Data Hazards

Philipp Koehn

26 October 2016
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 wiring to circuit to directly connect output of ALU for next instruction
Load and Subtract

lw $s0,20($t0)

sub $t0,$s0,$t3

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

- Example

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

- Plan
Add (Stage 1)

If: Instruction Fetch

ID: Instruction decoder
   register file read

Ex: Execute / address calculate

Mem: Memory access

Wb: Write Back

Selector
PC
Add
Address
Instruction Memory
Instruction

Selector

Add

Sign
extended

Shift
Left

Zero

ALU

Select

Address
Read data
Data Memory
Write data

Read register 1
Read data 1

Read register 2
Read data 2

Write register
Write data

Selector

Add (Stage 1)
Subtract (Stage 1), Add (Stage 2)

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

Selector -> PC

Add

4

Instruction Memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2

Registers

Address

Instruction

Select

EX: Execute /
address calculate

Shift Left

ADD

Zero

ALU

Result

Selector

Data Memory
Write data
Address

Selector

Data

Memory
Subtract (Stage 2), Add (Stage 3)
Subtract (Stage 3), Add (Stage 4)
**Data Hazard**

IF: Instruction Fetch

ID: Instruction decoder
register file read

EX: Execute /
address calculate

MEM: Memory access

WB: Write Back

Add

Sign extended

Shift Left

Zero

ALU Result

Selector

Write data

Read data 1

Read data 2

Read register 1

Read register 2

Write register

Instruction Fetch

Instruction decoder

Execute

Write Back

Selector

PC

Selector

Selector

Address

Instruction Memory

Instruction

Read 
register 1

Read 
register 2

Write 
data

Data Memory

Write 
data

Address

Data

Write 
data

Add

Instruction decoder

Execution

Write Back

Selector

PC

Selector

Address

Instruction Memory

Instruction

Read register 1

Read register 2

Write register

Write data
Forwarding Data
Forwarding Unit

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

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

- Format of decision
  - \texttt{Forward.Rs} and \texttt{Forward.Rt}
    Register value
  - \texttt{Hazard.Rs} and \texttt{Hazard.Rt}
    Control lines for selectors for input to ALU
Forwarding Logic

• Logic in forwarding unit

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
stalling
**Load and Subtract**

- **Example**

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

- **Plan**

  ![Pipeline Diagram]

  1. **IF**
   - `lw $s0, 20($t0)`
   - `bubble`

  2. **ID**
   - `bubble`

  3. **EX**
   - `bubble`
   - `sub $t0, $s0, $t3`

  4. **MEM**

  5. **WB**

Load (Stage 1)

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

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

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

MEM: Memory access
- Data Memory
  - Read data
  - Write data

WB: Write Back
- Selector
- Address
- Read data
Subtract (Stage 1), Load (Stage 2)
Subtract (Stage 2), Load (Stage 3)

IF: Instruction Fetch

ID: Instruction decoder
- register file read

EX: Execute /
- address calculate

MEM: Memory access

WB: Write Back

IF:
- Instruction Fetch
- 4
- Add
- Address
- Instruction Memory
- Instruction

ID:
- Instruction decoder
- register file read
- Read register 1
- Read register 2
- Write register
- Read data 1
- Read data 2
- Write data
- Sign extended
- Shift Left

EX:
- ADD
- Zero
- ALU Result

MEM:
- Read data
- Data Memory
- Write data

WB:
- Selector
- Address
- Data Memory
- Write data

SELECTOR

PC

SELECTOR
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

Instruction Memory
Instruction

Address

Instruction Memory

Register file read

Add

Sign extended

Shift Left

ADD

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

Selector

Zero

ALU Result

Address
data

Data Memory
Write data

Selector

Four
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
Load and Sub Processing

- Fetch of load instruction

```assembly
lw $s0,20($t0)
```
Load and Sub Processing

- Fetch of sub instruction
Load and Sub Processing

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

lw $s0, 20($t0)

sub $t0, $s0, $t3

- load: address calculation
- sub: instruction decode

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

lw $s0,20($t0)

IF
ID
EX

sub $t0,$s0,$t3

IF
• Insertion of nop instruction
- 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
  – identity 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

```plaintext
if (EX.MemRead and
    (EX.Rd = ID.Rs or
     EX.Rd = ID.Rt))
  PC = PC - 4
  ID/EX = nop
```
Additional Forwarding Logic

- Additional logic in forwarding unit

  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
### Forwarding Unit

**IF (Instruction Fetch):**
- **Selector**
- **PC**
- **Address**
- **Instruction Memory**

**ID (Instruction decoder):**
- **Instruction decoder**
- **Register file read**

**EX (Execute):**
- **Execute**
- **Address calculate**

**MEM (Memory access):**
- **Memory access**

**WB (Write Back):**
- **Write Back**

**ALU:**
- **ADD**
  - **Sign extended**
  - **Shift Left**

**Registers:**
- **Read register 1**
- **Read data 1**
- **Read register 2**
- **Read data 2**
- **Write register**
- **Write data**

**Forwarding Unit:**
- **Selector**
- **Address**
- **Data Memory**
- **Write data**

**Selector:**
- **Zero**
- **Selector**

**Selector:**
- **Address**
- **Data Memory**
- **Write data**

---

**Philipp Koehn**

**Computer Systems Foundation: Data Hazards**

26 October 2016