Digital Design & Computer Arch.

Lecture 18a: VLIW

Prof. Onur Mutlu

ETH Zürich

Spring 2020

30 April 2020

Approaches to (Instruction-Level) Concurrency

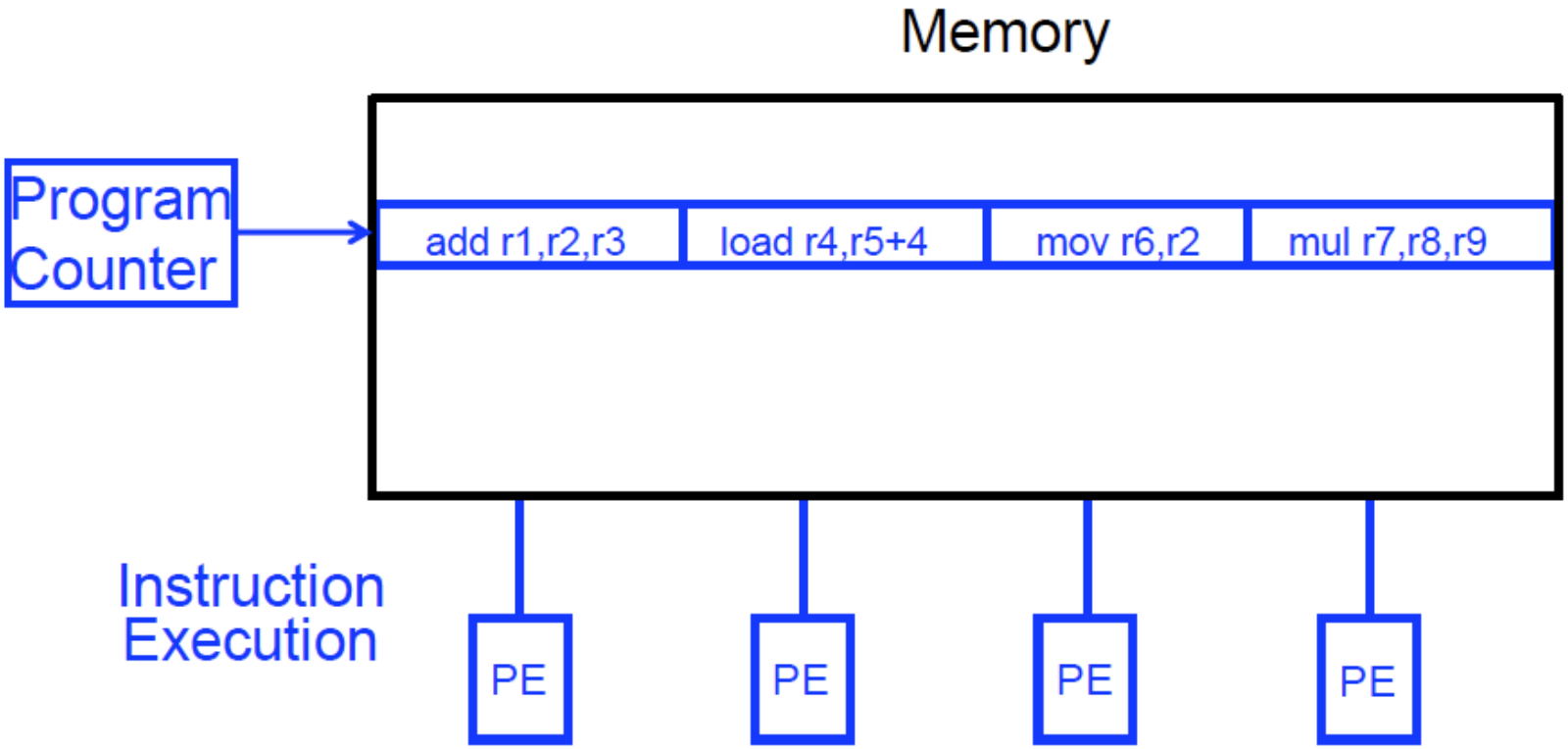
- Pipelining
- Out-of-order execution
- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- Fine-Grained Multithreading
- SIMD Processing (Vector and array processors, GPUs)

VLIW

VLIW Concept

- Superscalar
  - Hardware fetches multiple instructions and checks dependencies between them
- VLIW (Very Long Instruction Word)
  - Software (compiler) packs independent instructions in a larger "instruction bundle" to be fetched and executed concurrently
  - Hardware fetches and executes the instructions in the bundle concurrently
- No need for hardware dependency checking between concurrently-fetched instructions in the VLIW model

VLIW Concept



- Fisher, "Very Long Instruction Word architectures and the ELI-512," ISCA 1983.
  - ELI: Enormously longword instructions (512 bits)

VLIW (Very Long Instruction Word)

- A very long instruction word consists of multiple independent instructions packed together by the compiler
  - Packed instructions can be logically unrelated (contrast with SIMD/vector processors, which we will see soon)

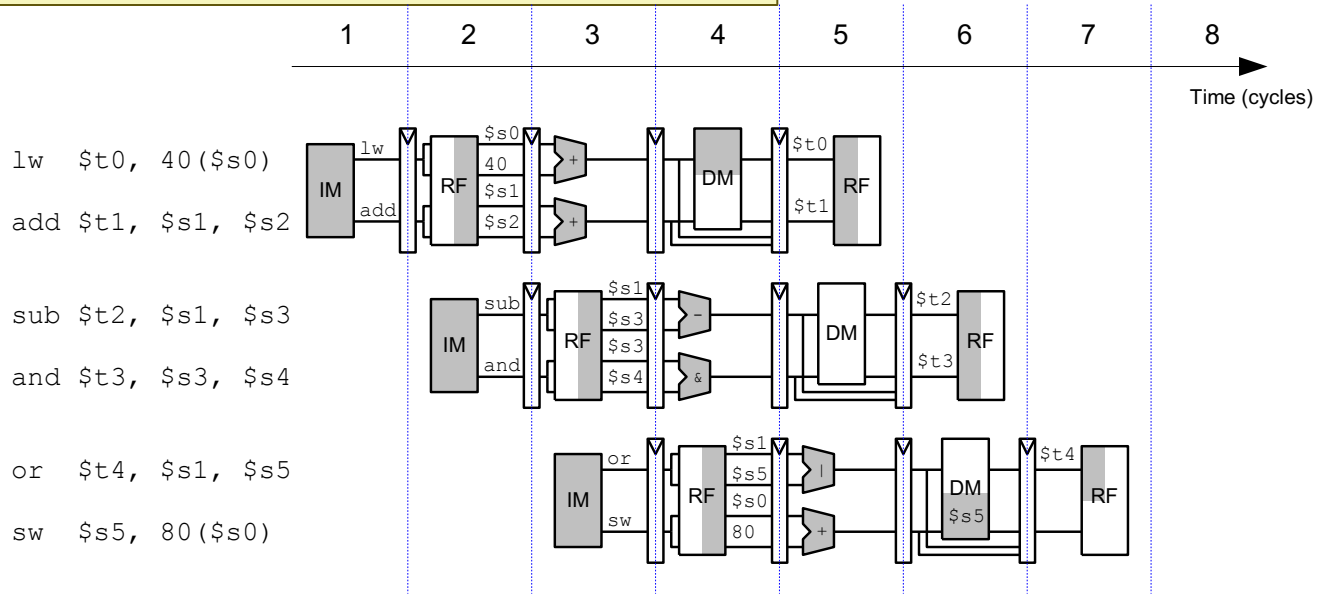
- Idea: Compiler finds independent instructions and statically schedules (i.e. packs/bundles) them into a single VLIW instruction

- Traditional Characteristics
  - Multiple functional units
  - All instructions in a bundle are executed in lock step
  - Instructions in a bundle statically aligned to be directly fed into the functional units

VLIW Performance Example (2-wide bundles)

```
lw  $t0, 40($s0)
add $t1, $s1, $s2
sub $t2, $s1, $s3
and $t3, $s3, $s4
or  $t4, $s1, $s5
sw  $s5, 80($s0)
```

Ideal IPC = 2



Actual IPC = 2 (6 instructions issued in 3 cycles)

VLIW Lock-Step Execution

- Lock-step (all or none) execution: If any operation in a VLIW instruction stalls, all instructions stall

- In a truly VLIW machine, the compiler handles all dependency-related stalls, hardware does **not** perform dependency checking
  - What about variable latency operations?

VLIW Philosophy

- Philosophy similar to RISC (simple instructions and hardware)
  - Except multiple instructions in parallel

- RISC (John Cocke, 1970s, IBM 801 minicomputer)
  - Compiler does the hard work to translate high-level language code to simple instructions (John Cocke: control signals)
    - And, to reorder simple instructions for high performance
  - Hardware does little translation/decoding → very simple

- VLIW (Josh Fisher, ISCA 1983)
  - Compiler does the hard work to find instruction level parallelism
  - Hardware stays as simple and streamlined as possible
    - Executes each instruction in a bundle in lock step
    - Simple → higher frequency, easier to design

Commercial VLIW Machines

- Multiflow TRACE, Josh Fisher (7-wide, 28-wide)
- Cydrome Cydra 5, Bob Rau
- Transmeta Crusoe: x86 binary-translated into internal VLIW
- TI C6000, Trimedia, STMicro (DSP & embedded processors)
  - Most successful commercially

- Intel IA-64
  - Not fully VLIW, but based on VLIW principles
  - EPIC (Explicitly Parallel Instruction Computing)
  - Instruction bundles can have dependent instructions
  - A few bits in the instruction format specify explicitly which instructions in the bundle are dependent on which other ones

VLIW Tradeoffs

- Advantages
  - + No need for dynamic scheduling hardware → simple hardware
  - + No need for dependency checking within a VLIW instruction → simple hardware for multiple instruction issue + no renaming
  - + No need for instruction alignment/distribution after fetch to different functional units → simple hardware

- Disadvantages
  - -- Compiler needs to find N independent operations per cycle
    - -- If it cannot, inserts NOPs in a VLIW instruction
    - -- Parallelism loss AND code size increase
  - -- Recompilation required when execution width (N), instruction latencies, functional units change (Unlike superscalar processing)
  - -- Lockstep execution causes independent operations to stall
    - -- No instruction can progress until the longest-latency instruction completes

VLIW Summary

- VLIW simplifies hardware, but requires complex compiler techniques
- Solely-compiler approach of VLIW has several downsides that reduce performance
  - -- Too many NOPs (not enough parallelism discovered)
  - -- Static schedule intimately tied to microarchitecture
    - -- Code optimized for one generation performs poorly for next
  - -- No tolerance for variable or long-latency operations (lock step)

++ Most compiler optimizations developed for VLIW employed in optimizing compilers (for superscalar compilation)

- Enable code optimizations

++ VLIW successful when parallelism is easier to find by the compiler (traditionally embedded markets, DSPs)

An Example Work: Superblock

The Superblock: An Effective Technique

for VLIW and Superscalar Compilation

Wen-mei W. Hwu Scott A. Mahlke William Y. Chen Pohua P. Chang

Nancy J. Warter Roger A. Bringmann Roland G. Ouellette Richard E. Hank

Tokuzo Kiyohara Grant E. Haab John G. Holm Daniel M. Lavery *

Hwu et al., The superblock: An effective technique for VLIW and superscalar compilation. The Journal of Supercomputing, 1993.

- Lecture Video on Static Instruction Scheduling
  - https://www.youtube.com/watch?v=isBEVkIjgGA

Another Example Work: IMPACT

IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors

Pohua P. Chang *Scott A. Mahlke* *William Y. Chen* *Nancy J. Warter* *Wen-mei W. Hwu*

Center for Reliable and High-Performance Computing
University of Illinois
Urbana, IL 61801

The performance of multiple-instruction-issue processors can be severely limited by the compiler's ability to generate efficient code for concurrent hardware. In the IMPACT project, we have developed IMPACT-I, a highly optimizing C compiler to exploit instruction level concurrency. The optimization capabilities of the IMPACT-I C compiler are summarized in this paper. Using the IMPACT-I C compiler, we ran experiments to analyze the performance of multiple-instruction-issue processors executing some important non-numerical programs. The multiple-instruction-issue processors achieve solid speedup over high-performance single-instruction-issue processors.

Chang et al., IMPACT: an architectural framework for multiple-instruction-issue processors. ISCA 1991.

Digital Design & Computer Arch.

Lecture 18a: VLIW

Prof. Onur Mutlu

ETH Zürich

Spring 2020

30 April 2020