DYNAMIC SCHEDULING

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

- Announcement
  - Homework 2 will be uploaded tonight

- This lecture
  - Dynamic scheduling
    - Forming data flow graph on the fly
  - Register renaming
    - Removing false data dependence
    - Architectural vs. physical registers
Goal: exploiting more ILP by avoiding stall cycles

Branch prediction can avoid the stall cycles in the frontend
**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
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
- ADD F4, F1, F5
- SUB F6, F5, F7

Long latency operation:
- DIV F1, F2, F3

Dependent instruction:
- 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
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

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

**Data flow**

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

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
Register Renaming

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

![Diagram showing architectural registers and physical storage locations with WAR and RAW hazards highlighted.]

- 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
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
```

```
DIV Q1, F6, F7
ADD Q2, Q1, F8
```

```
DIV    F1, F2, F3
ADD  F4, F1, F5
SUB   F5, F6, F7
ADD  F4, F5, F8
DIV    F1, F2, F3
ADD  F4, F1, F5
SUB Q1, F6, F7
ADD Q2, Q1, F8
```
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Architectural Registers</th>
<th>Physical Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td>F1, F2, F3</td>
<td>P10</td>
</tr>
<tr>
<td>ADD</td>
<td>F4, F1, F5</td>
<td>P11</td>
</tr>
<tr>
<td>SUB</td>
<td>F5, F6, F7</td>
<td>P12</td>
</tr>
<tr>
<td>ADD</td>
<td>F4, F5, F8</td>
<td>P13</td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>P14</td>
</tr>
<tr>
<td></td>
<td>F2</td>
<td>P15</td>
</tr>
<tr>
<td></td>
<td>F3</td>
<td>P16</td>
</tr>
<tr>
<td></td>
<td>F4</td>
<td>P17</td>
</tr>
<tr>
<td></td>
<td>F5</td>
<td>P18</td>
</tr>
<tr>
<td></td>
<td>F6</td>
<td>P19</td>
</tr>
<tr>
<td></td>
<td>F7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F8</td>
<td></td>
</tr>
</tbody>
</table>
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

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

DIV P12, P11, P10
ADD P14, P12, P15
```

Architectural Registers

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
</table>

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
```

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

Physical Locations
```
P10
P11
P12
P13
P14
P15
P16
P17
P18
P19
```

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