Multiple Instruction Issue

Multiple instructions issued each cycle

⇒ better performance

increase instruction throughput

decrease in CPI (below 1)

⇒ greater hardware complexity, potentially longer wire lengths

⇒ harder code scheduling job for the compiler

Superscalar processors

- instructions are scheduled by the hardware
- different numbers of instructions may be issued simultaneously

VLIW ("very long instruction word") **processors**

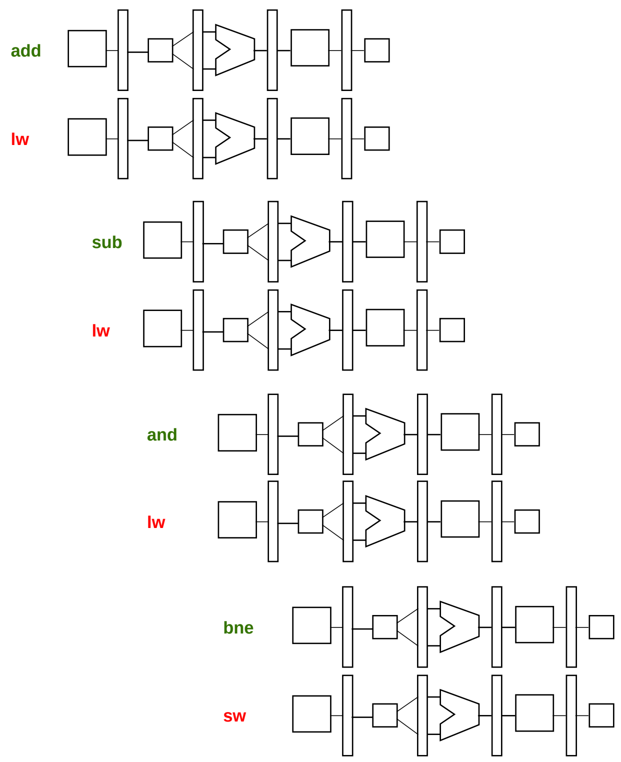
- instructions are scheduled by the compiler
- a fixed number of operations are formatted as one big instruction
- usually **LIW** (3 operations) today

Superscalars

Definition:

- a processor that can execute more than one instruction per cycle
- for example,
  integer computation, floating point computation, data transfer, transfer of control
- **issue width** = the number of **issue slots**, 1 slot/instruction
- not all types of instructions can be issued in each issue slot

2-way Superscalar



Superscalars

Require:

- instruction fetching of multiple instructions at once
    - dynamic branch prediction & fetching beyond conditional branches
- methods for determining which instructions can issue next
- the ability to issue multiple instructions in parallel
- methods for committing several instructions in fetch order

In-order vs. Out-of-order Execution

In-order instruction execution

- instructions are fetched, executed & complete in compiler-generated order
- if one instruction stalls, all instructions behind it stall
- instructions are **statically scheduled** by the hardware
  - means they are scheduled in their compiler-generated order
  - how many of the next *n* instructions can be issued, where *n* is the superscalar issue width
  - superscalars can have structural & data hazards within the *n* instructions
- 2 styles of static instruction scheduling
  - dispatch buffer (Alpha 21164)
  - shift register model (UltraSPARC-1)
- advantages of in-order instruction scheduling: simpler implementation
  - faster clock cycle
  - fewer transistors

In-order vs. Out-of-order Execution

Out-of-order instruction execution

- instructions are fetched in compiler-generated order
- instruction completion may be in-order (today) or out-of-order (older computers)
- in between they may be executed out of their compiler-generated order
- instructions behind a stalled instruction can pass it
- instructions are **dynamically scheduled** by the hardware
  - hardware decides in what order instructions can be executed
- advantages: higher performance
  - better at hiding latencies, less processor stalling
  - higher utilization of functional units

Alpha 21164

In-order instruction issue

- **instruction slotting**
    - after decode, instructions “arranged” according to functional unit type
    - constraints on types of instructions that can issue together & hardware capabilities
        - an example:
            - 2 integer ops, load/store, FP
            - 1 integer ALU does shifts & integer multiplies; the other executes branches
    - can issue up to 4 instructions
- completely empty the instruction buffer before fill it again
    - compiler can pad with `nops` so the second (conflicting) instruction is issued with the following instructions, not alone
- no data dependences in same issue cycle (some exceptions)
- hardware to:
    - detect data hazards
    - control bypass logic

UltraSparc 1

In-order instruction issue

- **shift register model**
- (almost) any instruction in any slot
    - choose from 2 integer, 2 FP/graphics, 1 load/store, 1 branch
- some data dependent instructions can issue in same cycle

Superscalars

Performance impact:

- increase performance because execute instructions in parallel, not just overlapped
- CPI potentially < 1 (.5 on our R3000 example)
- IPC (instructions/cycle) potentially > 1 (2 on our R3000 example)
- better functional unit utilization

but

- need to fetch more instructions – how many?
- need **independent instructions** to execute instructions in parallel
  i.e., their operands have to be different
  this is called **instruction-level parallelism (ILP)**

```
lw $t0, 0(s1)
addu $t0, $t0, $s2
```

- need a **good mix of instructions** to utilize the functional units
- need more instructions to hide load delays – why?
- need to make better branch predictions – why?

Code Scheduling on Superscalars

```
Loop:  lw $t0, 0(s1)
       addu $t0, $t0, $s2
       sw $t0, 0($s1)
       addi $s1, $s1, -4
       bne $s1, $0, Loop

Loop:  lw $t0, 0(s1)
       addi $s1, $s1, -4
       addu $t0, $t0, $s2
       sw $t0, 4($s1)
       bne $s1, $0, Loop
```

	ALU/branch instruction	Data transfer instruction	clock cycle
`Loop:`	`addi $s1, $s1, -4`	`lw $t0, 0($s1)`	1
			2
	`addu $t0, $t0, $s2`		3
	`bne $s1, $0, Loop`	`sw $t0, 4($s1)`	4

`lw addu sw` is the **critical path**

Illustrates why CPI on a dual-issue processor is not usually .5!

Code Scheduling on Superscalars

	ALU/branch instruction	Data transfer instruction	clock cycle
`Loop:`	`addi $s1, $s1, -16`	`lw $t0, 0($s1)`	1
		`lw $t1, 4($s1)`	2
	`addu $t0, $t0, $s2`	`lw $t2, 8($s1)`	3
	`addu $t1, $t1, $s2`	`lw $t3, 12($s1)`	4
	`addu $t2, $t2, $s2`	`sw $t0, 16($s1)`	5
	`addu $t3, $t3, $s2`	`sw $t1, 12($s1)`	6
		`sw $t2, 8($s1)`	7
	`bne $s1, $0, Loop`	`sw $t3, 4($s1)`	8

What is the cycles per iteration?

What is the IPC?

Loop unrolling provides:

+ fewer instructions that cause hazards (branches)
+ more independent instructions (from different iterations)
+ increase in throughput
- uses **more registers**
- must change **offsets**

Superscalars

Hardware impact:

- more & pipelined functional units
- multiported registers for multiple register access
- multiple decoders
- more hazard detection logic
- more bypass logic, plus buses to the added functional units
- wider instruction fetch

or else the processor has structural hazards (due to an unbalanced design) and stalling

But there are restrictions on instruction types that can be issued together to reduce the amount of hardware.

Static (compiler) scheduling helps

Today’s Superscalars

R10000 & R12000: 4 instructions

Alpha 21064: 2 instructions

Alpha 21164, 21264: 4 instructions

Pentium III (Coppermine): 5 RISClike operations dispatched to functional units

UltraSPARC-1, UltraSPARC-1: 4 instructions

UltraSPARC-3: 6 instructions dispatched