18-747 Lecture 19: Very Long Instruction Word Architectures

James C. Hoe
Dept of ECE, CMU
November 7, 2001

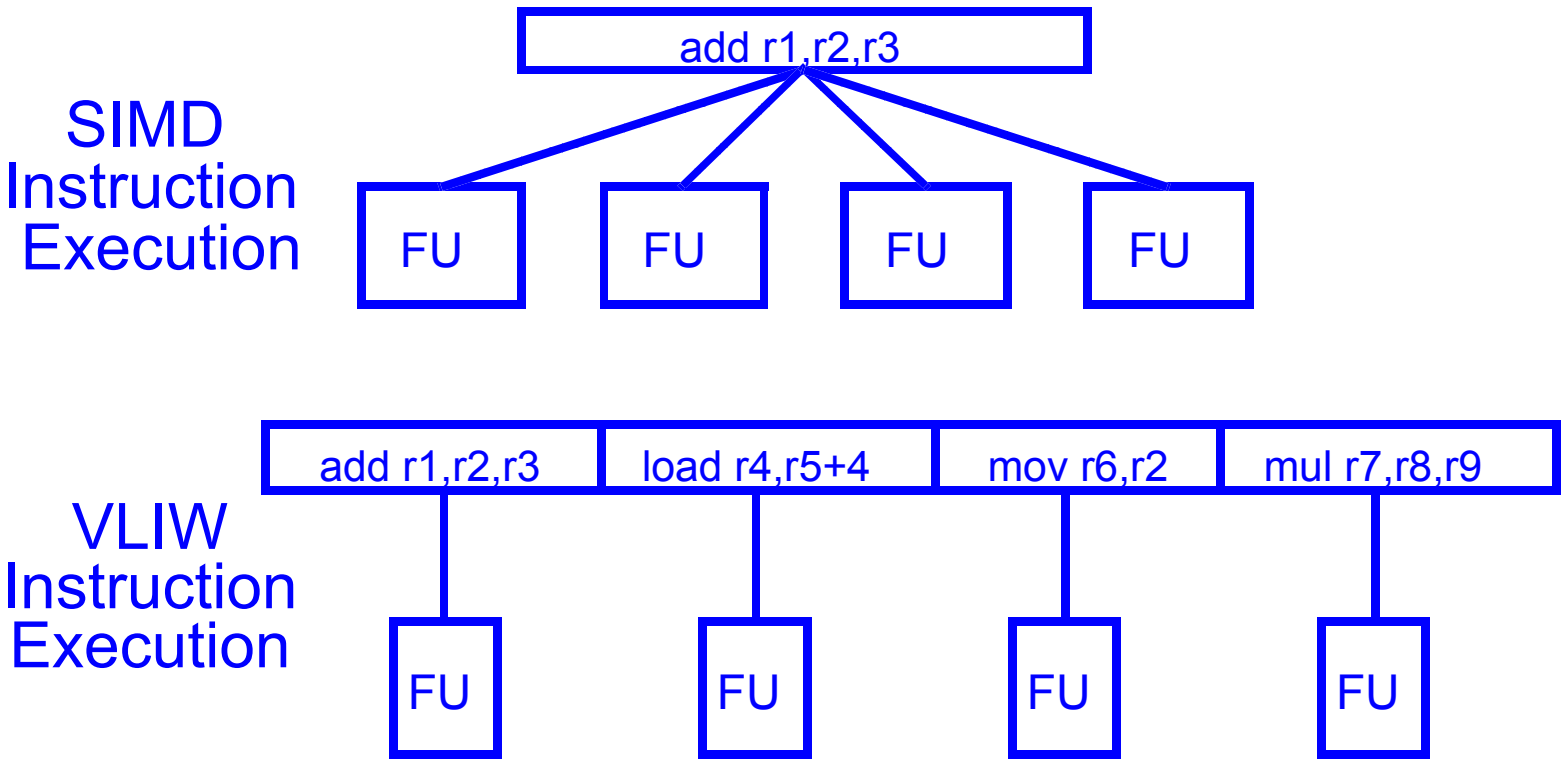
Reading Assignments: *papers below*

Announcements: Project 2 (and HW3) due on coming Friday (coming Monday)

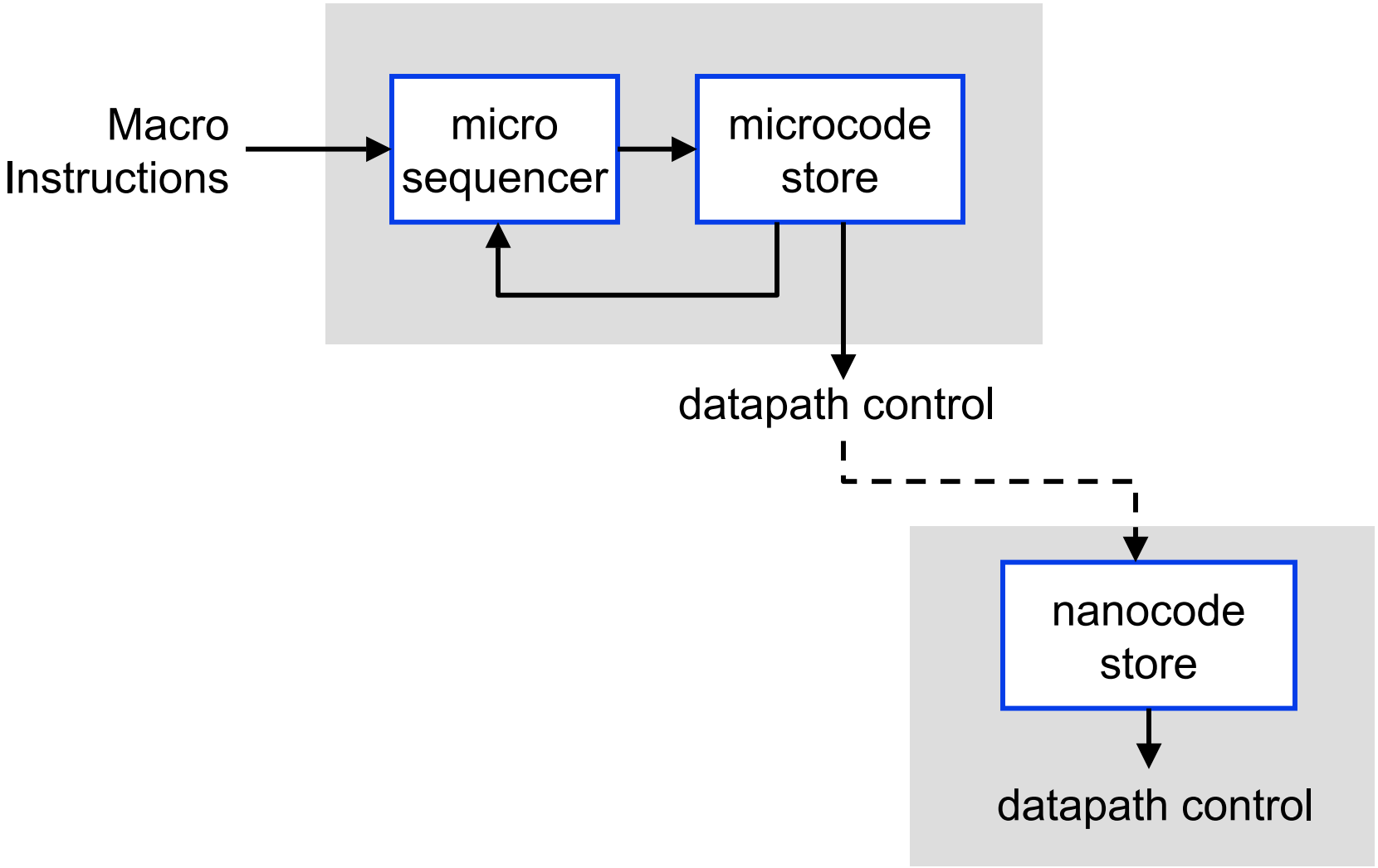
Handouts:
Itanium Processor Microarchitecture, Sharangpani & Arora
The Technology Behind Crusoe Processors, Transmeta Corp.
The Intel IA-64 Compiler Code Generator, Bharadwaj, et al.
Continuous Program Optimization: Design and Evaluation, Kistler & Franz

What Is VLIW?

- VLIW hardware is simple and straightforward, like SIMD machines.
- While SIMD broadcasts one instruction, VLIW separately directs each functional unit

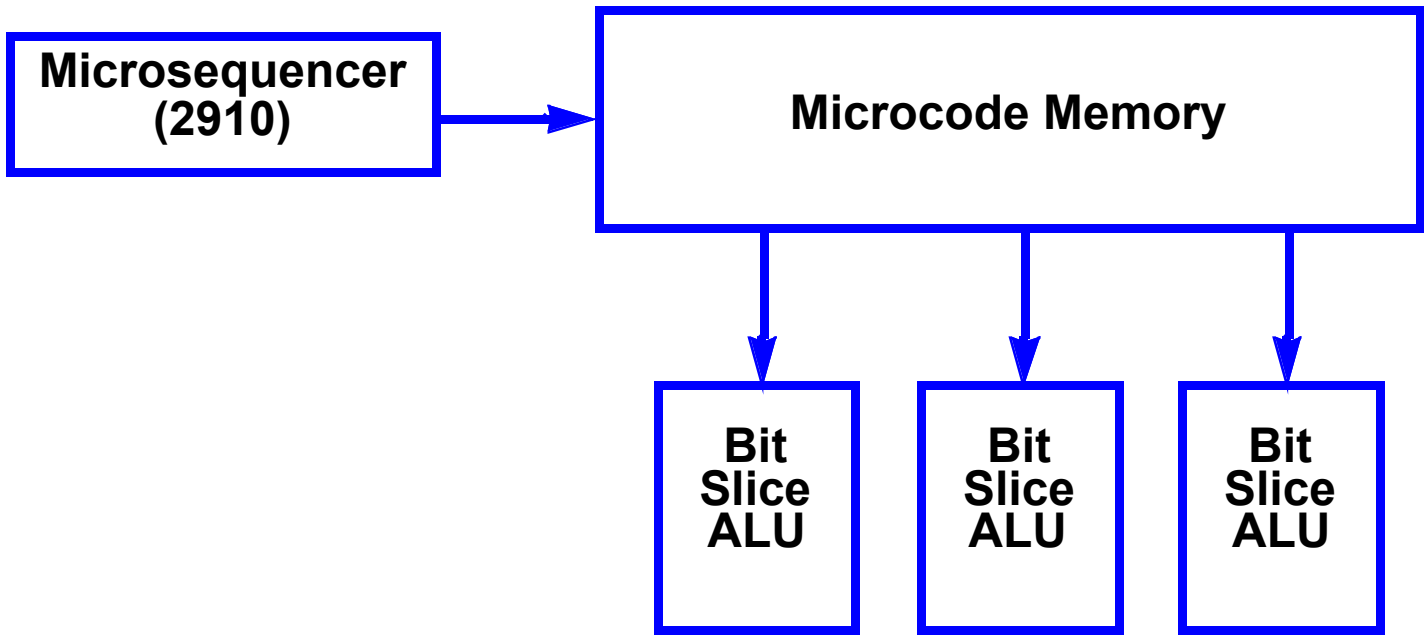


Historical Perspective: Microcoding, nanocoding (and RISC)



Horizontal Microcode and VLIW

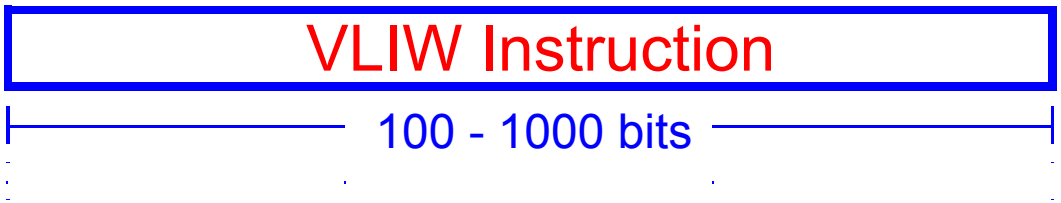
- A generation of high-performance, application-specific computers relied on *horizontally* microprogrammed computing engines.



- Aggressive (but tedious) hand programming at the microcode level provided performance well above sequential processors.

Principles of VLIW Operation

- Statically scheduled ILP architecture.
- Wide instructions specify many independent simple operations.



- Multiple functional units executes all of the operations in an instruction concurrently, providing fine-grain parallelism within each instruction
- Instructions directly control the hardware with no interpretation and minimal decoding.
- A powerful optimizing compiler is responsible for locating and extracting ILP from the program and for scheduling operations to exploit the available parallel resources

The processor does not make any run-time control decisions below the program level

Formal VLIW Models

- Josh Fisher proposed the first VLIW machine at Yale (1983)
- Fisher's *Trace Scheduling* algorithm for microcode compaction could exploit more ILP than any existing processor could provide.
- The ELI-512 was to provide massive resources to a single instruction stream
  - 16 processing clusters- multiple functional units/cluster.
  - partial crossbar interconnect.
  - multiple memory banks.
  - attached processor – no I/O, no operating system.
- Later VLIW models became increasingly more regular
  - Compiler complexity was a greater issue than originally envisioned

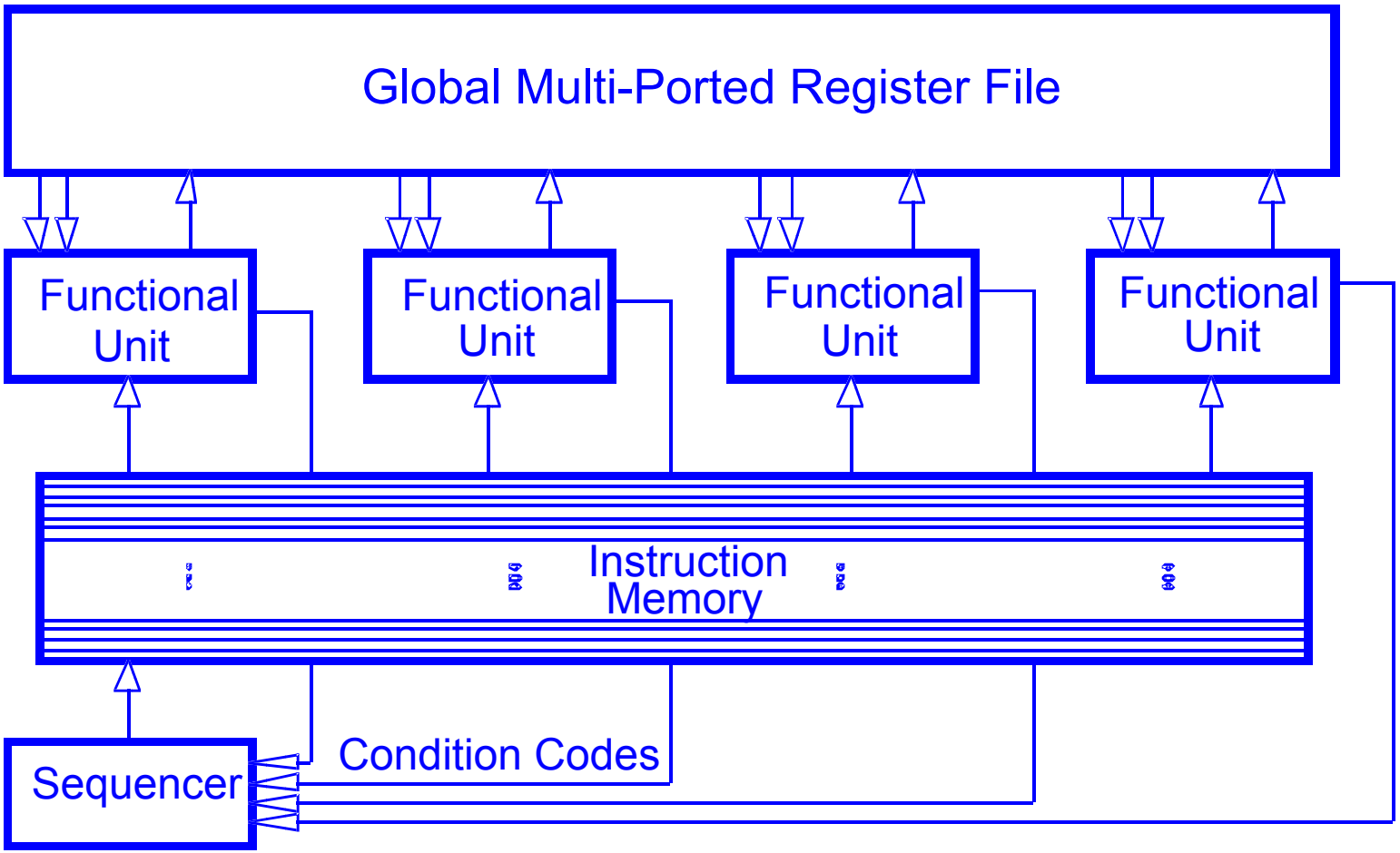
Ideal Models for VLIW Machines

- Almost all VLIW research has been based upon an ideal processor model.
- This is primarily motivated by compiler algorithm developers to simplify scheduling algorithms and compiler data structures.
  - This model includes:
    - Multiple universal functional units
    - Single-cycle global register file

    and often:
    - Single-cycle execution
    - Unrestricted, Multi-ported memory
    - Multi-way branching

    and sometimes:
    - Unlimited resources (Functional units, registers, etc.)

VLIW Execution Characteristics



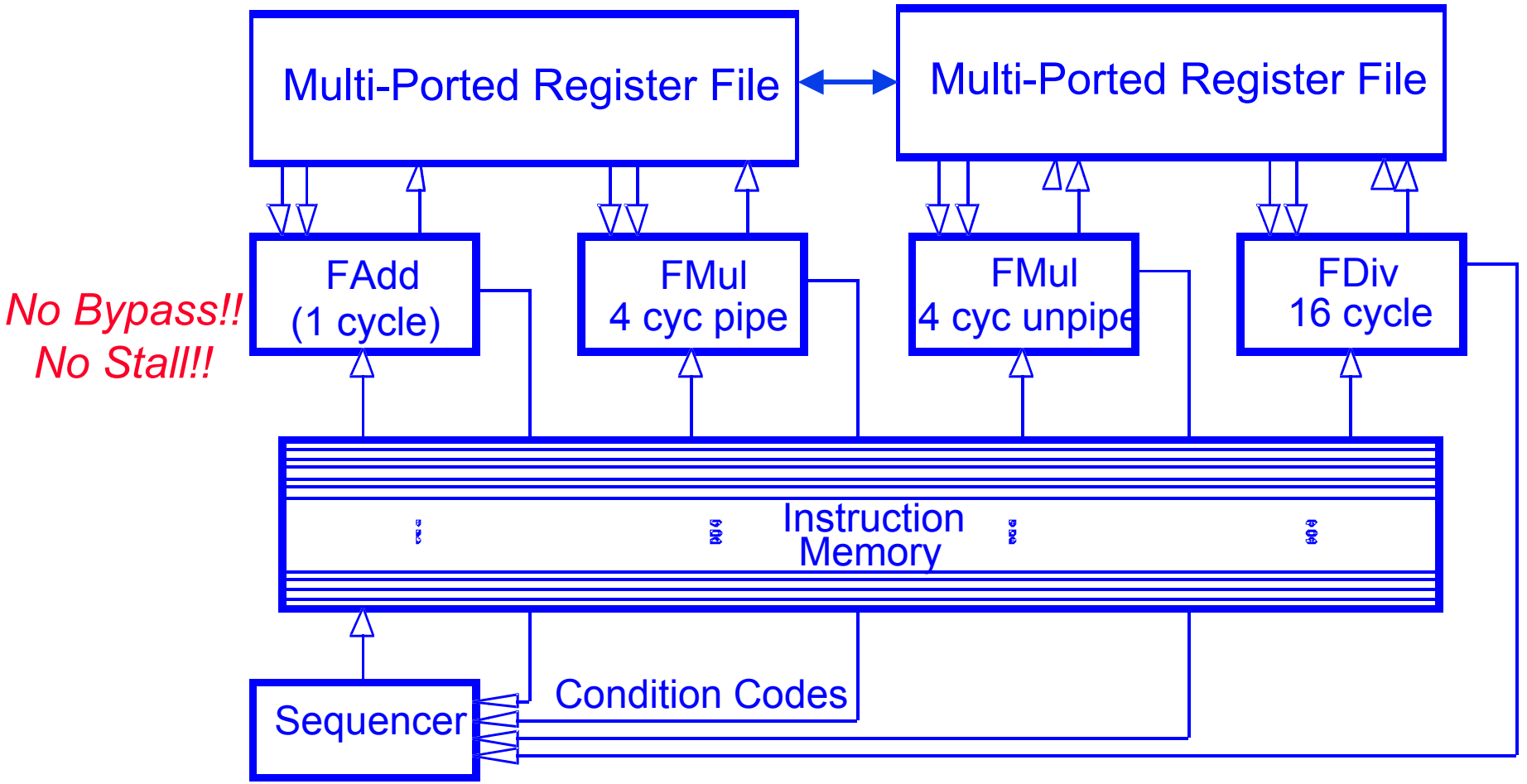
Basic VLIW architectures are a generalized form of horizontally microprogrammed machines

VLIW Design Issues

- Unresolved design issues
  - The best functional unit mix
  - Register file and interconnect topology
  - Memory system design
  - Best instruction format
- Many questions could be answered through experimental research
  - Difficult - needs effective retargetable compilers
- Compatibility issues still limit interest in general-purpose VLIW technology

However, VLIW may be the only way to build 8-16 operation/cycle machines.

Realistic VLIW Datapath



Scheduling for Fine-Grain Parallelism

- The program is translated into primitive RISC-style (three address) operations
- Dataflow analysis is used to derive an operation precedence graph from a portion of the original program
- Operations which are independent can be scheduled to execute concurrently contingent upon the availability of resources
- The compiler manipulates the precedence graph through a variety of semantic-preserving transformations to expose additional parallelism

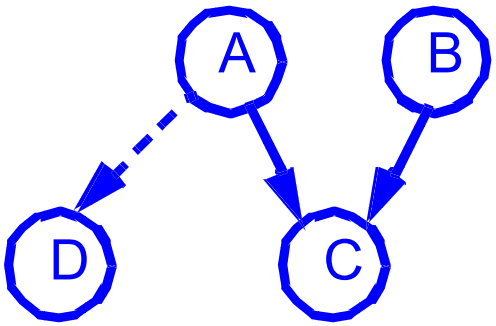
Example

```
e = (a + b) * (c + d)
b++;
```

Original Program

```
A:  r1 = a + b
B:  r2 = c + d
C:  e = r1 * r2
D:  b = b + 1
```

3-Address Code



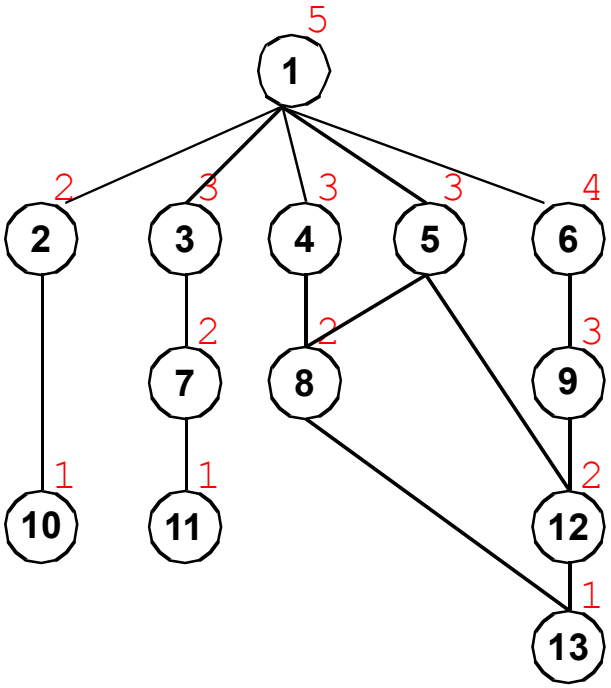
Dependency Graph

00:	add a,b,r1	add c,d,r2	add b,1,b
01:	mul r1,r2,e	nop	nop

VLIW Instructions

VLIW List Scheduling

- Assign Priorities
- Compute Data Ready List - all operations whose predecessors have been scheduled.
- Select from DRL in priority order while checking resource constraints
- Add newly ready operations to DRL and repeat for next instruction



4-wide VLIW				Data Ready List
1				{1}
6	3	4	5	{2,3,4,5,6}
9	2	7	8	{2,7,8,9}
12	10	11		{10,11,12}
13				{13}

Enabling Technologies for VLIW

- VLIW Architectures achieve high performance through the combination of a number of key enabling *hardware* and *software* technologies.
  - Optimizing Schedulers (compilers)
  - Static Branch Prediction
  - Symbolic Memory Disambiguation
  - Predicated Execution
  - (Software) Speculative Execution
  - Program Compression

Strengths of VLIW Technology

- Parallelism can be exploited at the instruction level
  - Available in both vectorizable and sequential programs.
- Hardware is regular and straightforward
  - Most hardware is in the datapath performing useful computations.
  - Instruction issue costs scale approximately linearly

  Potentially very high clock rate
- Architecture is "*Compiler Friendly*"
  - Implementation is completely exposed - 0 layer of interpretation
  - Compile time information is easily propagated to run time.
- Exceptions and interrupts are easily managed
- Run-time behavior is highly predictable
  - Allows real-time applications.
  - Greater potential for code optimization.

Weaknesses of VLIW Technology

- No object code compatibility between generations
- Program size is large (explicit NOPs)

  Multiflow machines predated "dynamic memory compression" by encoding NOPs in the instruction memory
- Compilers are extremely complex
  - Assembly code is almost impossible
- Philosophically incompatible with caching techniques
- VLIW memory systems can be very complex
  - Simple memory systems may provide very low performance
  - Program controlled multi-layer, multi-banked memory
- Parallelism is underutilized for some algorithms.

VLIW vs. Superscalar *[Bob Rau, HP]*

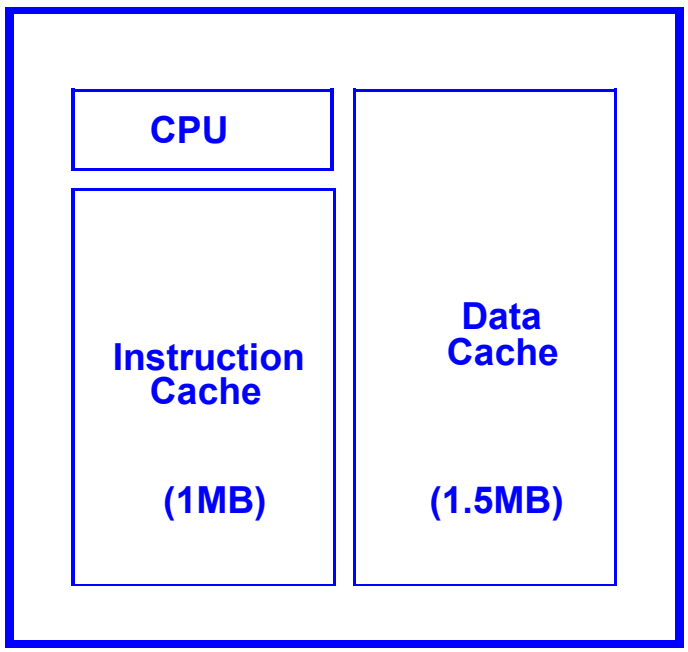
Attributes	Superscalar	VLIW
Multiple instructions/cycle	yes	yes
Multiple operations/instruction	no	yes
Instruction stream parsing	yes	no
Run-time analysis of register dependencies	yes	no
Run-time analysis of memory dependencies	maybe	occasionally
Runtime instruction reordering	maybe (Resv. Stations)	no
Runtime register allocation	maybe (renaming)	maybe (iteration frames)

Real VLIW Machines

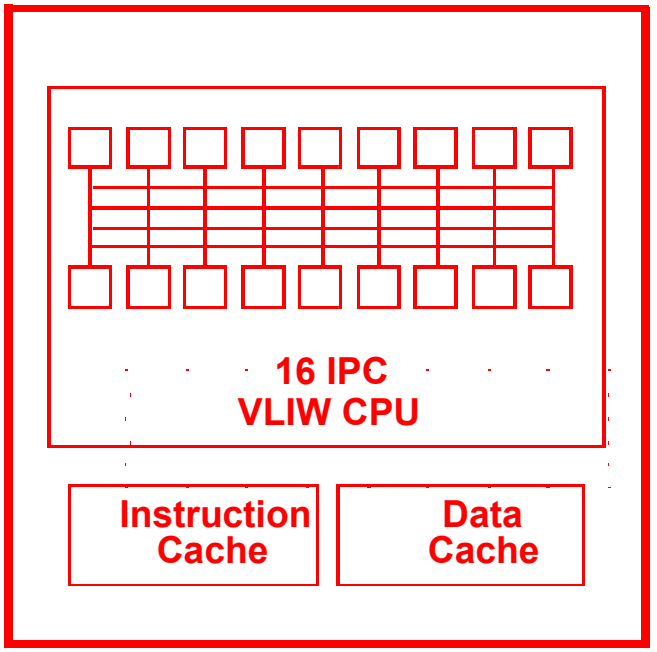
- VLIW Minisupercomputers/Superminicomputers:
  - Multiflow TRACE 7/300, 14/300, 28/300 *[Josh Fisher]*
  - Multiflow TRACE /500 *[Bob Colwell]*
  - Cydrome Cydra 5 *[Bob Rau]*
  - IBM Yorktown VLIW Computer (research machine)
- Single-Chip VLIW Processors:
  - Intel iWarp, Philip's LIFE Chips (research)
- Single-Chip VLIW Media (through-put) Processors:
  - Trimedia, Chromatic, Micro-Unity
- DSP Processors (TI TMS320C6x)

- Intel/HP EPIC IA-64 (Explicitly Parallel Instruction Comp.)
- Transmeta Crusoe (x86 on VLIW??)
- Sun MAJC (Microarchitecture for Java Computing)

Why VLIW Now?



1 Billion Transistor Superscalar Processor



1 Billion Transistor VLIW Processor

- Nonscalability of Superscalar Processor
  - ILP and complexity
- Better compilation technology

Performance Obstacles of Superscalars

- Branches
  - branch prediction helps, but penalty is still significant
  - limits scope of dynamic and static ILP analysis + code motion
- Memory Load Latency
  - CPU speed increases at 60% per year
  - memory speed increases only 5% per year
- Memory Dependence
  - disambiguation is hard, both in hardware and software
- Sequential Execution Semantics ISAs
  - total ordering of all the instructions
  - implicit inter-instruction dependences

Very expensive to implement wide dynamic superscalars

Intel/HP EPIC/IA-64 Architecture

- EPIC (Explicitly Parallel Instruction Computing)
  - An ISA philosophy/approach

    e.g. CISC, RISC, VLIW
  - Very closely related to but not the same as VLIW
- IA-64
  - An ISA definition

    e.g. IA-32 (was called x86), PA-RISC
  - Intel's new 64-bit ISA
  - An EPIC type ISA
- Itanium (was code named Merced)
  - A processor implementation of an ISA

    e.g. P6, PA8500
  - The first implementation of the IA-64 ISA

IA-64 EPIC vs. Classic VLIW

- Similarities:
  - Compiler generated wide instructions
  - Static detection of dependencies
  - ILP encoded in the binary (a group)
  - Large number of architected registers
- Differences:
  - Instructions in a bundle can have dependencies
  - Hardware interlock between dependent instructions
  - Accommodates varying number of functional units and latencies
  - Allows dynamic scheduling and functional unit binding

  Static scheduling are "suggestive" rather than absolute

  $\Rightarrow$ Code compatibility across generations

  but software won't run at top speed until it is recompiled so "shrink-wrap binary" might need to include multiple builds

Project 3 Ideas

- How to get 15% more performance
  - better branch prediction
  - better instruction and data prefetching
  - value prediction
  - out-of-order/speculative load
  - better trace cache
- How to get good performance-to-cost ratio
  - start with an “effective” mechanism

    Concentrate on the bottleneck!!
  - eliminate excess

Good places to start looking for ideas are Proceedings of ISCA (International Symposium on Computer Architecture) and Micro (International Symposium on Microarchitecture). Papers can be downloaded from http://ieeexplore.ieee.org using any CMU machine. Make sure you cite the sources in your report.

What to Do?

- Just twiddling “baseline8” parameters won’t get you 15%
- But, you most likely will have to *tune* the machine parameters to get the full benefit of whatever mechanisms you elect to add

  Hint: don’t run the full benchmarks for all tuning runs. Use representative sample sections instead!!
- Once you achieved 15% performance gain, tune some more for lower cost
- 10% Early Bonus
  - report due 12/5, presentation due 12/7
- 5% Presentation Bonus *(must also earn early bonus)*
  - selected for presentation on 12/10