Pipelining and Exceptions

- Exceptions represent another form of control dependence.
- Therefore, they create a potential branch hazard.
- Exceptions must be recognized early enough in the pipeline that subsequent instructions can be flushed before they change any permanent state.
- As long as we do that, everything else works the same as before.
- Exception-handling that always correctly identifies the offending instruction is called precise interrupts.

Pipelining in Today’s Most Advanced Processors

- Not fundamentally different than the techniques we discussed
  - Deeper pipelines
- Pipelining is combined with
  - superscalar execution
  - out-of-order execution
  - VLIW (very-long-instruction-word)
Deeper Pipelines

- How much deeper is productive? What are the limiting effects?
  - Pipeline latching overhead
  - Losses due to stalls and hazards
  - Clock Speeds achievable

Instruction Level Parallelism (ILP)

- Definition: “Exploitation of parallelism across instructions of a program in order to achieve high performance.”
  - Focused on pipelining, one instruction issued at a time.
  - Many instructions in flight (multiple cycle latencies)
  - Static, dynamic pipelining techniques, ILP memories
  - All of these systems are subject to the “Flynn Bottleneck” or issue bottleneck. IP rate cannot exceed the clock rate
  - High clock rate systems can be very difficult to design, limited by circuit technology

- Solution: Multiple issue machines!
Instruction Level Parallelism (cont)

- Pipelined machines thus far
- Multiple issue machines issue several instructions to functional units in a **single clock cycle**.
- Of course, these machines are pipelined as well, but we'll focus on the multiple issue part...

Superscalar Processors

- Issue several instructions simultaneously from a single instruction stream.
- Hardware detects if they're independent, then issues them to the appropriate functional unit.
- Each cycle, 1 to n instructions "issued" for execution.
- How does hardware decide?
Superscalar Issue

Sequential instruction semantics define allowable issue order (+ hardware constraints)
Issue restrained by data hazards, control hazards, and structural hazards.
What about two dependent instructions?
  – Some aggressive implementations allow issue, but others don’t (clock period vs. CPI)
What about branches? (more about this later)
What operations can be executed in parallel?

Superscalar Execution

- To execute four instructions in the same cycle, we must find four independent instructions.
- If the four instructions fetched are guaranteed by the compiler to be independent, this is a VLIW machine.
- If the four instructions fetched are only executed together if hardware confirms that they are independent, this is an in-order superscalar processor.
- If the hardware actively finds four (not necessarily consecutive) instructions that are independent, this is an out-of-order superscalar processor.
Superscalar Scheduling

- Assume in-order, 2-issue, ld-store followed by integer
  
  \[
  \text{lw } \$6, 36(\$2) \\
  \text{add } \$5, \$6, \$4 \\
  \text{lw } \$7, 1000(\$5) \\
  \text{sub } \$9, \$12, \$5 \\
  \]

- Assume 4-issue, any combination (VLIW?)
  
  \[
  \text{lw } \$6, 36(\$2) \\
  \text{add } \$5, \$6, \$4 \\
  \text{lw } \$7, 1000(\$5) \\
  \text{sub } \$9, \$12, \$5 \\
  \text{sw } \$5, 200(\$6) \\
  \text{add } \$3, \$9, \$9 \\
  \text{and } \$11, \$7, \$6 \\
  \]

Superscalar vs. Superpipelined

- Multiple instructions in the same stage, same clock rate as scalar

- More total stages, faster clock rate
Implications of Superscalar Execution

Pros:
- achieve even higher levels of ILP without increasing clock rate (2-issue, 4-issue, 8-issue, ?-issue)
- sequential source compatibility (same binaries as single issue machines and X-issue machines for all X)

Con:
- detecting dependences at runtime make issue decision difficult, how many opcodes and register addresses must be compared at the same time?
- hardware issue and alignment restrictions must be enforced
- aggressive ILP requires LOTS of hardware, superscalar issue is on top of this...

How complex is this decision?

Sample Issue Restrictions: DEC Alpha 21064 (circa 1992)

- Two instructions must be contained within an aligned quadword (no odd pair alignment)
- Both instructions must not be in class IB0 or IB1 (see table in programmer’s manual)
- Data dependence restrictions
- Complex structural hazard restrictions
- How complex is the issue unit? Is there any alternative?
Very Long Instruction Word (VLIW)

- Alternative approach to ILP, explicit “issue packets”
  - Explicitly Parallel Instruction Computing (EPIC, IA-64)
- “Smart compiler and a dumb machine.” -- Fisher
- Idea: compiler must already analyze and schedule for high performance, why not “explicitly specify” the issue packets
- Various tricks can be used to reduce storage overhead for NOPs; they correspond to stall cycles in the superscalar

```
add r2, r3, r4  lw  r5, 0(r6) #INST 0
sub  r5, r7, r9  nop #INST 1
mul  r10, r5, r8  add r2, r3, r5 #INST 2
beq  r2, r0, label10 nop #INST 3
...
add r2, r3, r4  lw  r5,0(r6)
sub  r5, r7, r9  nop
mul  r10, r5, r8  add r2, r3, r5
beq  r2, r0, label10
...
```

Register dependences

```
add  r2, r3, r4  sub  r4, r3, r2  #INST0
...  ...
lw  r5,32(r9)  add  r9, r5, r5  #INST25
```

- VLIW’s don’t have dependences between instructions in a single issue packet
- all of the above cases; instructions don’t use the same data...
- => more efficient use of the register namespace
VLIW Analysis

- **Pros:**
  - Simple hardware; no critical path dependence test and alignment

- **Cons:**
  - Embedding of dependence structure prevents use of single binary format => need to distribute binaries for each processor design or even system configuration.
  - Intel EPIC takes a compromise position, explicitly specifying parallelism within an issue packet, but not between them (relying on HW).

Superscalars and VLIWs

- **Marketplace has chosen Superscalars... mostly.**
  - 4-way: P6, Ultrasparc, R10000, PowerPC 620
  - 8-way? These are getting harder to build... and returns are diminishing...

- **VLIW Machines have been built.**
  - Multiflow (Fisher, Colwell), Cydrome (Rau, others), i860 (Intel)
  - Dynamic translation is lowering the software compatibility stumbling block.
  - This is a key feature in Merced IA-64 implementation, enables 8-way machines(?)
Implications of Multiple issue

- Increased ILP further stresses instruction fetch and data memory systems
- Greater hardware functional unit concurrency is required; perhaps complex hardware for hazard detection
- Dramatic increases in register bandwidth (and perhaps register names) for the many instructions in flight
  - 4-issue needs how many read ports? write ports?
- Aggressive compilation needed to expose enough ILP to make the machines work well (we’ll talk about this later)
- Impact of control hazards is MUCH worse (loss in issue slots, not cycles)

Dynamic or Out-of-Order Scheduling

- Issues (begins execution of) an instruction as soon as all of its dependences are satisfied, even if prior instructions are stalled.
  - lw $6, 36($2)
  - add $5, $6, $4
  - lw $7, 1000($5)
  - sub $9, $12, $5
  - sw $5, 200($6)
  - add $3, $9, $9
  - and $11, $7, $6
Reservation Stations

- are a mechanism to allow dynamic scheduling

![Diagram of Reservation Stations](image)

PowerPC 604
Pentium II

Po
Dynamic Pipelining

- Instruction issue enforced the value relationships which maintain sequential execution order semantics.
- Results are written to the register file in order
  - Instructions are “retired” in order
  - Values in the meantime come thru the result bus
- So, what really determines the order of the execution of the operations?
- Imagine doing this with precise exceptions.... !

Pipelining Summary

- ET = Number of instructions * CPI * cycle time
- Data hazards and branch hazards prevent CPI from reaching 1.0, but forwarding and branch prediction get it pretty close.
- Data hazards and branch hazards need to be detected by hardware.
- Pipeline control uses combinational logic. All data and control signals move together through the pipeline.
- Pipelining attempts to get CPI close to 1. To improve performance we must reduce CT (superpipelining) or CPI below one (superscalar, VLIW).
Pentium III

- 10-stage pipeline
- Translates up to 3 insts/cycle into micro-ops
- Executes up to 6 micro-ops per cycle, but can only retire 3/cycle.
- L1 instruction and data caches both 16K, 4-way
- L2 cache on-chip, 256K, 2-way
- 32-entry Instruction and 64-entry data TLBs
- Out-of-order scheduling
Pentium IV

- 20-stage pipeline
- ALUs clocked at twice the chip clock rate
- Out-of-order scheduling
- Peak execution throughput 6 instructions/cycle.
- L1 Data cache 8KB, L2 unified cache 256 KB
- L1 Instruction cache is a trace cache, holding decoded instructions in execution order.
- Up to 126 micro-ops in-flight.

Intel Itanium (IA-64)

- Not x86 ISA
- EPIC architecture (Explicitly Parallel Instruction Computing)
- 3-instruction VLIW bundles.
- 41-bit instructions (3 instructions, plus 5 extra bits, fit in 128 bit bundle)
- Can execute up to 2 bundles/cycle.
- 10-stage pipeline
- In-order execution
- Predicated instructions add $4, $5, $6 [sp1]
- L1 data caches both 16KB, 2-way, 32-byte lines
- L2 onchip, 96 KB, 6-way, 64-byte lines
- 128 floating point, 128 integer, 64 predication, and 8 branch registers.
AMD Athlon

- x86 compatible
- Translates up to 3 insts/cycle into micro-ops
- Executes up to 9 micro-ops per cycle, but can only retire 6/cycle.
- L1 caches both 64K, 2-way
- L2 cache off-chip, 2-way, size variable
- Instruction and data TLBs, 256-entry
- Out-of-order scheduling

Alpha 21364

- 64KB, 2-way instruction and data L1 caches
- 1.5 MB, 6-way on-chip L2 cache
- Victim caches
- 7-stage pipeline
- Out-of-order execution
- 80 instructions in-flight
- 4 instructions/cycle peak sustained throughput