Lecture: Static ILP

- Topics: loop unrolling, software pipelines (Sections C.5, 3.2)
- HW3 posted, due in a week
Assume that there is a dependence where the final result of the first instruction is required before starting the second instruction.
Problem 2

• Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 20-stage and 40-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.
Problem 2

- Assume an unpipelined processor where it takes 5ns to go through the circuits and 0.1ns for the latch overhead. What is the throughput for 1-stage, 20-stage and 50-stage pipelines? Assume that the P.O.P and P.O.C in the unpipelined processor are separated by 2ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

  - 1-stage: 1 instr every 5.1ns
  - 20-stage: first instr takes 0.35ns, the second takes 2.8ns
  - 50-stage: first instr takes 0.2ns, the second takes 4ns
  - Throughputs: 0.20 BIPS, 0.63 BIPS, and 0.48 BIPS
ILP

- Instruction-level parallelism: overlap among instructions: pipelining or multiple instruction execution

- What determines the degree of ILP?
  - dependences: property of the program
  - hazards: property of the pipeline
Static vs Dynamic Scheduling

- Arguments against dynamic scheduling:
  - requires complex structures to identify independent instructions (scoreboards, issue queue)
    - high power consumption
    - low clock speed
    - high design and verification effort
  - the compiler can “easily” compute instruction latencies and dependences – complex software is always preferred to complex hardware (?)
Loop Scheduling

• The compiler’s job is to minimize stalls

• Focus on loops: account for most cycles, relatively easy to analyze and optimize
Assumptions

- Load: 2-cycles (1 cycle stall for consumer)
- FP ALU: 4-cycles (3 cycle stall for consumer; 2 cycle stall if the consumer is a store)
- One branch delay slot
- Int ALU: 1-cycle (no stall for consumer, 1 cycle stall if the consumer is a branch)

LD -> any : 1 stall
FPALU -> any: 3 stalls
FPALU -> ST : 2 stalls
IntALU -> BR : 1 stall
Loop Example

for (i=1000; i>0; i--)
    x[i] = x[i] + s;

Loop:

    L.D      F0, 0(R1)     ; F0 = array element
    ADD.D    F4, F0, F2    ; add scalar
    S.D      F4, 0(R1)     ; store result
    DADDUI   R1, R1,# -8  ; decrement address pointer
    BNE      R1, R2, Loop  ; branch if R1 != R2
    NOP

Source code

Assembly code

LD -> any: 1 stall
FPALU -> any: 3 stalls
FPALU -> ST: 2 stalls
IntALU -> BR: 1 stall
## Loop Example

```
for (i=1000; i>0; i--)
    x[i] = x[i] + s;
```

### Source code

<table>
<thead>
<tr>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loop:</strong></td>
</tr>
<tr>
<td>L.D F0, 0(R1) ; F0 = array element</td>
</tr>
<tr>
<td>ADD.D F4, F0, F2 ; add scalar</td>
</tr>
<tr>
<td>S.D F4, 0(R1) ; store result</td>
</tr>
<tr>
<td>DADDUI R1, R1,# -8 ; decrement address pointer</td>
</tr>
<tr>
<td>BNE R1, R2, Loop ; branch if R1 != R2</td>
</tr>
<tr>
<td>NOP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10-cycle schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD -&gt; any : 1 stall</td>
</tr>
<tr>
<td>FPALU -&gt; any: 3 stalls</td>
</tr>
<tr>
<td>FPALU -&gt; ST : 2 stalls</td>
</tr>
<tr>
<td>IntALU -&gt; BR : 1 stall</td>
</tr>
</tbody>
</table>

---

### Loop Schedule

```
Loop:      L.D         F0, 0(R1)          ; F0 = array element
           ADD.D    F4, F0, F2        ; add scalar
           S.D         F4, 0(R1)          ; store result
           DADDUI  R1, R1,# -8      ; decrement address pointer
           BNE        R1, R2, Loop    ; branch if R1 != R2
           NOP
```
Smart Schedule

- By re-ordering instructions, it takes 6 cycles per iteration instead of 10
- We were able to violate an anti-dependence easily because an immediate was involved
- Loop overhead (instrs that do book-keeping for the loop): 2
  Actual work (the ld, add.d, and s.d): 3 instrs
  Can we somehow get execution time to be 3 cycles per iteration?
Problem 1

for (i=1000; i>0; i--)
x[i] = y[i] * s;

Source code

Loop:  L.D    F0, 0(R1)      ; F0 = array element
       MUL.D  F4, F0, F2      ; multiply scalar
       S.D    F4, 0(R2)       ; store result
       DADDUI R1, R1,#-8      ; decrement address pointer
       DADDUI R2, R2,#-8      ; decrement address pointer
       BNE    R1, R3, Loop    ; branch if R1 != R3
       NOP

Assembly code

• How many cycles do the default and optimized schedules take?
Problem 1

for (i=1000; i>0; i--)
  x[i] = y[i] * s;

Source code

Loop:
  L.D          F0, 0(R1)  ; F0 = array element
  MUL.D        F4, F0, F2 ; multiply scalar
  S.D          F4, 0(R2)  ; store result
  DADDUI       R1, R1, # -8 ; decrement address pointer
  DADDUI       R2, R2, # -8 ; decrement address pointer
  BNE          R1, R3, Loop ; branch if R1 != R3
  NOP

Assembly code

• How many cycles do the default and optimized schedules take?

Unoptimized: LD 1s MUL 4s SD DA DA BNE 1s -- 12 cycles

Optimized: LD DA MUL DA 2s BNE SD -- 8 cycles
Loop Unrolling

- Loop overhead: 2 instrs; Work: 12 instrs
- How long will the above schedule take to complete?
Scheduled and Unrolled Loop

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0, 0(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F6, -8(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F10, -16(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F14, -24(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4, F0, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8, F6, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12, F10, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16, F14, F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F4, 0(R1)</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F8, -8(R1)</td>
<td></td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1, R1, # -32</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F12, 16(R1)</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>R1, R2, Loop</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F16, 8(R1)</td>
<td></td>
</tr>
</tbody>
</table>

- Execution time: 14 cycles or 3.5 cycles per original iteration

- LD -> any: 1 stall
- FPALU -> any: 3 stalls
- FPALU -> ST: 2 stalls
- IntALU -> BR: 1 stall
Loop Unrolling

- Increases program size
- Requires more registers
- To unroll an n-iteration loop by degree k, we will need \((n/k)\) iterations of the larger loop, followed by \((n \mod k)\) iterations of the original loop
Automating Loop Unrolling

- Determine the dependences across iterations: in the example, we knew that loads and stores in different iterations did not conflict and could be re-ordered

- Determine if unrolling will help – possible only if iterations are independent

- Determine address offsets for different loads/stores

- Dependency analysis to schedule code without introducing hazards; eliminate name dependences by using additional registers
Problem 2

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Source code

Loop:
  L.D  F0, 0(R1)   ; F0 = array element
  MUL.D F4, F0, F2 ; multiply scalar
  S.D  F4, 0(R2)   ; store result
  DADDUI R1, R1,#-8 ; decrement address pointer
  DADDUI R2, R2,#-8 ; decrement address pointer
  BNE  R1, R3, Loop ; branch if R1 != R3
  NOP

Assembly code

• How many unrolls does it take to avoid stall cycles?
Problem 2

```plaintext
for (i=1000; i>0; i--)
    x[i] = y[i] * s;
```

**Source code**

**Assembly code**

```
Loop:
    L.D    F0, 0(R1)               ; F0 = array element
    MUL.D  F4, F0, F2              ; multiply scalar
    S.D    F4, 0(R2)               ; store result
    DADDUI R1, R1,#-8              ; decrement address pointer
    DADDUI R2, R2,#-8              ; decrement address pointer
    BNE    R1, R3, Loop            ; branch if R1 != R3
    NOP
```

- How many unrolls does it take to avoid stall cycles?

  Degree 2: LD LD MUL MUL DA DA 1s SD BNE SD
  Degree 3: LD LD LD MUL MUL MUL MUL DA DA SD SD BNE SD
  – 12 cyc/3 iterations

**Cycle Count**

- LD -> any : 1 stall
- FPMUL -> any: 5 stalls
- FPMUL -> ST : 4 stalls
- IntALU -> BR : 1 stall
### Superscalar Pipelines

<table>
<thead>
<tr>
<th>Integer pipeline</th>
<th>FP pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handles L.D, S.D, ADDUI, BNE</td>
<td>Handles ADD.D</td>
</tr>
</tbody>
</table>

- What is the schedule with an unroll degree of 5?
Superscalar Pipelines

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Integer pipeline</th>
<th>FP pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L.D F0,0(R1)</td>
<td>ADD.D F4,F0,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F6,-8(R1)</td>
<td>ADD.D F8,F6,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F10,-16(R1)</td>
<td>ADD.D F12,F10,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F14,-24(R1)</td>
<td>ADD.D F16,F14,F2</td>
</tr>
<tr>
<td></td>
<td>L.D F18,-32(R1)</td>
<td>ADD.D F20,F18,F2</td>
</tr>
<tr>
<td></td>
<td>S.D F4,0(R1)</td>
<td>S.D F12,-16(R1)</td>
</tr>
<tr>
<td></td>
<td>S.D F8,-8(R1)</td>
<td>DADDUI R1,R1,# -40</td>
</tr>
<tr>
<td></td>
<td>S.D F16,16(R1)</td>
<td>BNE R1,R2,Loop</td>
</tr>
<tr>
<td></td>
<td>S.D F20,8(R1)</td>
<td>S.D F20,8(R1)</td>
</tr>
</tbody>
</table>

- Need unroll by degree 5 to eliminate stalls (fewer if we move DADDUI up)
- The compiler may specify instructions that can be issued as one packet
- The compiler may specify a fixed number of instructions in each packet: Very Large Instruction Word (VLIW)
Problem 3

for (i=1000; i>0; i--)
    x[i] = y[i] * s;

Loop:     L.D         F0, 0(R1)          ; F0 = array element
           MUL.D    F4, F0, F2        ; multiply scalar
           S.D         F4, 0(R2)          ; store result
           DADDUI  R1, R1,# -8      ; decrement address pointer
           DADDUI  R2, R2,#-8       ; decrement address pointer
           BNE        R1, R3, Loop    ; branch if R1 != R3
           NOP

Source code

Assembly code

LD -> any : 1 stall
FPMUL -> any: 5 stalls
FPMUL -> ST : 4 stalls
IntALU -> BR : 1 stall

• How many unrolls does it take to avoid stalls in the superscalar pipeline?
Problem 3

for (i=1000; i>=0; i--)
    x[i] = y[i] * s;

<table>
<thead>
<tr>
<th>Source code</th>
<th>Assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop:</td>
<td>L.D F0, 0(R1) ; F0 = array element</td>
</tr>
<tr>
<td></td>
<td>MUL.D F4, F0, F2 ; multiply scalar</td>
</tr>
<tr>
<td></td>
<td>S.D F4, 0(R2) ; store result</td>
</tr>
<tr>
<td></td>
<td>DADDUI R1, R1,#-8 ; decrement address pointer</td>
</tr>
<tr>
<td></td>
<td>DADDUI R2, R2,#-8 ; decrement address pointer</td>
</tr>
<tr>
<td></td>
<td>BNE R1, R3, Loop ; branch if R1 != R3</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>

• How many unrolls does it take to avoid stalls in the superscalar pipeline?

LD
LD
LD  MUL
LD  MUL
LD  MUL
LD  MUL
LD  MUL
SD  MUL

7 unrolls. Could also make do with 5 if we moved up the DADDUIs.
Title

- Bullet