Multiple Issue

- Previous techniques - Try to achieve an ideal CPI of 1 by overcoming data and control hazards

- **Multiple issue** - Issue several instructions per clock cycle
- **Goal** - Get CPI below 1 ("IPC" is now our goal!)
- CPI can't get <1 *unless* we issue ***multiple instructions per cycle***

- Two primary architectures: *superscalar* and *VLIW*

Superscalar Processors

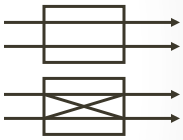
- Fetch multiple instructions per cycle
- Issue varying number of instructions every cycle: 1 - 8 instructions (as dependences permit)
- Must be 1) independent and 2) satisfy resource constraints

- Statically or dynamically scheduled

- Dynamic scheduling using Tomasulo's Algorithm
- Accurate branch prediction

Dual-Issue MIPS

- Let's start with a statically scheduled superscalar
- Suppose we can issue two instructions
  - Instr1: Load, Store, integer ALU operation
  - Instr2: Floating-point operation



- Fetch 64 bits per cycle
- Instructions are paired and aligned to 64 bits
- Could swap - have to inspect instructions
- Second issued only if first is issued (a so-called pipeline scheduling rule - these can get quite complicated!)

Pipeline Operation

Type	Pipe Stages							
Int	IF	ID	EX	MEM	WB			
FP	IF	ID	EX	MEM	WB			
Int		IF	ID	EX	MEM	WB		
FP		IF	ID	EX	MEM	WB		
Int			IF	ID	EX	MEM	WB	
FP			IF	ID	EX	MEM	WB	
Int				IF	ID	EX	MEM	WB
FP				IF	ID	EX	MEM	WB

Two instructions issue together. The floating point instructions may take multiple cycles in the EX stage. Out-of-order completion is possible.

Does the FP stall the integer pipeline???

Statically Scheduled Dual-Issue

- To keep pipeline busy - compiler must find a floating point and integer operation
- This pipeline structure only useful if floating point unit is pipelined (or have multiple of them) - avoids FP becoming the bottleneck
  - E.g., Avg. FP latency 4 with nonpipelined. Then only every 4 cycles can we issue an FP -- utilization is 25%
- Integer and FP means no dependences
- Check for structural hazard - look at the opcodes - minimizes the hazard hardware

Problem for FP load, store, move

- Contention for FP register ports

```
ST      F0,40(R1)
ADDD    F2,F4,F6
```

  - Add another register port
  - Handle structural hazard by scheduling - FP load and stores have to issue by themselves
- Dependence between instructions

```
LD      F0,40(R1)
ADDD    F2,F0,F6
```

  - Check for hazard - only data dependences on a load or int-to-fp move
  - Relatively simple in this case to check

Using Results

- For load, result not available until two cycles later
- Implies - *three* instructions after the load can't use the result (as opposed to *one* instruction when we had single issue!)

Cycle	Instr1	Instr2
0	LD F0	can't use F0
1	can't use F0	can't use F0
2	can use F0	can use F0

- Delayed branches have same problem
  - More slots - harder to fill - need good prediction!

Exploiting Parallelism

- For static scheduling - compiler has to find the parallelism (unroll loops and apply other transformations)
- Hardware scheduling will be complicated
- Instruction decoding also gets complex since we have to check hazards across multiple instructions
- How does unrolling help???

Multi Issue w/Dynamic Scheduling

- Static scheduling can be difficult
  - Usually many rules and constraints that must be observed - limit effectiveness (e.g., no dependent instructions issued in same cycle, need FP and integer rather than integer and integer, etc.)
  - But limited parallelism to begin with!
  - Code scheduled for different pipeline may not run well
- Hardware scheduling comes to the rescue!
  - (well sorta – it has own trade-offs)
  - Dynamically schedule the code based on what the hardware resource can do
- Based on Tomasulo's algorithm

Dual-Issue w/Dynamic Scheduling

- Extend Tomasulo's algorithm for issuing two instruction simultaneously
- Issue to reservation stations in order to simplify table updates
  - Separate data structures for int and FP registers
- Can issue FP and integer together - different register sets
- Can't issue instructions with dependences to reservation stations in same cycle
  - E.g., a FP ADD uses result of an FP load

Achieving Dual-Issue

- Previous issuing limitation - no different than problem faced by compiler
- Desire to issue two dependent instructions to reservation stations together - their execution is serialized at the reservation stations
- Double pump instruction issue - effectively issues both together on same cycle (renaming done in 1/2 cycle)
- Observe issue restrictions - limited dependences between FP and integer (gets complex quickly!)

VLIW Processors

- Statically scheduled by compiler - every cycle we know what will be issued
- VLIW – "Very Long Instruction Word"
- Multiple instructions (or "operations") are scheduled in a single long instruction word.
- Operations execute together - achieving multiple issue and a CPI less than 1
- Operations in same VLIW are (usually) independent

VLIW Processors

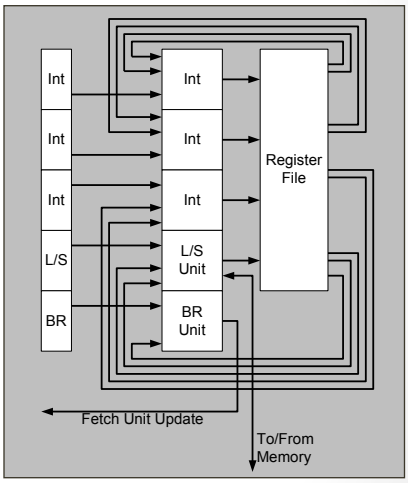
- Compiler finds independent operations that can be scheduled in instruction words
- Historically came from microcoding
- Early machines from Multiflow, Cydrome

- **Main advantage** - less hardware complexity since scheduling decisions made statically
- For high issue widths, scheduling window becomes a bottleneck in superscalars

VLIW Architecture

- Conceptual view of a VLIW

- 3 integer units
- 1 load/store unit
- 1 branch unit



Finding Parallelism for VLIWs

- Trace scheduling - find large sequences of instructions to schedule (remember basic blocks are too small!)
- Predication - flatten the CFG so operations can be scheduled together (can help form long sequences of instructions)
- Software pipelining - form a pipeline in the application code from one loop iteration to a subsequent one

VLIW Architectures

- Main advantage comes at a cost
- Disadvantages include:
  - **Legacy code support** - when issue width and FU mix changes, the code has to be rescheduled
  - **Code growth** - unused operation slots filled with NOPs leads to wasted code space
  - **Instruction bandwidth** - on every cycle, it may be impossible to fill all instruction slots
  - **Window for finding parallelism** - sophisticated code transformations, accurate profile information
  - **Compiler complexity** - the compiler becomes tremendously more complex and must be verified just as the dynamic scheduling hardware has to be verified in superscalars
  - **Fully interconnected FUs** - requires clustered architectures

Superscalar vs. VLIW

Superscalar Processors	**VLIW Processors**
Hardware makes decisions	Compiler makes decisions
Dynamic issue capability	Static issue capability
Complexity in hardware-	Complexity in the compiler
Dynamic knowledge+	Static knowledge (profiling)
Expensive for many FUs-	Cheaper for many FUs+
Legacy code+	Reschedule (not so true now)
Memory disambiguation+	Powerful speculation+
No bookkeeping code+	Program behavior+
Local window-	Code growth-
Most current processors	***IA-64, many DSPs***

Problems with Multiple Issue

- Inherent difficulty of finding enough ILP
- Complexity of the hardware
- VLIW or superscalar limitations

Limits on ILP

- With static scheduling - may need to unroll loops a lot fo find sufficient ILP
- With a 5-wide VLIW and each FU is pipelined, we need to find > 5 independent operations

- With two FPUs & each 5 stages deep. How many independent operations do we need?

  To keep 100% busy, 2*5=10 operations

Needed ILP = average FU pipe depth * number of FUs

Hardware Cost

- Superscalar - dynamic scheduling
- VLIW - issue isn't the problem

We have to keep operations in a VLIW in lock-step through the pipeline due to static scheduling

- Consider cache misses - nondeterministic latencies.
- Schedule assuming no misses, the processor has to stall to ensure the schedule is correct.

Hardware Cost

- Easy to increase number of FUs (just do it!)
- But....

  Register and memory bandwidth requirements increase!!
- We've already seen register bandwidth increases
- **Memory bandwidth** - to keep FUs busy, we need to fetch more data
- With > FUs, then we need > L/S units
- But... these are expensive since they impact the bandwidth needed off the chip to memory