruction currently in EX
  - control flag that there is indeed a memory read in EX

- Format of decision
  - overwrite instruction currently in ID with "nop"
  - reset program counter
Formal Names

• Relevant information for decision
  - **ID.Rs** and **ID.Rt**
    identify of input registers used in instruction currently in ID (either first or second operand)
  - **EX.Rd**
    identity of load register used in instruction currently in EX
  - **EX.MemRead**
    control flag that there is indeed a memory read in EX

• Format of decision
  - **ID/EX**
    overwrite instruction currently in ID with "nop"
  - **PC**
    reset program counter
Stalling Logic

- Logic in stalling unit

\[
\text{if (EX.MemRead and}
\quad (EX.Rd = ID.Rs \text{ or}
\quad \quad \quad \text{EX.Rd = ID.Rt}))
\]

\[
\text{PC = PC - 4}
\]

\[
\text{ID/EX = nop}
\]
Stalling Unit

IF: Instruction Fetch
ID: Instruction decoder register file read

Selector → PC

Address
Instruction Memory
Instruction

Hazard Detection Unit
ID.Rs
ID.Rt
EX.Rd
ID/EX
EX.MemRead
EX.Rd
ID/EX

Read register 1
Read register 2
Write register
Write data

Read data 1
Read data 2

Philipp Koehn
Computer Systems Fundamentals: Data Hazards
9 October 2019
Additional Forwarding Logic

- Additional logic in forwarding unit

```c
if (WB.Rd == EX.Rs)
    Forward.Rs = WB.RdValue
    Hazard.Rs = 1
else
    Hazard.Rs = 0

if (WB.Rd == EX.Rt)
    Forward.Rt = WB.RdValue
    Hazard.Rt = 1
else
    Hazard.Rt = 0
```

- Also relevant in "add, anything, add" sequence where result from first add is used in last add