

Portland State University
ECE 587/687

Very Long Instruction Word (VLIW) Architectures

© Copyright by Alaa Alameldeen and Haitham Akkary 2009

Very Long Instruction Word Architectures

- Compiler extracts parallelism
 - ◆ Static scheduling
 - ◆ Trace scheduling
- Multiple operations to various execution units grouped together in one very long instruction
- Compiler schedule based on pre-knowledge of hardware execution units and latencies
- Simple hardware
 - ◆ No dependence checks in hardware

Portland State University – ECE 587/687 – Spring 2009 2

Trace Scheduling

- Profiling to identify most frequent traces
- Compiler schedules as if trace is one big basic block
 - ◆ Moves instructions across branches
 - ◆ Moves loads above earlier stores
 - ◆ Compensation code for early exits from trace
 - ◆ NOPs inserted when slots cannot be filled with parallel operations
 - ◆ Compiler schedules for a specific hardware implementation
- Repeat scheduling the next most frequent trace, including compensation code
- See figures in paper p. 265, 266

Portland State University – ECE 587/687 – Spring 2009 3

The ELI-512

- Has 16 clusters,
 - ◆ Each containing an ALU and some storage
 - ◆ Arranged circularly with some extra communication links
- 500-bit instruction word to initiate the following in each instruction cycle:
 - ◆ 16 ALU operations: 8 32-bit integer, 8 use 64-bit ALUs (including floating point)
 - ◆ 8 pipelined memory references
 - ◆ 32 register accesses
 - ◆ Many data movements
 - ◆ A multi-way conditional jump based on several independent tests
- See top figure in paper p. 268

Portland State University – ECE 587/687 – Spring 2009 4

Some VLIW Requirements

- Difference between VLIWs and vector architectures (discuss)
- VLIWs need clever Jump mechanisms
 - ◆ N tests, N+1 jump destinations
 - ◆ Similar to C's switch statement
- VLIW compilers must predict memory banks

Portland State University – ECE 587/687 – Spring 2009 5

VLIW Issues

- Backward compatibility
 - ◆ Requires recompilation for new hardware
- Low code density
 - ◆ NOPs and compensation code
- Compiler cannot anticipate dynamic events or account for variable execution latencies
 - ◆ Cache misses
 - ◆ Memory disambiguation
 - ◆ Branch outcomes

Portland State University – ECE 587/687 – Spring 2009 6

EPIC and the Itanium

- Solves some of the issues in VLIW
 - ◆ Compiler provides register dependence information and hardware schedules accordingly
 - Eliminates NOPS
 - Provides backward compatibility
 - ◆ Provides hardware mechanisms for events that cannot be predicted by the compiler
 - Predication for branches
 - Control speculation
 - Data speculation
- Go to Itanium Slides

Reading Assignment

- Dean Tullsen et al., "Simultaneous Multithreading: Maximizing On-Chip Parallelism," ISCA 1995 (Review)
- Dean Tullsen et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," ISCA 1996 (Skim)