The Pipeline Design of Modern Processors

Hung-Wei Tseng
Recap: Pipelining
Recap: Pipelining

After this point, we are completing an instruction each cycle!
Local predictor — every branch instruction has its own state
2-bit — each state is described using 2 bits
Change the state based on actual outcome
If we guess right — no penalty
If we guess wrong — flush (clear pipeline registers) for mis-predicted instructions that are currently in IF and ID stages and reset the PC

Recap: 2-bit/Bimodal local predictor

<table>
<thead>
<tr>
<th>Predict Taken</th>
<th>branch PC</th>
<th>target PC</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x400048</td>
<td>0x400032</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0x400080</td>
<td>0x400068</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0x401080</td>
<td>0x401100</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>0x4000F8</td>
<td>0x400100</td>
<td>01</td>
</tr>
</tbody>
</table>

Predict Taken

Strong Not Taken 00 (0)
Strong Taken 11 (3)
Weak Taken 10 (2)
Weak Not Taken 01 (1)
Recap: Global history (GH) predictor

States associated with history

Global History Register

Predict Taken

Branch Target Buffer

branch PC | target PC
---|---
0x400048 | 0x400032
0x400080 | 0x400068
0x401080 | 0x401100
0x4000F8 | 0x400100
Outline

• Super Scalar
• Out-of-order execution
Do we still have to stall?

• How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline with “full” data forwarding?

```
lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10, 8
bne  $10,$5, LOOP
```

A. 0
B. 1
C. 2
D. 3
E. 4
The effect of code optimization

By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

A. (1) & (2)
B. (2) & (3)
C. (3) & (4)
D. (4) & (5)
E. None of the pairs can be reordered
If we can predict the future ...

- Consider the following dynamic instructions:
  1. `lw    $6,0($10)`
  2. `add   $7, $6,$12`
  3. `sw    $7,0($10)`
  4. `addi  $10,$10, 8`
  5. `bne   $10, $5, LOOP`
  6. `lw    $6,0($10)`
  7. `add   $7, $6,$12`
  8. `sw    $7,0($10)`
  9. `addi  $10,$10, 8`
 10. `bne   $10, $5, LOOP`

Which of the following pair can we reorder without affecting the correctness if the branch prediction is perfect?

A. (2) and (4)
B. (3) and (5)
C. (5) and (6)
D. (6) and (9)
E. (9) and (10)
Consider the following dynamic instructions:

- lw $6,0($10)
- add $7, $6,$12
- sw $7,0($10)
- addi $10,$10, 8
- bne $10, $5, LOOP
- lw $6,0($10)
- add $7, $6,$12
- sw $7,0($10)
- addi $10,$10, 8
- bne $10, $5, LOOP

Which of the following pair can we reorder without affecting the correctness if the branch prediction is perfect?

A. (2) and (4)
B. (3) and (5)
C. (5) and (6)
D. (6) and (9)
E. (9) and (10)
If we can predict the future ...

- Consider the following dynamic instructions:
  ① lw $6,0($10)
  ② add $7, $6,$12
  ③ sw $7,0($10)
  ④ addi $10,$10, 8
  ⑤ bne $10, $5, LOOP
  ⑥ lw $6,0($10)
  ⑦ add $7, $6,$12
  ⑧ sw $7,0($10)
  ⑨ addi $10,$10, 8
  ⑩ bne $10, $5, LOOP

Can we use “branch prediction” to predict the future and reorder instructions across the branch?

Which of the following pair can we reorder without affecting the correctness if the branch prediction is perfect?

A. (2) and (4)
B. (3) and (5)
C. (5) and (6)
D. (6) and (9)
E. (9) and (10)
Dynamic instruction scheduling/Out-of-order (OoO) execution
Tips of drawing a pipeline diagram

• Each instruction has to go through all 5 pipeline stages: IF, ID, EXE, MEM, WB in order — only valid if it’s single-issue, MIPS 5-stage pipeline

• An instruction can enter the next pipeline stage in the next cycle if
  • No other instruction is occupying the next stage
  • This instruction has completed its own work in the current stage
  • The next stage has all its inputs ready

• Fetch a new instruction only if
  • We know the next PC to fetch
  • We can predict the next PC
  • Flush an instruction if the branch resolution says it’s mis-predicted.
What do you need to execute an instruction?

- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — all data dependencies are resolved
- Whenever the target functional unit is available
- This instruction has completed its own work in the current stage
- No other instruction is occupying the next stage
- The next stage has all its inputs ready
Scheduling instructions: based on data dependencies

- Draw the data dependency graph, put an arrow if an instruction depends on the other.
  1. `lw $6,0($10)`
  2. `add $7, $6,$12`
  3. `sw $7,0($10)`
  4. `addi $10,$10, 8`
  5. `bne $10, $5, LOOP`
  6. `lw $6,0($10)`
  7. `add $7, $6,$12`
  8. `sw $7,0($10)`
  9. `addi $10,$10, 8`
  10. `bne $10, $5, LOOP`

- **In theory**, instructions without dependencies can be executed in parallel or out-of-order.
- Instructions with dependencies can never be reordered.
If we can predict the future ...

- Consider the following dynamic instructions:
  1. lw $6, 0($10)
  2. add $7, $6, $12
  3. sw $7, 0($10)
  4. addi $10, $10, 8
  5. bne $10, $5, LOOP
  6. lw $6, 0($10)
  7. add $7, $6, $12
  8. sw $7, 0($10)
  9. addi $10, $10, 8
  10. bne $10, $5, LOOP

Which of the following pair can we reorder without affecting the correctness if the branch prediction is perfect?

A. (2) and (4)
B. (3) and (5)
C. (5) and (6)
D. (6) and (9)
E. (9) and (10)

We still can only reorder (5) and (6) even though (2) & (4) are not depending on each other!
False dependencies

- We are still limited by **false dependencies**
- They are not “true” dependencies because they don’t have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 4 and 1
    - 4 and 3, 6 and 2, 7 and 3, 9 and 5, 9 and 6, 9 and 8
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1, 7 and 2
• Consider the following dynamic instructions
  ① lw $12, 0($20)
  ② add $12, $10, $12
  ③ sub $18, $12, $10
  ④ lw $12, 8($20)
  ⑤ add $14, $18, $12
  ⑥ add $18, $14, $14
  ⑦ sw $14, 16($20)
  ⑧ addi $20, $20, 8

which of the following pair is not a “false dependency”

A. (1) and (4)
B. (1) and (8)
C. (5) and (7)
D. (4) and (8)
E. (7) and (8)
• Consider the following dynamic instructions

1. `lw $12, 0($20)`
2. `add $12, $10, $12`
3. `sub $18, $12, $10`
4. `lw $12, 8($20)`
5. `add $14, $18, $12`
6. `add $18, $14, $14`
7. `sw $14, 16($20)`
8. `addi $20, $20, 8`

which of the following pair is not a "false dependency"?

A. (1) and (4)
B. (1) and (8)
C. (5) and (7)
D. (4) and (8)
E. (7) and (8)
False dependencies

- Consider the following dynamic instructions
  ① lw $12, 0($20)
  ② add $12, $10, $12
  ③ sub $18, $12, $10
  ④ lw $12, 8($20)
  ⑤ add $14, $18, $12
  ⑥ add $18, $14, $14
  ⑦ sw $14, 16($20)
  ⑧ addi $20, $20, 8

which of the following pair is not a “false dependency”

A. (1) and (4) WAW
B. (1) and (8) WAR
C. (5) and (7) True dependency (RAW)
D. (4) and (8) WAR
E. (7) and (8) WAR
Out-of-order execution

• Any sequence of instructions has set of RAW, WAW, and WAR hazards that constrain its execution.
• Can we design a processor that extracts as much parallelism as possible, while still respecting these dependences?
Register renaming
False dependencies

- They are not “true” dependencies because they don’t have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 4 and 1 4 and 3, 6 and 2, 7 and 3, 9 and 5, 9 and 6, 9 and 8
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1, 7 and 2

- False dependencies coming from the sharing/competition of registers
  1. lw $6,0($10)
  2. add $7, $6,$12
  3. sw $7,0($10)
  4. addi $10,$10, 8
  5. bne $10, $5, LOOP
  6. lw $6,0($10)
  7. add $7, $6,$12
  8. sw $7,0($10)
  9. addi $10,$10, 8
  10. bne $10, $5, LOOP

  1. lw $20,0($10)
  2. add $21, $6,$12
  3. sw $20,0($10)
  4. addi $22,$10, 8
  5. bne $22, $5, LOOP
Register renaming

• Provide a set of “physical registers” and a mapping table mapping “architectural registers” to “physical registers”
• Allocate a physical register for a new output
• Stages
  • Dispatch (D) — allocate a “physical” for the output of a decoded instruction
  • Issue (I) — collect pending values/branch outcome from common data bus
  • Execute (INT, AQ/AQ/MEM, M1/M2/M3, BR) — send the instruction to its corresponding pipeline if no structural hazards
  • Write Back (WB) — broadcast the result through CDB
Overview of a processor supporting register renaming

Fetch/decode instruction

Renaming logic

Instruction Queue

Unresolved Branch

Register mapping table

Physical Registers

Address Resolution

Integer ALU

Floating-Point Adder

Floating-Point Mul/Div

Branch

Load Queue

Store Queue

Memory

Physical Registers

valid value

P1

P2

P3

P4

P5

P6

...

...

valid value

physical register #

X1 register #

X2 register #

X3 register #

...

...

Address

Dest

Reg.

Value

Addr.

Addr.

Add.

Data

Dest

Reg.
Register renaming in motion

1. lw $6,0($10)  
2. add $7,$6,$12  
3. sw $7,0($10)  
4. addi $10,$10,8  
5. bne $10,$5,LOOP  
6. lw $6,0($10)  
7. addi $10,$10,8  
8. bne $10,$5,LOOP  
9. lw $6,0($10)  
10. addi $10,$10,8

### Renamed instruction

<table>
<thead>
<tr>
<th></th>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lw P1, 0($10)</td>
<td>$5</td>
<td>0</td>
<td>1</td>
<td>P6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>$7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>$12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>$10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>$12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>$10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>$12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>$10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw $6,0($10)
2. add $7,$6,$12
3. sw $7,0($10)
4. addi $10,$10,8
5. bne $10,$5,LOOP
6. lw $6,0($10)
7. add $7,$6,$12
8. sw $7,0($10)
9. addi $10,$10,8
10. bne $10,$5,LOOP

Renamed instruction
1. lw P1, 0($10)
2. add P2, P1, $12

Physical Register

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
<td>1</td>
<td></td>
<td>P6</td>
</tr>
<tr>
<td>add P2, P1, $12</td>
<td>$6, P1</td>
<td>1</td>
<td></td>
<td>P7</td>
</tr>
<tr>
<td></td>
<td>$7, P2</td>
<td></td>
<td></td>
<td>P8</td>
</tr>
<tr>
<td></td>
<td>$10</td>
<td></td>
<td></td>
<td>P9</td>
</tr>
<tr>
<td></td>
<td>$12</td>
<td></td>
<td></td>
<td>P10</td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw $6, 0($10)
2. add $7, $6, $12
3. sw $7, 0($10)
4. addi $10, $10, 8
5. bne $10, $5, LOOP
6. lw $6, 0($10)
7. add $7, $6, $12
8. sw $7, 0($10)
9. addi $10, $10, 8
10. bne $10, $5, LOOP

Renamed instruction

Physical Register

Valid | Value | In use
P1 | 0 | 1
P2 | 0 | 1
P3 | | P8
P4 | | P9
P5 | | P10

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1       lw P1, 0($10)   $5</td>
<td></td>
</tr>
<tr>
<td>2      add P2, P1, $12   $6</td>
<td></td>
</tr>
<tr>
<td>3       sw P2, 0($10)   $7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$10</td>
</tr>
<tr>
<td>5</td>
<td>$12</td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw   P1, 0($10)  
2. add  P2, P1, $12  
3. sw   P2, 0($10)   
4. addi P3, $10, 8   
5. bne  $10, $5, LOOP 
6. lw   $6,0($10)    
7. add  $7,$6,$12    
8. sw   $7,0($10)    
9. addi $10,$10,8   
10. bne $10,$5,LOOP  

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 lw P1, 0($10)</td>
<td>$5 P1</td>
</tr>
<tr>
<td>2 add P2, P1, $12</td>
<td>$6 P1</td>
</tr>
<tr>
<td>3 sw P2, 0($10)</td>
<td>$7 P2</td>
</tr>
<tr>
<td>4 addi P3, $10, 8</td>
<td>$10 P3</td>
</tr>
<tr>
<td></td>
<td>$12 P3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw $6,0($10)
2. add $7, $6, $12
3. sw $7, 0($10)
4. addi $10, $10, 8
5. bne $10, $5, LOOP
6. lw $6, 0($10)
7. add $7, $6, $12
8. sw $7, 0($10)
9. addi $10, $10, 8
10. bne $10, $5, LOOP

Renamed instruction:
1. lw P1, 0($10)
2. add P2, P1, $12
3. sw P2, 0($10)
4. addi P3, $10, 8
5. bne P3, $5, LOOP

Physical Register:
- R: Register
- I: Immediate
- AR: Address Register
- LSQ: Load Store Queue
- MEM: Memory

Valid | Value | In use
--- | --- | ---
P1 | 0 | 1
P2 | 0 | 1
P3 | 0 | 1
P4 | | |
P5 | | |
P6 | | |
P7 | | |
P8 | | |
P9 | | |
P10 | | |
Register renaming in motion

lw    $6,0($10)
add   $7,$6,$12
sw    $7,0($10)
addi  $10,$10,8
bne   $10,$5,LOOP
lw    $6,0($10)
add   $7,$6,$12
sw    $7,0($10)
addi  $10,$10,8
bne   $10,$5,LOOP

Renamed instruction
1 lw    P1, 0($10)
2 add   P2, P1, $12
3 sw    P2, 0($10)
4 addi  P3, $10, 8
5 bne   P3, $5, LOOP
6 lw    P4, 0(P3)

Physical Register
$5
$6  P1
$7  P2
$10 P3
$12

Valid  Value  In use
P1  1      1  P6
P2  0      1  P7
P3  0      1  P8
P4  0      1  P9
P5  0      1  P10
Register renaming in motion

Renamed instruction
1 lw P1, 0($10)  
2 add P2, P1, $12  
3 sw P2, 0($10)  
4 addi P3, $10, 8  
5 bne P3, $5, LOOP  
6 lw P4, 0(P3)  
7 add P5, P1, $12  
8  
9  
10 bne $10, $5, LOOP

Physical Register
<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>I</th>
<th>AR</th>
<th>LSQ</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>INT</td>
</tr>
<tr>
<td>2</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>INT</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>I</td>
<td>INT</td>
<td>WB</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>

Valid Value In use
<table>
<thead>
<tr>
<th></th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>P6</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td>P5</td>
<td>0</td>
<td>1</td>
<td>P10</td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw  $6,0($10)
2. add $7, $6, $12
3. sw  $7, 0($10)
4. addi $10, $10, 8
5. bne $10, $5, LOOP
6. lw  $6, 0($10)
7. add $7, $6, $12
8. sw  $7, 0($10)
9. addi $10, $10, 8
10. bne $10, $5, LOOP

Renamed instruction
1. lw  P1, 0($10)
2. add P2, P1, $12
3. sw  P2, 0($10)
4. addi P3, $10, 8
5. bne P3, $5, LOOP
6. lw  P4, 0(P3)
7. add P5, P1, $12
8. sw  P5, 0(P3)

Physical Register
- $5
- $6  P1
- $7  P5
- $10 P3
- $12

Valid  Value  In use
P1  1    1    P6
P2  1    1    P7
P3  1    1    P8
P4  0    1    P9
P5  0    1    P10
Register renaming in motion

1. lw $6,0($10)
2. add $7,$6,$12
3. sw $7,0($10)
4. addi $10,$10,8
5. bne $10,$5,LOOP
6. lw $6,0($10)
7. add $7,$6,$12
8. sw $7,0($10)
9. addi $10,$10,8
10. bne $10,$5,LOOP

Renamed instruction:
1. lw P1, 0($10)
2. add P2, P1, $12
3. sw P2, 0($10)
4. addi P3, $10, 8
5. bne P3, $5, LOOP
6. lw P4, 0(P3)
7. add P5, P1, $12
8. sw P5, 0(P3)
9. addi P6, P3, 8
10. bne $10,$5,LOOP

Physical Register:
- $5
  - Value: 1
  - In use: 1
- $6
  - Value: P1
- $7
  - Value: P5
- $10
  - Value: P3
- $12

Valid | Value | In use | Valid | Value | In use
---|---|---|---|---|---
P1 | 1 | 1 | P6 | 0 | 1
P2 | 1 | 1 | P7 |
P3 | 1 | 1 | P8 |
P4 | 0 | 1 | P9 |
P5 | 0 | 1 | P10 |
Register renaming in motion

1. `lw  $6,0($10)`
2. `add $7,$6,$12`
3. `sw  $7,0($10)`
4. `addi $10,$10,8`
5. `bne $10,$5,LOOP`
6. `lw  $6,0($10)`
7. `add $7,$6,$12`
8. `sw  $7,0($10)`
9. `addi $10,$10,8`
10. `bne $10,$5,LOOP`

Renamed instruction

1. `lw    P1, 0($10)`
2. `add   P2, P1, $12`
3. `sw    P2, 0($10)`
4. `addi  P3, P10, 8`
5. `bne   P9, 0($10)`
6. `lw    P4, 0($10)`
7. `add   P5, P1, $12`
8. `sw    P5, 0($10)`
9. `addi  P6, 0($10)`
10. `bne   P6, 0($10)`

Physical Register

<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td>1</td>
<td>1</td>
<td>P6</td>
</tr>
<tr>
<td>$6</td>
<td>1</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td>$7</td>
<td>1</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td>$10</td>
<td>0</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td>$12</td>
<td>0</td>
<td>1</td>
<td>P10</td>
</tr>
</tbody>
</table>

Valid | Value | In use
---|-------|--------|
P1     1     1
P2     1     1
P3     1     1
P4     0     1
P5     0     1

Table:

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lw    P1, 0($10)</code></td>
<td>$5</td>
<td>1</td>
<td>1</td>
<td>P6</td>
</tr>
<tr>
<td><code>add   P2, P1, $12</code></td>
<td>$6</td>
<td>1</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td><code>sw    P2, 0($10)</code></td>
<td>$7</td>
<td>1</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td><code>addi  P3, P10, 8</code></td>
<td>$10</td>
<td>0</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td><code>bne   P9, 0($10)</code></td>
<td>$12</td>
<td>0</td>
<td>1</td>
<td>P10</td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw $6, 0($10)
2. add $7, $6, $12
3. sw $7, 0($10)
4. addi $10, $10, 8
5. bne $10, $5, LOOP
6. lw $6, 0($10)
7. add $7, $6, $12
8. sw $7, 0($10)
9. addi $10, $10, 8
10. bne $10, $5, LOOP

Renamed instruction
1. lw P1, 0($10)
2. add P2, P1, $12
3. sw P2, 0($10)
4. addi P3, $10, 8
5. bne P9, 0, LOOP
6. lw P4, 0($3)
7. add P5, P1, $12
8. sw P5, 0($3)
9. addi P6, P3, 8
10. bne P6, 0($10)

Physical Register

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
<td>1</td>
<td>1</td>
<td>P6</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>add P2, P1, $12</td>
<td>$6</td>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw P2, 0($10)</td>
<td>$7</td>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi P3, $10, 8</td>
<td>$10</td>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne P9, 0, LOOP</td>
<td>$12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

- lw $6,0($10)
- add $7,$6,$12
- sw $7,0($10)
- addi $10,$10,8
- bne $10,$5,LOOP
- lw $6,0($10)
- add $7,$6,$12
- sw $7,0($10)
- addi $10,$10,8
- bne $10,$5,LOOP

### Renamed instruction

1. lw P1, 0($10)
2. add P2, P1, $12
3. sw P2, 0($10)
4. addi P3, $10, 8
5. bne P9, $5, LOOP
6. lw P4, 0(P3)
7. add P5, P1, $12
8. sw P5, 0(P3)
9. addi P6, P3, 8
10. bne P6, 0($10)

### Physical Register

<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td></td>
<td></td>
<td></td>
<td>P1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$6</td>
<td>P1</td>
<td></td>
<td></td>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$7</td>
<td>P5</td>
<td></td>
<td></td>
<td>P3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$10</td>
<td>P3</td>
<td></td>
<td></td>
<td>P4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$12</td>
<td></td>
<td></td>
<td></td>
<td>P5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Register renaming in motion

```
lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP
lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP
```

**Physical Register**

<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>P6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$6</td>
<td>P1</td>
<td>1</td>
<td></td>
<td>P7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td>P5</td>
<td>1</td>
<td></td>
<td>P8</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$10</td>
<td>P5</td>
<td>1</td>
<td></td>
<td>P9</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$12</td>
<td>P3</td>
<td>1</td>
<td></td>
<td>P10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Renamed instruction**

1. lw P1, 0($10)
2. add P2, P1, $12
3. sw P2, 0($10)
4. addi P3, $10, 8
5. bne P9, $5, LOOP
6. lw P4, 0(P3)
7. add P5, P1, $12
8. sw P5, 0(P3)
9. addi P6, P3, 0
10. bne P6, 0($10)
Register renaming in motion

lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP
lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP
Register renaming in motion

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 lw P1, 0($10)</td>
<td>$5</td>
<td>1</td>
<td>1</td>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>P6</td>
</tr>
<tr>
<td>2 add P2, P1, $12</td>
<td>$6</td>
<td>1</td>
<td>1</td>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td>3 sw P2, 0($10)</td>
<td>$7</td>
<td>1</td>
<td>1</td>
<td>P3</td>
<td>1</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td>4 addi P3, $10, 8</td>
<td>$10</td>
<td>1</td>
<td>1</td>
<td>P4</td>
<td>1</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td>5 bne P4, $5, LOOP</td>
<td>$12</td>
<td>1</td>
<td>1</td>
<td>P5</td>
<td>1</td>
<td>1</td>
<td>P10</td>
</tr>
</tbody>
</table>
Register renaming in motion

- lw $6,0($10)
- add $7,$6,$12
- sw $7,0($10)
- addi $10,$10,8
- bne $10,$5,LOOP
- lw $6,0($10)
- add $7,$6,$12
- sw $7,0($10)
- addi $10,$10,8
- bne $10,$5,LOOP

Takes 12 cycles to issue all instructions
Through data flow graph analysis

INT — 2 cycles for depending instruction to start
MEM — 4 cycles for the depending instruction to start
MUL/DIV — 4 cycles for the depending instruction to start
BR — 2 cycles to resolve
What about “linked list”

• For the following C code and it’s translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )

A. 9  
B. 10  
C. 11  
D. 12  
E. 13
What about “linked list”

• For the following C code and its translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )

A. 9
B. 10
C. 11
D. 12
E. 13
What about “linked list”

• For the following C code and its translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor only fetches 1 instruction per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```c
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )
```

```mips
LOOP: lw $10, 8($10)
addi $7, $7, 1
bne $10, $0, LOOP
```

A. 9  
B. 10  
C. 11  
D. 12  
E. 13
What about “linked list”

Static instructions

```
LOOP: lw   $10, 8($10)
      addi  $7, $7, 1
      bne  $10, $0, LOOP
```

Dynamic instructions

```
① lw   $10, 8($10)
② addi  $7, $7, 1
③ bne  $10, $0, LOOP
④ lw   $10, 8($10)
⑤ addi  $7, $7, 1
⑥ bne  $10, $0, LOOP
⑦ lw   $10, 8($10)
⑧ addi  $7, $7, 1
⑨ bne  $10, $0, LOOP
```
Register renaming in motion

lw $10, 8($10)
addi $7, $7, 1
bne $10, $0, LOOP
lw $10, 8($10)
addi $7, $7, 1
bne $10, $0, LOOP
lw $10, 8($10)
addi $7, $7, 1
bne $10, $0, LOOP

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
</tr>
<tr>
<td></td>
<td>$6</td>
</tr>
<tr>
<td></td>
<td>$7</td>
</tr>
<tr>
<td></td>
<td>$10</td>
</tr>
<tr>
<td></td>
<td>$12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

1. `lw   $10, 8($10)`  
2. `addi $7, $7, 1`  
3. `bne $10, $0, LOOP`  
4. `lw   $10, 8($10)`  
5. `addi $7, $7, 1`  
6. `bne $10, $0, LOOP`  
7. `lw   $10, 8($10)`  
8. `addi $7, $7, 1`  
9. `bne $10, $0, LOOP`  
10. `lw   $10, 8($10)`

### Renamed instruction

1. `lw P1, 0($10)`  
2. `add P2, $7, 1`  
3. `...`  
4. `...`  
5. `...`  
6. `...`  
7. `...`  
8. `...`  
9. `...`  
10. `...`

### Physical Register

<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td>P1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td>P2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td>P1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$10</td>
<td>P2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Valid | Value | In use

| P1    | 0     | 1     |
| P2    | 0     | 1     |
| P3    |       |       |
| P4    |       |       |
| P5    |       |       |
| P6    |       |       |
| P7    |       |       |
| P8    |       |       |
| P9    |       |       |
| P10   |       |       |
Register renaming in motion

lw $10, 8($10)  
addi $7, $7, 1  
bne $10, $0, LOOP  
lw $10, 8($10)  
addi $7, $7, 1  
bne $10, $0, LOOP  
lw $10, 8($10)  
addi $7, $7, 1  
bne $10, $0, LOOP

Renamed instruction
1 lw P1, 0($10)  
2 add P2, $7, 1  
3 bne P1, $0, LOOP  
4  
5  
6  
7  
8  
9  
10

Physical Register

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5 P5</td>
</tr>
<tr>
<td>add P2, $7, 1</td>
<td>$6 P6</td>
</tr>
<tr>
<td>bne P1, $0, LOOP</td>
<td>$7 P2</td>
</tr>
<tr>
<td></td>
<td>$10 P1</td>
</tr>
<tr>
<td></td>
<td>$12 P10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

- lw    $10, 8($10)
- addi  $7, $7, 1
- bne   $10, $0, LOOP
- lw    $10, 8($10)
- addi  $7, $7, 1
- bne   $10, $0, LOOP
- lw    $10, 8($10)
- addi  $7, $7, 1
- bne   $10, $0, LOOP

Renamed instruction
1 lw    P1, 0($10)
2 add   P2, $7, 1
3 bne   P1, $0, LOOP
4 lw    P3, 0(P1)
5
6
7
8
9
10

Physical Register
$5
$6
$7  P2
$10 P3
$12

Valid  Value  In use
P1  0  1  P6
P2  0  1  P7
P3  0  1  P8
P4
P5
P6
P7
P8
P9
P10
Register renaming in motion

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
</tr>
<tr>
<td>add P2, $7, 1</td>
<td>$6</td>
</tr>
<tr>
<td>bne P1, $0, LOOP</td>
<td>$7</td>
</tr>
<tr>
<td>lw P3, 0(P1)</td>
<td>$10</td>
</tr>
<tr>
<td>add P4, P2, 1</td>
<td>$12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

- lw $10, 8($10)
- addi $7, $7, 1
- bne $10, $0, LOOP
- lw $10, 8($10)
- addi $7, $7, 1
- bne $10, $0, LOOP
- lw $10, 8($10)
- addi $7, $7, 1
- bne $10, $0, LOOP

Renamed instruction
1. lw P1, 0($10)
2. add P2, $7, 1
3. bne P1, $0, LOOP
4. lw P3, 0(P1)
5. add P4, P2, 1
6. bne P3, $0, LOOP

Physical Register
- $5
- $6
- $7
- $10
- $12

Valid | Value | In use
--- | --- | ---
P1 | 1 | 1
P2 | 1 | 1
P3 | 0 | 1
P4 | 0 | 1
P5 | 1 | 1
P6 |  |
P7 |  |
P8 |  |
P9 |  |
P10 |  |
Register renaming in motion

1. lw $10, 8($10)
2. addi $7, $7, 1
3. bne $10, $0, LOOP
4. lw $10, 8($10)
5. addi $7, $7, 1
6. bne $10, $0, LOOP
7. lw $10, 8($10)
8. addi $7, $7, 1
9. bne $10, $0, LOOP

Renamed instruction

1. lw P1, 0($10)
2. add P2, $7, 1
3. bne P1, $0, LOOP
4. lw P3, 0(P1)
5. add P4, P2, 1
6. bne P3, $0, LOOP
7. lw P5, 0(P3)

Physical Register

- $5
- $6
- $7
- $10
- $12

Valid  Value  In use
P1  1       1   P6
P2  1       1   P7
P3  0       1   P8
P4  0       1   P9
P5  0       1   P10
Register renaming in motion

1. lw  $10, 8($10)
2. addi $7, $7, 1
3. bne $10, $0, LOOP
4. lw  $10, 8($10)
5. addi $7, $7, 1
6. bne $10, $0, LOOP
7. lw  $10, 8($10)
8. addi $7, $7, 1
9. bne $10, $0, LOOP
10. lw $10, 8($10)

Renamed instruction

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lw</td>
<td>P1, 0($10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>add</td>
<td>P2, $7, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>bne</td>
<td>P1, $0, LOOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>lw</td>
<td>P3, 0(P1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>add</td>
<td>P4, P2, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>bne</td>
<td>P3, $0, LOOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lw</td>
<td>P5, 0(P3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>add</td>
<td>P6, P4, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Register renaming in motion

1. lw   $10, 8($10)
2. addi $7, $7, 1
3. bne  $10, $0, LOOP
4. lw   $10, 8($10)
5. addi $7, $7, 1
6. bne  $10, $0, LOOP
7. lw   $10, 8($10)
8. addi $7, $7, 1
9. bne  $10, $0, LOOP

Renamed instruction

1. lw    P1, 0($10)
2. add   P2, $7, 1
3. bne   P1, $0, LOOP
4. lw    P3, 0(P1)
5. add   P4, P2, 1
6. bne   P3, $0, LOOP
7. lw    P5, 0(P3)
8. add   P6, P4, 1
9. bne   P5, $0, LOOP

Physical Register

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 lw P1, 0($10)</td>
<td>$5</td>
</tr>
<tr>
<td>2 add P2, $7, 1</td>
<td>$6</td>
</tr>
<tr>
<td>3 bne P1, $0, LOOP</td>
<td>$7</td>
</tr>
<tr>
<td>4 lw P3, 0(P1)</td>
<td>$10 P6 P5</td>
</tr>
<tr>
<td>5 add P4, P2, 1</td>
<td>$12</td>
</tr>
<tr>
<td>6 bne P3, $0, LOOP</td>
<td></td>
</tr>
<tr>
<td>7 lw P5, 0(P3)</td>
<td></td>
</tr>
<tr>
<td>8 add P6, P4, 1</td>
<td></td>
</tr>
<tr>
<td>9 bne P5, $0, LOOP</td>
<td></td>
</tr>
</tbody>
</table>

Valid | Value | In use | Valid | Value | In use
P1    1     1     P6    0     1
P2    1     1     P7    0     1
P3    0     1     P8    0     1
P4    0     1     P9    0     1
P5    0     1     P10  0     1
Register renaming in motion

lw   $10, 8($10)
addi $7, $7, 1
bne  $10, $0, LOOP
lw   $10, 8($10)
addi $7, $7, 1
bne  $10, $0, LOOP
lw   $10, 8($10)
addi $7, $7, 1
bne  $10, $0, LOOP

Renamed instruction

1  lw    P1, 0($10)
2  add   P2, $7, 1
3  bne   P1, $0, LOOP
4  lw    P3, 0(P1)
5  add   P4, P2, 1
6  bne   P3, $0, LOOP
7  lw    P5, 0(P3)
8  add   P6, P4, 1
9  bne   P5, $0, LOOP

Physical Register

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 lw P1, 0($10)</td>
<td>$5</td>
</tr>
<tr>
<td>2 add P2, $7, 1</td>
<td>$6</td>
</tr>
<tr>
<td>3 bne P1, $0, LOOP</td>
<td>$7</td>
</tr>
<tr>
<td>4 lw P3, 0(P1)</td>
<td>$10</td>
</tr>
<tr>
<td>5 add P4, P2, 1</td>
<td>$12</td>
</tr>
</tbody>
</table>

Valid  Value  In use

| P1  | 1      | 1       |
| P2  | 1      | 1       |
| P3  | 0      | 1       |
| P4  | 0      | 1       |
| P5  | 0      | 1       |
| P6  | 0      | 1       |
| P7  | 1      |         |
| P8  | 1      |         |
| P9  | 1      |         |
| P10 | 1      |         |
Register renaming in motion

Renamed instruction

1  lw  P1, 0($10)
2  add P2, $7, 1
3  bne P1, $0, LOOP
4  lw  P3, 0(P1)
5  add P4, P2, 1
6  bne P3, $0, LOOP
7  lw  P5, 0(P3)
8  add P6, P4, 1
9  bne P5, $0, LOOP
10

Physical Register

<table>
<thead>
<tr>
<th>Renamed Instruction</th>
<th>R</th>
<th>I</th>
<th>AR</th>
<th>LSQ</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  lw P1, 0($10)</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2  add P2, $7, 1</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3  bne P1, $0, LOOP</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4  lw P3, 0(P1)</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5  add P4, P2, 1</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6  bne P3, $0, LOOP</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7  lw P5, 0(P3)</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8  add P6, P4, 1</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9  bne P5, $0, LOOP</td>
<td>R</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid Value In use

<table>
<thead>
<tr>
<th>Renamed Instruction</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  lw P1, 0($10)</td>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>P6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2  add P2, $7, 1</td>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>P7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3  bne P1, $0, LOOP</td>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>P8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4  lw P3, 0(P1)</td>
<td>P4</td>
<td>0</td>
<td>1</td>
<td>P9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5  add P4, P2, 1</td>
<td>P5</td>
<td>0</td>
<td>1</td>
<td>P10</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Valid Value In use
Super Scalar
Since we have more functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!

Super-scalar: fetch/decode/issue more than one instruction each cycle
  - Fetch width: how many instructions can the processor fetch/decode each cycle
  - Issue width: how many instructions can the processor issue each cycle
Overview of a processor supporting register renaming

What if we widen the pipeline to fetch/issue two instructions at the same time?
2-issue RR processor in motion

1. `lw` $6,0($10)
2. `add` $7,$6,$12
3. `sw` $7,0($10)
4. `addi` $10,$10,8
5. `bne` $10,$5,LOOP
6. `lw` $6,0($10)
7. `add` $7,$6,$12
8. `sw` $7,0($10)
9. `addi` $10,$10,8
10. `bne` $10,$5,LOOP

### Renamed instruction

<table>
<thead>
<tr>
<th>No.</th>
<th>Instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>lw</code> P1, 0($10)</td>
<td>$5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>add</code> P2, P1, $12</td>
<td>$6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>$7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>$12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-issue RR processor in motion

lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP
lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP

Renamed instruction

<table>
<thead>
<tr>
<th></th>
<th>Renamed instruction</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lw</td>
<td>P1, 0($10)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>add</td>
<td>P2, P1, $12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>sw</td>
<td>P2, 0($10)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>addi</td>
<td>P3, $10, 8</td>
<td></td>
</tr>
</tbody>
</table>

Physical Register

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$ 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$ 6</td>
<td>R</td>
<td>P1</td>
</tr>
<tr>
<td>7</td>
<td>$ 7</td>
<td>R</td>
<td>P2</td>
</tr>
<tr>
<td>8</td>
<td>$10</td>
<td>R</td>
<td>P3</td>
</tr>
<tr>
<td>9</td>
<td>$12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid Value In use

<table>
<thead>
<tr>
<th></th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-issue RR processor in motion

- lw $6, 0($10)
- add $7, $6, $12
- sw $7, 0($10)
- addi $10, $10, 8
- bne $10, $5, LOOP
- lw $6, 0($10)
- add $7, $6, $12
- sw $7, 0($10)
- addi $10, $10, 8
- bne $10, $5, LOOP

Renamed instruction

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
</tr>
<tr>
<td>add P2, P1, $12</td>
<td>$6 P1</td>
</tr>
<tr>
<td>sw P2, 0($10)</td>
<td>$7 P2</td>
</tr>
<tr>
<td>addi P3, $10, 8</td>
<td>$10 P3</td>
</tr>
<tr>
<td>bne P3, $5, LOOP</td>
<td>$12</td>
</tr>
<tr>
<td>lw P4, 0(P3)</td>
<td></td>
</tr>
</tbody>
</table>

Physical Register

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P5</td>
<td>0</td>
<td>P10</td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-issue RR processor in motion

lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP
lw   $6,0($10)
add  $7,$6,$12
sw   $7,0($10)
addi $10,$10,8
bne  $10,$5,LOOP

Renamed instruction

<table>
<thead>
<tr>
<th></th>
<th>lw P1, 0($10)</th>
<th>add P2, P1, $12</th>
<th>sw P2, 0($10)</th>
<th>addi P3, $10, 8</th>
<th>bne P3, $5, LOOP</th>
<th>lw P4, 0(P3)</th>
<th>add P5, P1, $12</th>
<th>sw P5, 0(P3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Register

<table>
<thead>
<tr>
<th></th>
<th>$5</th>
<th>$6</th>
<th>$7</th>
<th>$10</th>
<th>$12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>I</td>
<td>AR</td>
<td>P1</td>
<td>P5</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

Valid Value In use

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>0</th>
<th>1</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P2</td>
<td>0</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>0</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td></td>
<td>P5</td>
<td>0</td>
<td>1</td>
<td>P10</td>
</tr>
</tbody>
</table>

64
2-issue RR processor in motion

1. `lw $6,0($10)`
2. `add $7,$6,$12`
3. `sw $7,0($10)`
4. `addi $10,$10,8`
5. `bne $10,$5,LOOP`
6. `lw $6,0($10)`
7. `add $7,$6,$12`
8. `sw $7,0($10)`
9. `addi $10,$10,8`
10. `bne $10,$5,LOOP`

Renamed instruction:

1. `lw P1, 0($10)`
2. `add P2, P1, $12`
3. `sw P2, 0($10)`
4. `addi P3, $10, 8`
5. `bne P3, $5, LOOP`
6. `lw P4, 0(P3)`
7. `add P5, P1, $12`
8. `sw P5, 0(P3)`
9. `addi P6, P3, 8`
10. `bne P6, 0($10)`

Physical Register:

- **$5**
- **$6**
- **$7**
- **$10**
- **$12**

**Valid** | **Value** | **In use**
--- | --- | ---
**P1** | 0 | 1
**P2** | 0 | 1
**P3** | 1 | 1
**P4** | 0 | 1
**P5** | 0 | 1
**P6** | 0 | 1
**P7** | 0 | 1
**P8** | 1 | 1
**P9** | 0 | 1
**P10** | 0 | 1
2-issue RR processor in motion

Renamed instruction

1 lw P1, 0($10)
2 add P2, P1, $12
3 sw P2, 0($10)
4 addi P3, $10, 8
5 bne P3, P5, LOOP
6 lw P4, 0(P3)
7 add P5, P1, $12
8 sw P5, 0(P3)
9 addi P6, P3, 8
10 bne P6, 0($10)

Physical Register

P1 1 1 P6
P2 0 1 P7
P3 1 1 P8
P4 0 1 P9
P5 0 1 P10
2-issue RR processor in motion

- lw   $6,0($10)
- add  $7,$6,$12
- sw   $7,0($10)
- addi $10,$10,8
- bne  $10,$5,LOOP
- lw   $6,0($10)
- add  $7,$6,$12
- sw   $7,0($10)
- addi $10,$10,8
- bne  $10,$5,LOOP

Renamed instruction:
1. lw   P1, 0($10)
2. add  P2, P1, $12
3. sw   P2, 0($10)
4. addi P3, $10, 8
5. bne  P3, 5, LOOP
6. lw   P4, 0(P3)
7. add  P5, P1, $12
8. sw   P5, 0(P3)
9. addi P6, P3, 8
10. bne P6, 0($10)

Physical Register:
- $5
- $6  P1
- $7  P5
- $10 P3
- $12

Valid  Value  In use  Valid  Value  In use
P1  1   1     P6  1
P2  0   1     P7  1
P3  1   1     P8  1
P4  0   1     P9  1
P5  0   1     P10 1

Valid  Value  In use  Valid  Value  In use
P1  1   1     P6  1
P2  0   1     P7  1
P3  1   1     P8  1
P4  0   1     P9  1
P5  0   1     P10 1
2-issue RR processor in motion

 lw $6,0($10)
 add $7,$6,$12
 sw $7,0($10)
 addi $10,$10,8
 bne $10,$5,LOOP
 lw $6,0($10)
 add $7,$6,$12
 sw $7,0($10)
 addi $10,$10,8
 bne $10,$5,LOOP

 Renamed instruction
 1 lw P1, 0($10)
 2 add P2, P1, $12
 3 sw P2, 0($10)
 4 addi P3, $10, 8
 5 bne P9, $5, LOOP
 6 lw P4, 0(P3)
 7 add P5, P1, $12
 8 sw P5, 0(P3)
 9 addi P6, P3, 8
 10 bne P6, 0($10)

 Physical Register

 Valid | Value | In use
 P1    | 1     | 1    | P6
 P2    | 1     | 1    | P7
 P3    | 1     | 1    | P8
 P4    | 0     | 1    | P9
 P5    | 0     | 1    | P10

 Valid | Value | In use
 $5    | 1     | 1    |
 $6    | P1    |
 $7    | P5    |
 $10   | P3    |
 $12   |

 Valid | Value | In use
 $5    | 1     | 1    |
 $6    | P1    |
 $7    | P5    |
 $10   | P3    |
 $12   |
2-issue RR processor in motion

lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP
lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP

Renamed instruction

1 lw P1, 0($10)
2 add P2, P1, $12
3 sw P2, 0($10)
4 addi P3, $10, 8
5 bne P9, $5, LOOP
6 lw P4, 0(P3)
7 add P5, P1, $12
8 sw P5, 0(P3)
9 addi P6, P3, 8
10 bne P6, 0($10)

Physical Register

$5
$6
$7
$10
$12

Valid | Value | In use
--- | --- | ---
P1 | 1 | 1
P2 | 1 | 1
P3 | 1 | 1
P4 | 1 | 1
P5 | 0 | 1
P6 |  |  
P7 |  |  
P8 |  |  
P9 |  |  
P10 |  |  

Valid | Value | In use
--- | --- | ---
P1 | 1 | 1
P2 | 1 | 1
P3 | 1 | 1
P4 | 1 | 1
P5 | 0 | 1
P6 |  |  
P7 |  |  
P8 |  |  
P9 |  |  
P10 |  |  

Making the question into a table.
2-issue RR processor in motion

1. `lw` $6,0($10)
2. `add` $7,$6,$12
3. `sw` $7,0($10)
4. `addi` $10,$10,8
5. `bne` $10,$5,LOOP
6. `lw` $6,0($10)
7. `add` $7,$6,$12
8. `sw` $7,0($10)
9. `addi` $10,$10,8
10. `bne` $10,$5,LOOP

Renamed instruction

1. `lw` P1, 0($10)
2. `add` P2, P1, $12
3. `sw` P2, 0($10)
4. `addi` P3, $10, 8
5. `bne` P9, $5, LOOP
6. `lw` P4, 0(P3)
7. `add` P5, P1, $12
8. `sw` P5, 0(P3)
9. `addi` P6, P3, 0
10. `bne` P6, 0($10)

Physical Register

<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td>P1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$7</td>
<td>P5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>P6</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>P7</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>P8</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>P9</td>
</tr>
<tr>
<td>P5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
2-issue RR processor in motion

lw  $6,0($10)
add $7,$6,$12
sw  $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP
lw  $6,0($10)
add $7,$6,$12
sw  $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP

Renamed instruction
1 lw  P1, 0($10)
2 add P2, P1, $12
3 sw  P2, 0($10)
4 addi P3, $10, 8
5 bne P9, $5, LOOP
6 lw  P4, 0(P3)
7 add P5, P1, $12
8 sw  P5, 0(P3)
9 addi P6, P3, 0
10 bne P6, 0($10)

Physical Register
<table>
<thead>
<tr>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10</td>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$12</td>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-issue RR processor in motion

1. `lw   $6,0($10)`
2. `add  $7,6,$12`
3. `sw   $7,0($10)`
4. `addi $10,10,8`
5. `bne  $10,$5,LOOP`
6. `lw   $6,0($10)`
7. `add  $7,6,$12`
8. `sw   $7,0($10)`
9. `addi $10,10,8`
10. `bne  $10,$5,LOOP`

Renamed instruction:
1. `lw   P1, 0($10)`
2. `add  P2, P1, $12`
3. `sw   P2, 0($10)`
4. `addi P3, $10, 8`
5. `bne  P9, $5, LOOP`
6. `lw   P4, 0(P3)`
7. `add  P5, P1, $12`
8. `sw   P5, 0(P3)`
9. `addi P6, P3, 0`
10. `bne  P6, 0($10)`
2-issue RR processor in motion

<table>
<thead>
<tr>
<th>Renamed instruction</th>
<th>Physical Register</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
<th>Valid</th>
<th>Value</th>
<th>In use</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw P1, 0($10)</td>
<td>$5</td>
<td>1</td>
<td>1</td>
<td>P6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add P2, P1, $12</td>
<td>$6</td>
<td>1</td>
<td>1</td>
<td>P7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw P2, 0($10)</td>
<td>$7</td>
<td>1</td>
<td>1</td>
<td>P8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi P5, $10, 8</td>
<td>$10</td>
<td>1</td>
<td>1</td>
<td>P9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne P5, $5, LOOP</td>
<td>$12</td>
<td>1</td>
<td>1</td>
<td>P10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-issue RR processor in motion

1. lw $6,0($10)
2. add $7,$6,$12
3. sw $7,0($10)
4. addi $10,$10,8
5. bne $10,$5,LOOP
6. lw $6,0($10)
7. add $7,$6,$12
8. sw $7,0($10)
9. addi $10,$10,8
10. bne $10,$5,LOOP
2-issue RR processor in motion

lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP
lw $6,0($10)
add $7,$6,$12
sw $7,0($10)
addi $10,$10,8
bne $10,$5,LOOP

Renamed instruction

1 lw P1, 0($10)
2 add P2, P1, $12
3 sw P2, 0($10)
4 addi P3, $10, 8
5 bne P9, $5, LOOP
6 lw P4, 0(P3)
7 add P5, P1, $12
8 sw P5, 0(P3)
9 addi P6, P3, 8
10 bne P6, 0($10)

Physical Register

Valid Value In use

$5
P5
$6 P1
$7 P5
$10 P3
$12 P3

Valid Value In use

P1 1 1 P6
P2 1 1 P7
P3 1 1 P8
P4 1 1 P9
P5 1 1 P10

75
For the following C code and its translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )

A. 9
B. 10
C. 11
D. 12
E. 13
What about “linked list” (2)

- For the following C code and it’s translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```c
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL )
```

```mips
LOOP: lw $10, 8($10)
      addi $7, $7, 1
      bne $10, $0, LOOP
```

A. 9  
B. 10  
C. 11  
D. 12  
E. 13
What about “linked list” (2)

• For the following C code and its translation in MIPS, how many cycles it takes the processor to issue all instructions? Assume the current PC is already at the first instruction and this linked list has only three nodes. This processor can fetch 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```c
    do {
        number_of_nodes++;
        current = current->next;
    } while ( current != NULL )
```

```mips
    LOOP: lw $10, 8($10)
          addi $7, $7, 1
          bne $10, $0, LOOP
```

A. 9
B. 10
C. 11
D. 12
E. 13
What about “linked list”

Static instructions

```
LOOP: lw $10, 8($10)
addi $7, $7, 1
bne $10, $0, LOOP
```

Dynamic instructions

```
1. lw $10, 8($10)
2. addi $7, $7, 1
3. bne $10, $0, LOOP
4. lw $10, 8($10)
5. addi $7, $7, 1
6. bne $10, $0, LOOP
7. lw $10, 8($10)
8. addi $7, $7, 1
9. bne $10, $0, LOOP
```

Instruction Queue

1 3 2 5 7

What if (6) is mis-predicted

X7 is changed by (8) already!!!
Announcements

• CAPE/Survey
  • Screenshot of your CAPE
  • Fill the survey
  • Count as a full-credit assignment and we're dropping your lowest two assignments now.

• Assignment 5 is up.
  • Given that we're dropping 2 lowest assignment grade and give you a full credit one once you submitted your post-CAPE survey, you probably don't need to turn in that.
  • Strongly encourage to practice that since it covers the material for the last week of class

• Regarding final exam —
  • 9/4 8am—6pm — any consecutive, non-stop 3-hour slot you pick
  • Open books, open notes, but it's going to be twice longer than the midterm
  • Lockdown browser
    • Test at https://canvas.ucsd.edu/courses/17390/quizzes/31824

• No zoom, no response to piazza posts for fairness