Multiple Instruction Issue

Multiple instructions issued to FUs each cycle

- a processor that can execute more than one instruction per cycle
- **issue width** = the number of instructions the hardware can execute in parallel
- not all types of instructions can be issued together
  - a 4-wide example:
    2 ALUs, 1 load/store unit, 1 FPU --
    1 ALU does shifts & integer multiplies;
    the other executes branches

Multiple Instruction Issue

Performance impact: increase instruction throughput

- execute multiple instructions in parallel, not just overlapped
  - CPI potentially < 1 (.25 if 4-wide)
  - IPC (instructions/cycle) potentially > 1 (4 if 4-wide)
- better functional unit utilization

Multiple Instruction Issue on Superscalars

Requires more capability & more sophisticated design:

- instruction fetch
  - fetch multiple instructions at once
  - prefetch speculatively beyond conditional branches
  - dynamic branch prediction
- instruction issue
  - issue multiple instructions in parallel
  - determine which instruction**s** can be issued next
  - choose which of the ready instructions to issue
- execution
  - execute multiple instructions at once
- instruction commit
  - commit several instructions in fetch order

Multiple Instruction Issue on Superscalars

Hardware impact: duplicate & more complex hardware, potentially longer wires

- more & pipelined functional units
- multi-ported registers for multiple register access
- more buses from the register file to the additional functional units
- multiple decoders
- more hazard detection logic
- more bypass logic
- prefetching logic
- multi-banked L1 data cache

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

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

Static (compiler) scheduling helps.

Multiple Instruction Issue

Software impact: harder code scheduling job for the compiler

- need more independent instructions
- need a good local mix of instructions

Code Scheduling on Superscalars

Original code

```
Loop:   lw R1, 0(R5)
        addu R1, R1, R6
        sw R1, 0(R5)
        addi R5, R5, -4
        bne R5, R0, Loop
```

Code Scheduling on Superscalars

Original code

```
Loop:  lw R1, 0(R5)
       addu R1, R1, R6
       sw R1, 0(R5)
       addi R5, R5, -4
       bne R5, R0, Loop
```

With load-latency-hiding code

```
Loop:  lw R1, 0(s1)
       addi R5, R5, -4
       addu R1, R1, R6
       sw R1, 4(R5)
       bne R5, $0, Loop
```

	ALU/branch instructions	memory instructions	clock cycle
Loop:			1
			2
			3
			4

Code Scheduling on Superscalars: Loop Unrolling

	ALU/branch instruction	Data transfer instruction	clock cycle
Loop:	addi R5, R5, -16	lw R1, 0(R5)	1
		lw R2, 12(R5)	2
	addu R1, R1, R6	lw R3, 8(R5)	3
	addu R2, R2, R6	lw R4, 4(R5)	4
	addu R3, R3, R6	sw R1, 16(R5)	5
	addu R4, R4, R6	sw R2, 12(R5)	6
		sw R3, 8(R5)	7
	bne R5, R0, Loop	sw R4, 4(R5)	8

What is the cycles per iteration?
What is the IPC?

Code Scheduling on Superscalars: Loop Unrolling

Advantages:

+

+

Disadvantages:

-

-

Multiple Instruction Issue

Scheduling instructions

- which instructions are sent to the functional units for execution
- schedule for available FUs & to hide latency

Superscalar processors

- instructions are scheduled for execution ***by the hardware***
- ***different*** numbers of instructions may be issued at once

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

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