

# Microprocessors

VLIW

Very Long Instruction Word Computing  
April 18th, 2002

# The Background

- Classical CISC
  - Instructions fairly short, basically do one thing at a time. No instruction parallelism.
- Classical RISC
  - Instructions fairly short, basically do one thing at a time. Processor introduces instruction parallelism
- VLIW
  - Instructions much longer, do several things at the same time (several separate operations).
  - All these operations are done in parallel

# Typical VLIW Machine

- Instruction word is a long packet
  - Which is broken down into operations
  - Each operation is like a conventional microprocessor instruction
- An example, the Multiflow machine
  - Operations are conventional
  - But can have 4 or 7 operations/instruction
  - Instruction length is 128 ( $32 \times 4$ ) or 224 ( $32 \times 7$ )
  - The 4 or 7 instructions are executed in parallel
  - Programmer must be aware of parallelism

# Multiple Operations

- An instruction contains multiple ops
  - For example, instruction might have
    - $R1 \leftarrow R2$ ,  $R3 \leftarrow R4 + R5$ , Jump L3,  $R6 \leftarrow R7$
    - Here we do the three assignments
    - Then the next instruction comes from L3
    - Can only have one effective jump in a group
    - And the operations need to be independent
    - Since they will be executed in parallel

# The Notion of Independence

- Suppose we have two operations in the same instruction:
  - $R1 \leftarrow R2 + R3, R4 \leftarrow R1 + R7$
  - This is not wrong, but these instructions are executed together at the same time
    - Not sequentially
    - Not “as if” sequentially
  - So second operation uses old value of R1
  - Ordering of operations in instruction makes no difference to the result of the instruction.

# More on Independence

- Consider these two operations in a single instruction:
  - $R1 \leftarrow R2, R2 \leftarrow R1$
  - This is OK, results in an exchange, since loads done from old register values, independent stores to new register values.
  - Program (really compiler) has to be VERY aware of grouping of operations into instructions (unlike classical RISC)

# Latency and Interlocks

- For classical RISC, we usually use interlocks.
  - This means that if we try to reference a register before the result is ready, the pipeline stalls till the result is ready.
  - Gives the illusion of strictly sequential execution.
  - Some attempts early on (MIPS Load delay) to avoid this assumption, but since abandoned.

# More on the MIPS Load Delay

- In the original MIPS
  - If you load an integer register
  - Then you cannot reference the result in the next instruction, must wait one instruction.
  - This is called a load delay slot
- Abandoned in later MIPS designs
  - Which use full interlocking

# More on MIPS Interlocking

- Why did MIPS change their tune
  - After all MIPS originally stood for something like Microprocessor without interlocking pipeline stages
  - Because new implementations (with different memory latencies) would have required more than one slot and we don't like correctness of code being dependent on the version of the implementation.
  - Because other instructions required interlocking anyway (e.g. floating-point)
  - Because it is not that painful to do interlocking

# Back to VLIW: Interlocking

- Now that we have 7 separate operations at the same time, what about interlocking
- We could implement interlocking but
  - The 7 streams are strictly synchronized, unlike separate pipelines in a RISC design, so an interlock would hold up all 7 streams
  - The logic for this kind of multiple interlocking would be more complex
- So VLIW systems typically do NOT have any interlocking, and instead use original MIPS approach.

# VLIW Latency Considerations

- Each operation has a known latency
- For example, a floating-point add might have a latency of 2, meaning that you have to wait two instruction times before using the result of the operation
- The programmer (really the compiler) must know all latencies and promise not to access anything too early
- Just like the original MIPS, but for all operations and compiler must keep track of multiple in flight instructions.
- No checks at runtime if you violate the rules

# What about Loads?

- Typical pattern for a load is
  - 2 clocks if in primary cache
  - 5 clocks if in secondary cache
  - 50 clocks if in main memory
- Can't assume the worst all the time
- Assume the best (or at least not the worst) and then if we are unlucky stall the entire pipeline (and all operations)

# VLIW: The Advantages

- There is really only one
- Greater speed
  - We are sure of executing 7 operations on each clock
  - And clock speed can be high, since no complex on the fly scheduling, and no complex interlocking, just simple operations with no checking
  - Easy to build, no problem in providing 7 sets of gates on the chip that operate entirely independently
- VLIW: Path to future performance gains?

# VLIW: The Disadvantages

- We need 7 independent operations for each instruction
  - Can we find them?
  - Not easily
  - We may end up with lots of NOPS
- Complex latency assumptions
  - Means that code depends on exact model of chip, since latencies change with new chips.

# Finding Parallelism

- How shall we fill the operations with good stuff, i.e. useful work
  - We are not interested in NOPS ☹
- Clever compilers look far ahead and rearrange the program to increase parallelism (trace scheduling)
- Speculative Execution

# Trace Scheduling

- Normally scheduling of instructions happens only in a basic block
  - A basic block is a sequence of instructions with no branches
- Trace Scheduling
  - Extends the search for a thread of instructions over many basic blocks
  - Gives a better chance of finding independent operations.

# Speculative Execution

- We might as well do something rather than NOPs, since it costs nothing extra in time.
- So if we can't do anything better, execute operations that might be useful.
  - If they turn out to be useful great
  - If not, well we didn't waste any time (not quite true as it turns out)

# Control Speculation

- Conditional Branches are the Enemy ☹
- Because we don't know which way to go
- So go both ways at once
- Compute two sets of results in different registers independently
- Then throw away the set that is useless
- Better to do something that *might* be useful rather than nothing at all.

# More on Control Speculation

- What if there are memory operations involved.
- Not so easy to speculate
- If we can predict the jump direction:
  - Then it may be worth doing stuff that likely will be OK
  - But will need undoing (fixup) code in the (hopefully unlikely) case that we chose the wrong way to go.

# The Load/Store Problem

- We like to do loads early
  - Because they have long latency
- We like to do stores late
  - Because that gives time for computing the results we need to store
- So in general we would like to move loads up and stores down
  - Which means we would like to move loads past stores where possible

# More on the Load/Store Problem

- If we have
  - Load from local variable A
  - Store to local variable B
- Then we can safely move:
  - A load from B
  - Past a Store to A
  - Since these are obviously independent

# More on the Load/Store Problem

- But what if we have
  - ... := A(J)
  - A(K) := ...
- Or a similar case with pointers
  - ... = \*q;
  - \*p = ...
- Likely we can do the move of the store past the load, but we don't know for sure
  - Since J may equal K
  - Or p and q may point to same variable

# Data Speculation

- Data Speculation addresses the subscript/pointer case where we don't know if we can move a load past a store.
- Assume we can do the move
- Proceed ahead with the assumption that the move was OK
- Check later on that it was OK
  - And if not, execute fixup code

# VLIW: The Past

- Two important commercial attempts
  - Multi-flow (a VLIW version of the VAX)
  - Cydrome
- Both companies went bankrupt ☹
  - Among the problems were performance
    - Too hard to find parallelism
    - Compilers not clever enough
    - Too much fixup code

# VLIW: The Future

- The Cydrome folk ended up at HP Labs
- They continued to develop their ideas
- Intel-HP followed up on their ideas
- And voila! The ia64 architecture, which borrows on these ideas.
- But tries to solve problems (EPIC)
- Will that be successful
  - Stay tuned for next exciting episode.