DYNAMIC SCHEDULING

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Homework 3 will be uploaded tonight (11:59PM)

- This lecture
  - Recap Branch Prediction
  - Dynamic scheduling
    - Forming data flow graph on the fly
  - Register renaming
    - Removing false data dependence
    - Architectural vs. physical registers
Recall: Branch Predictors

- **1-bit predictor**
- **2-bit predictor**
- **DHT: Limited PC Based**
- **BHT: Tagged**
- **BHT + DHT**
- **PC Based**
- **Local Predictor**
- **Global Predictor**
- **Tournament Predictor**
- **GHR: Correlated**
Recall: Branch Predictors

1-bit predictor

2-bit predictor

GHR: Correlated

DHT: Limited PC Based

PC Based

BHT: Tagged

Local Predictor

Global Predictor

BHT + DHT

Tournament Predictor
Branch Prediction Summary

- Dedicated predictor per branch
  - Program counter is used for assigning predictors to branches

- Capturing correlation among branches
  - Shift register is used to track history

- Predicting branch direction is not enough
  - Which instruction to be fetched if taken?

- Storing the target instruction can eliminate fetching
  - Extra hardware is required
Branch Target Buffer

- Store a target address for each branch
Branch Target Buffer

- Store tags and target addresses for each branch
Big Picture

- **Goal:** exploiting more ILP by avoiding stall cycles
  - Branch prediction can avoid the stall cycles in the frontend
Goal: exploiting more ILP by avoiding stall cycles

- Branch prediction can avoid the stall cycles in the frontend

More instructions are sent to the pipeline
Big Picture

- **Goal:** exploiting more ILP by avoiding stall cycles
  - Branch prediction can avoid the stall cycles in the frontend
    - More instructions are sent to the pipeline
  - Instruction scheduling can remove unnecessary stall cycles in the execution/memory stage
    - Static scheduling
      - Complex software (compiler)
      - Unable to resolve all data hazards (no access to runtime details)
    - Dynamic scheduling
      - Completely done in hardware
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering

Assembly code:

- `DIV F1, F2, F3`
- `ADD F4, F1, F5`
- `SUB F6, F5, F7`
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering

**Assembly code:**
- `DIV  F1, F2, F3`  
  - Long latency operation
- `ADD  F4, F1, F5`  
  - Dependent instruction
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering

Assembly code:

- `DIV F1, F2, F3` (Long latency operation)
- `ADD F4, F1, F5` (Dependent instruction)
- `SUB F6, F5, F7` (Independent instruction)

Out-of-order execution?
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering
  - Instructions are executed in data flow order

Program code

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>R1, R0, #1</td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>R2, R0, #4</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>R3, R3, R2</td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>R2, R2, #1</td>
<td></td>
</tr>
<tr>
<td>BNEQ</td>
<td>R2, R1, next</td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td>ADD R4, R4, R3</td>
<td></td>
</tr>
<tr>
<td>next:</td>
<td>BNEQ R2, R0, loop</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Scheduling

Key idea: creating an instruction schedule based on runtime information
- Hardware managed instruction reordering
- Instructions are executed in data flow order

Program code:
```
ADDI  R1, R0, #1
ADDI  R2, R0, #4
ADD   R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
ADD   R4, R4, R3
BNEQ R2, R0, loop
```
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
- Hardware managed instruction reordering
- Instructions are executed in data flow order

Program code:
```
ADDI R1, R0, #1
ADDI R2, R0, #4
ADD R3, R3, R2
ADDI R2, R2, #-1
BNEQ R2, R1, next
BNEQ R2, R0, loop
ADD R4, R4, R3
```

```
ADDI R1, R0, #1
ADDI R2, R0, #4
ADD R3, R3, R2
ADDI R2, R2, #-1
BNEQ R2, R1, next
BNEQ R2, R0, loop
ADD R4, R4, R3
```
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering
  - Instructions are executed in data flow order
Dynamic Scheduling

- **Key idea**: creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering
  - Instructions are executed in data flow order

Program code

```
ADDI  R1, R0, #1
ADDI  R2, R0, #4
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
BNEQ R2, R0, loop
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
BNEQ R2, R0, loop
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
ADD  R3, R3, R2
ADDI  R2, R2, #-1
BNEQ R2, R1, next
ADD  R3, R3, R2
ADDI  R2, R2, #-1
```

How to form data flow graph on the fly?
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations

DIV F1, F2, F3
ADD F4, F1, F5
SUB F5, F6, F7
ADD F4, F5, F8
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations

```
DIV    F1, F2, F3
ADD    F4, F1, F5
SUB    F5, F6, F7
ADD    F4, F5, F8
```

![Diagram showing the flow of operations through different stages of a pipeline including IF, ID, Queue, Ex, Mem, ROB, M1-M7, Div, Add, and Multiply units.](image-url)
Register Renaming

- **Eliminating WAR and WAW hazards**
  - Change the mapping between architectural registers and physical storage locations

![Diagram showing architectural units and WAR/WAW hazards]

- **RAW**
  - DIV: F1, F2, F3
  - ADD: F4, F1, F5
  - SUB: F5, F6, F7
  - ADD: F4, F5, F8

![Diagram of processor pipeline stages: IF, ID, Queue, Ex, Mem, Reorder Buffer (ROB), WB]

- **Integer unit**
- **FP/integer multiply**
- **FP adder**
- **FP/integer divider**
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations

![Diagram showing architectural registers and their mapping to physical storage locations.](image-url)
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations

WAR and WAW hazards can be removed using more registers
Register Renaming

- Eliminating WAR and WAW hazards
  1. allocate a free physical location for the new register
  2. find the most recently allocated location for the register

DIV   F1, F2, F3
ADD   F4, F1, F5
SUB   F5, F6, F7
ADD   F4, F5, F8

Architectural Registers
- F1
- F2
- F3
- F4
- F5
- F6
- F7
- F8

Physical Locations
- P10
- P11
- P12
- P13
- P14
- P15
- P16
- P17
- P18
- P19
Register Renaming

- Eliminating WAR and WAW hazards
  1. allocate a free physical location for the new register
  2. find the most recently allocated location for the register

```
DIV    F1, F2, F3
ADD    F4, F1, F5
SUB     F5, F6, F7
ADD    F4, F5, F8
```

DIV    P12, P11, P10
Register Renaming

- Eliminating WAR and WAW hazards
  - 1. allocate a free physical location for the new register
  - 2. find the most recently allocated location for the register

### Architectural Registers
- DIV F1, F2, F3
- ADD F4, F1, F5
- SUB F5, F6, F7
- ADD F4, F5, F8

### Physical Locations
- DIV P12, P11, P10
- ADD P14, P12, P15
Register Renaming

- Eliminating WAR and WAW hazards
  1. allocate a free physical location for the new register
  2. find the most recently allocated location for the register

```
DIV    F1, F2, F3
ADD    F4, F1, F5
SUB     F5, F6, F7
ADD    F4, F5, F8
```

```
DIV    P12, P11, P10
ADD    P14, P12, P15
SUB     P19, P17, P13
```
Register Renaming

- Eliminating WAR and WAW hazards
  - 1. allocate a free physical location for the new register
  - 2. find the most recently allocated location for the register

DIV  F1, F2, F3
ADD  F4, F1, F5
SUB  F5, F6, F7
ADD  F4, F5, F8

DIV  P12, P11, P10
ADD  P14, P12, P15
SUB  P19, P17, P13
ADD  P18, P19, P16
The main idea is to issue dynamic instructions out of program order while maintaining data flow.
The main idea is to issue dynamic instructions out of program order while maintaining data flow.
The main idea is to issue dynamic instructions out of program order while maintaining data flow.

Program Data Flow:
- ADDI R1, R0, #1
- ADDI R2, R0, #4
- SUBI R5, R2, 1
- DIV R3, R3, R2
- SUBI R6, R5, 1
- DIV R7, R3, R2
- MUL R4, R4, R3

Functional Units:
- Adder
- Divider
- Multiplier
Dynamic Scheduling

- The main idea is to issue dynamic instructions out of program order while maintaining data flow

Program Data Flow

- ADDI R1, R0, #1
- ADDI R2, R0, #4
- SUBI R5, R2, 1
- DIV R3, R3, R2
- SUBI R6, R5, 1
- DIV R7, R3, R2
- MUL R4, R4, R3

Functional Units

- Adder
- Divider
- Multiplier
- Decoded Queue
- Reservation Stations