Instruction Level Parallelism

Pipeline with data forwarding and accelerated branch

Loop Unrolling

Multiple Issue -- Multiple functional Units

Static vs Dynamic Optimizations
Optimization Example

C-code

```c
k = len;
do {
    k--;  
} while(k > 0)
```
Register Usage

$s0 \quad \text{len}
$s1 \quad \text{base address of A}
$s2 \quad x

$t1 \quad \text{address of A[k]}
$t0 \quad \text{value of A[k] (old and new)}
Basic loop using pointer hopping

```
sll  $t1, $s0, 2
addu $t1, $t1, $s1

loop:
    addi $t1, $t1, -4
    lw   $t0, 0($t1)
    add  $t0, $t0, $s2
    sw   $t0, 0($t1)
    bne  $t1, $s1, loop
```

xxx
Time for 1000 Iterations for Single- and Multi-cycle

• Single Cycle
  – Every instruction takes 800 picoseconds (ps)
  – Time = $5 \times 800 \times 1000 + 2 \times 800 = 4,001,600$ ps = 4001.6 nanoseconds (ns)

• Multicycle
  – 200 ps per cycle, variable number of CPI
  – Cycles = $(1 \times 3 + 3 \times 4 + 1 \times 5) \times 1000 + 2 \times 4 = 20,008$
  – Time = $20,008 \times 200$ ps = 4,001.6 ns
Pipeline
Filling Stall/Delay slots

```
sll  $t1, $s0, 2
addu $t1, $t1, $s1

loop:
  addi $t1, $t1, -4
  lw   $t0, 0($t1)
  nop
  add  $t0, $t0, $s2
  sw   $t0, 0($t1)
  bne  $t1, $s1, loop
  nop

xxx
```
Time for simple pipeline

- 200 ps per cycle, 1 CPI (including nops)

- Time = 7x200x1000 + 2x200 ps = 1,400.4 ns
Reordering Code
to fill branch delay and stall slots

sll  $t1, $s0, 2
addu $t1, $t1, $s1

loop:

lw   $t0, -4($t1)
addi $t1, $t1, -4
add  $t0, $t0, $s2
bne  $t1, $s1, loop
sw   $t0, 0($t1)

xxx
Time for pipeline with reordered code

- 200 ps per cycle, 1 CPI (including nops)

- Time = 5x200x1000 + 2x200 ps = 1,000.4 ns
Loop Unrolling step 1 (4 iterations)

```assembly
sll $t1, $s0, 2
addu $t1, $t1, $s1

loop:
    lw   $t0, -4($t1)
    addi $t1, $t1, -4
    add  $t0, $t0, $s2
    beq  $t1, $s1, loopend
    sw   $t0, 0($t1)

lw   $t0, -4($t1)
addi $t1, $t1, -4
add  $t0, $t0, $s2
beq  $t1, $s1, loopend
sw   $t0, 0($t1)
```

loopend:
xxx
Loop Unrolling step 2
One pointer with offsets

```
sll $t1, $s0, 2
addu $t1, $t1, $s1
loop:
  addi $t1, $t1, -16
  lw  $t0, 12($t1)
  nop
  add  $t0, $t0, $s2
  sw  $t0, 12($t1)
  lw  $t0, 8($t1)
  nop
  add  $t0, $t0, $s2
  sw  $t0, 8($t1)
  lw  $t0, 4($t1)
  nop
  add  $t0, $t0, $s2
  sw  $t0, 4($t1)
  lw  $t0, 0($t1)
  nop
  add  $t0, $t0, $s2
  sw  $t0, 0($t1)
  bne  $t1, $s1, loop
  sw  $t0, 0($t1)
```
Loop Unrolling step 3
Filling data hazard slots with register renaming

sll $t1, $s0, 2
addu $t1, $t1, $s1

loop:
addi $t1, $t1, -16
lw $t0, 12($t1)
lw $t3, 8($t1)
add $t0, $t0, $s2
sw $t0, 12($t1)

lw $t0, 4($t1)
add $t3, $t3, $s2
sw $t3, 8($t1)

lw $t3, 0($t1)
add $t0, $t0, $s2
sw $t0, 4($t1)

add $t3, $t3, $s2
bne $t1, $s1, loop
sw $t3, 0($t1)

xxx
Time for pipeline with loop unrolling

- 200 ps per cycle, 1 CPI (including nops)
- 4 iterations per loop means 250 times in loop

- Time = 14x200x250 + 2x200 ps = 700.4 ns
Dual Pipeline with Our MIPS

Upper path for ALU ops, Lower path for lw/sw
Dual Pipeline Requirements

- Two instruction (64 bits) IR
- Two additional read ports in register file
- An additional write port in register file
- Appropriate forwarding hardware

- Compiler must insure no conflicts between the two simultaneous instructions

- Instruction pairing and order can be done statically by compiler
Dual Pipeline

• Two instruction pipe
  – one for arithmetic or branch
  – one for load or store

• Instructions can be issued at same time
  – if no data dependencies
  – following instructions follow same delay rules

• Loop unrolling for more overlap

• Register renaming to avoid data dependency
Dual Pipeline Code pairing instructions

sll  $t1, $s0, 2
addu $t1, $t1, $s1

loop:
addi $t1, $t1, -4
lw   $t0, 0($t1)

nop
add  $t0, $t0, $s2

bne  $t1, $s1, loop

nop

sw   $t0, 0($t1)

xxx
Dual Pipeline Code
fill branch delay slot

sll $t1, $s0, 2
addu $t1, $t1, $s1
addi $t1, $t1, -4

loop:
    lw $t0, 0($t1)
nop
    add $t0, $t0, $s2
    bne $t1, $s1, loop
addi $t1, $t1, -4

xxx
Time for dual pipeline (no loop unrolling)

- 200 ps per cycle, 1 or 2 instr per cycle

- Time = 5x200x1000 + 3x200 ps = 1,000.6 ns
Loop Unrolling step 2
One pointer with offsets

```
sll $t1, $s0, 2
addu $t1, $t1, $s1
loop:
   addi $t1, $t1, -16
   lw $t0, 12($t1)
nop
   add $t0, $t0, $s2
   sw $t0, 12($t1)
   lw $t0, 8($t1)
nop
   add $t0, $t0, $s2
   sw $t0, 8($t1)
   lw $t0, 4($t1)
   nop
   add $t0, $t0, $s2
   sw $t0, 4($t1)
   lw $t0, 0($t1)
nop
   add $t0, $t0, $s2
   sw $t0, 0($t1)
   bne $t1, $s1, loop
   sw $t0, 0($t1)
xxx
```
Dual Pipe Optimization with loop unrolling

Unrolled and reordered loop

```
sll  $t1, $s0, 2
addu $t1, $t1, $s1
loop:
  addi $t1, $t1, -16
  lw   $t0, 12($t1)
  lw   $t3, 8($t1)
  add  $t0, $t0, $s2
  sw   $t0, 12($t1)
  lw   $t3, 0($t1)
  add  $t0, $t0, $s2
  sw   $t3, 8($t1)
  bne  $t1, $s1, loop
  sw   $t3, 0($t1)
loopend:
xxx
```
step 1, use more registers
(register renaming)

sll $t1, $s0, 2
addu $t1, $t1, $s1
loop:
  addi $t1, $t1, -16
  lw  $t0, 12($t1)
  lw  $t3, 8($t1)
  add $t0, $t0, $s2
  sw  $t0, 12($t1)
  lw  $t5, 4($t1)
  lw  $t7, 0($t1)
  add $t5, $t5, $s2
  sw  $t5, 4($t1)
  add $t7, $t7, $s2
  bne $t1, $s1, loop
  sw  $t7, 0($t1)
loopend:
  xxx
step 2, reorder/pair instructions

```
sll  $t1, $s0, 2
addu $t1, $t1, $s1

loop:
    addi $t1, $t1, -16
    lw   $t0, 12($t1)
    lw   $t3, 8($t1)
    lw   $t5, 4($t1)
    add  $t0, $t0, $s2
    add  $t3, $t3, $s2
    add  $t5, $t5, $s2
    lw   $t7, 0($t1)
    add  $t7, $t7, $s2
    bne  $t1, $s1, loop
    lw   $t0, 12($t1)
    lw   $t5, 4($t1)
    sw   $t0, 12($t1)
    lw   $t7, 0($t1)
    sw   $t3, 8($t1)
    lw   $t7, 0($t1)
    sw   $t5, 4($t1)
    sw   $t7, 0($t1)
    nop

xxx
```
step 2, fill branch delay

sll  $t1, $s0, 2
addu $t1, $t1, $s1
addi $t1, $t1, -16

loop:

lw   $t3, 8($t1)
add  $t0, $t0, $s2
add  $t3, $t3, $s2
sw   $t0, 12($t1)
add  $t5, $t5, $s2
lw   $t5, 4($t1)
add  $t7, $t7, $s2
lw   $t7, 0($t1)
bne  $t1, $s1, loop
addi $t7, $t7, -16
add  $t0, $t0, $s2
lw   $t0, -4($t1)

xxx
Time for dual pipeline

- 200 ps per cycle, 1 or 2 instr per cycle
- 4 iterations per loop, 250 times through loop

- Time = 8\times 200 \times 250 + 4\times 200 \text{ ps} = 400.8 \text{ ns}

- 10 times faster than single cycle or multi-cycle
- 3 \frac{1}{2} times faster than simple pipeline
- 1 \frac{3}{4} times faster than single pipeline, loop unrolled
More Parallelism?

• Suppose loop has more operations
  – Multiplication (takes longer than adds)
  – Floating point (takes much longer than integer)

• More parallel pipelines for different operations – all the above techniques could result in better performance

• Static (as above) vs dynamic reordering

• Speculation

• Out-of-order execution
Multiple Issue – Multiple Functional Units

Instruction fetch and decode unit

In-order issue

Reservation station
Reservation station

...

Reservation station
Reservation station

Functional units

Integer
Integer

...

Floating point
Load-store

Out-of-order execute

Commit unit

In-order commit
Dynamic Scheduling

• Processor determines which instructions to issue
• Reservation unit use copies of operand values from registers (equivalent to renaming)
• When operation is completed, results are buffered in the commit unit
• When can be done in order, results are written to registers or memory
• Speculation
  – guessing which way a branch goes
  – Guessing that a lw does not conflict with a sw
FIGURE 4.74 The microarchitecture of AMD Opteron X4. The extensive queues allow up to 106 RISC operations to be outstanding, including 24 integer operations, 36 floating point/SSE operations, and 44 loads and stores. The load and store units are actually separated into two parts, with the first part handling address calculation in the Integer ALU units and the second part responsible for the actual memory reference. There is an extensive bypass network among the functional units; since the pipeline is dynamic rather than static, bypassing is done by tagging results and tracking source operands, so as to allow a match when a result is produced for an instruction in one of the queues that needs the result. Copyright © 2009 Elsevier, Inc. All rights reserved.
FIGURE 4.75 The Opteron X4 pipeline showing the pipeline flow for a typical instruction and the number of clock cycles for the major steps in the 12-stage pipeline for integer RISC-operations. The floating point execution queue is 17 stages long. The major buffers where RISC-operations wait are also shown. Copyright © 2009 Elsevier, Inc. All rights reserved.