VU2 185.324

Compilation Techniques for VLIW Architectures

Dietmar Ebner ebner@complang.tuwien.ac.at
Florian Brandner brandner@complang.tuwien.ac.at

http://complang.tuwien.ac.at/cd/vliw

Last Lecture

- VLIW principles
- Forms of parallelism
  - Pipelining
  - Instruction level parallelism (ILP)
- VLIW vs. superscaler
  - Similar ILP
  - Differing techniques to achieve ILP

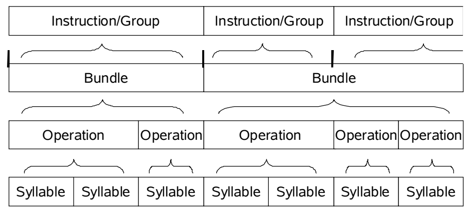
Last Lecture (2)

- Instruction set design
  - Expose details (Latencies/Resources)
  - Semantics of parallel execution
  - Exceptions/Interrupts
- Instruction encoding
  - Uncompressed/Fixed-overhead encoding
  - Distributed/Template-based encoding
  - Dispatching techniques

Terminology

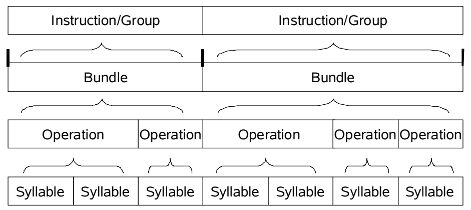
- Operation
- Syllable
- Instruction/group
  - Set of independent operations
- Bundle
  - Set of operations encoded as a unit
  - May have dependencies

Last Lecture (4)



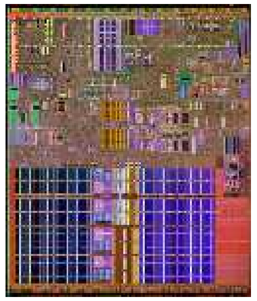
Last Lecture (5)

In many cases bundles and instructions coincide



In Today's Lecture

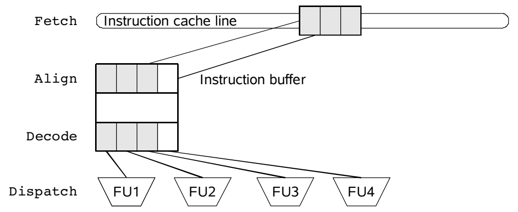
- VLIW Implementation
  - Architecture Design
  - Microarchitectural Design
- Examples
  - Multiflow Trace
  - ST231
  - Chili



Memory Architecture

- Strongly oriented towards RISC
  - Dedicated load/store operations
  - Typical addressing modes
    - register indirect, register + offset, register + register
    - pre/post increment/decrement
  - Similar alignment, and access sizes
- Same caching and prefetching techniques

Instruction Fetch



Source: P. Faraboschi, HP Research

Instruction Fetch

- Additional step required: Align
  - Find the boundaries of instructions
  - Usually employs an instruction buffer
- Expansion of NOPs
  - Both vertical and horizontal

Instruction Alignment

- Variable length instructions
  - Complicate the Align step
  - Might cause several cache accesses
- Interferes with cache organization
  - Instruction encoding
  - Alignment constraints

Memory Architecture (2)

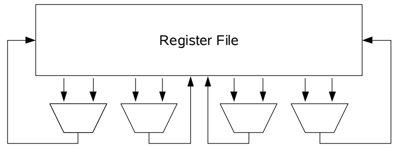
- Some techniques not unique to VLIW
- Stream buffers
  - Dedicated caches relaying on spatial locality only
- Fast local memories
  - Scratch pad memories
  - Lockable caches (turned into regular memory)
- DMA engines

Code Compression

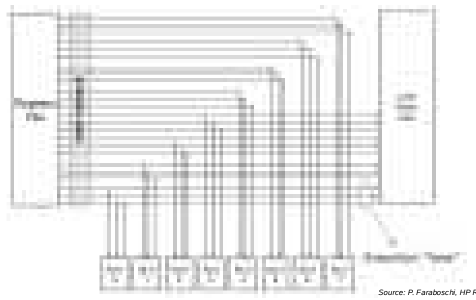
- Compressed code in memory
- Decompression
  - On instruction fetch
    - Yet another step before Align
  - On cache refill
    - Access to memory is slow, thus allows more complex compression schemes
    - Cache holds uncompressed instructions, thus reduces effective cache size
- Example: IBM Codepack

Register File

- Multiple ports
  - Typically 2 read, 1 write port for each execution unit

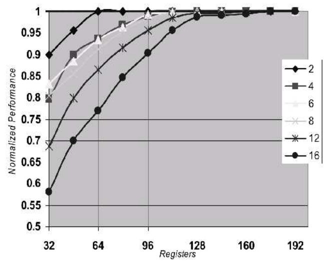


VLIW Datapath Example



Source: P. Faraboschi, HP Research

Register Count

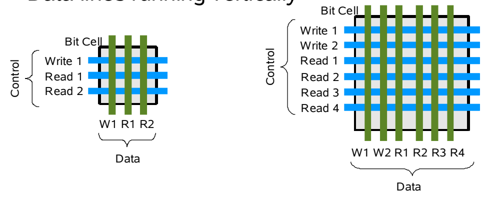


- Media-oriented benchmarks
- 2-16 execution units

Source: P. Faraboschi, HP Research

Register File Structure

- Organized as a two-dimensional grid
  - Built from bit cells
  - Control lines running horizontally
  - Data lines running vertically

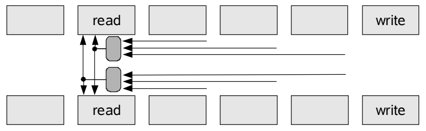


Register File Structure (3)

- Implementation
  - Dominated by wires
  - Thus shrinking transistors does not help
  - Area grows with the number of ports
  - Similarly the access time, and power consumption
- Example: 130nm process
  - Maximum of 20 ports (read/write)
  - Sweet spot: 12 ports (8 read/4 write)

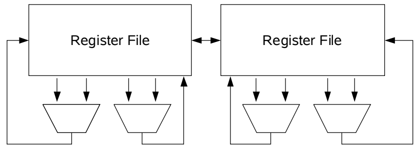
Bypassing

- Complexity increases:
  - Number of read ports x write ports x pipeline depth

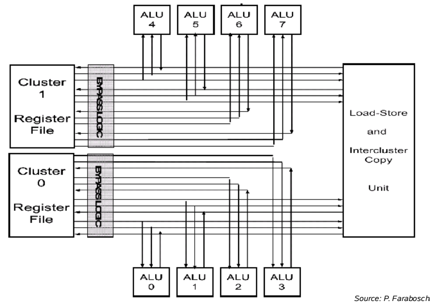


Clustered Register Files

- Reducing the number of ports
  - Split the register file into several smaller
  - Copy between the clusters



Clustered Datapath

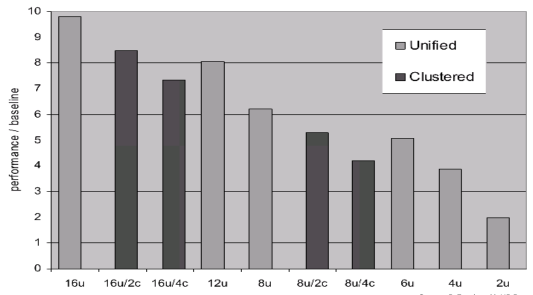


Source: P. Faraboschi, HP Research

Clustering

- Architecturally invisible
  - Illusion of a single large register file
    - Implemented in hardware
  - Encoding overhead (bits for all registers)
- Architecturally visible
  - Implicit vs. explicit copying
- Complicates the compiler

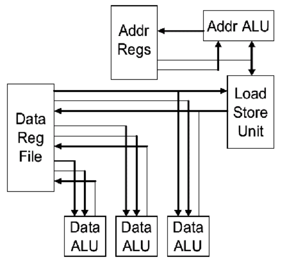
Cost of Clustering



Source: P. Faraboschi, HP Research

Address Registers

- Special form of clustering
- Not recommended



Source: P. Faraboschi, HP Research

Indexed Register Files

- Generalization of register windows (Sparc)
- Offer `base + offset` addressing for registers
- Rotating register files
  - Policy to update the base address
    - e.g. cycle over a subset of the register file
  - Used for software pipelining
- Benefits unclear

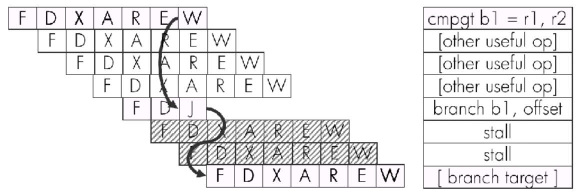
Branch Architecture

- Again similar to RISC
  - Common prediction techniques
- Unbundling branches
  - Split the branch into several steps
  - Prepare the target instruction streams to fetch
  - Select the instruction stream that will be taken
  - Execute the taken instructions

Unbundling branches

- Two-step branching
  - Separate branching from condition
  - Usually using dedicated branch registers
- Three-step branching
  - Additionally decouple the target address calculation

Example: Two-step branch



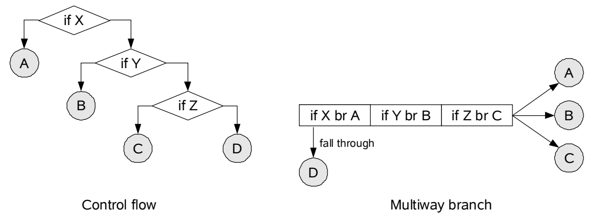
cmpgt b1 = r1, r2
[other useful op]
[other useful op]
[other useful op]
branch b1, offset
stall
stall
[branch target]

Source: P. Faraboschi, HP Research

Multiway Branches

- Several branches in one instruction
  - Can be seen as a single jump with multiple targets
- Why multiway branches?
  - One branch every 5-10 instructions
  - Branches get the bottleneck for larger ILP
  - Reduces the number of branches
  - Alternative: Predication

Multiway Branches (2)



Branches need to be prioritized when conditions are not mutual exclusive!

History

- Attached signal processors (70s-80s)
  - ILP similar to VLIW
  - Very idiosyncratic
  - Exclusively hand-coded
- Horizontal microcode (70-80s)
  - Control of hardware to emulate a complexer high-level instruction set
  - Hardware operates in parallel
  - Thus microcode needs to be parallel

The first VLIWs

- Research
  - ELI-512 (1983)
    - Joseph Fisher, et.al.
- Commercial
  - Multiflow - Trace (1984-1990)
    - Fisher, et. al.
  - Cydrome - Cydra (1984-1988)
    - Rau, et. al.
  - Initially successful

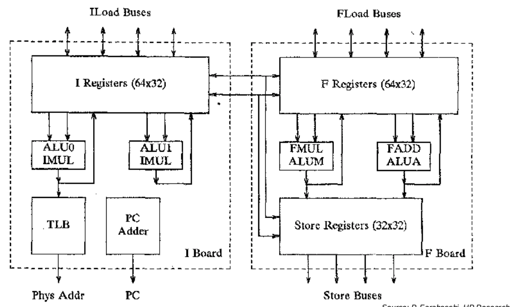
Todays VLIWs

- Intel Itanium (EPIC)
  - The only VLIW for workstations and servers
  - Research started 1989 by Hewlett-Packard
  - Partnered with Intel in 1994
  - Rau, Fisher both involved
- Many VLIWs in the embedded domain
  - Philips TriMedia, TI C6x, StarCore, ST2xx, ...
  - OnDemand Chili

Multiflow Trace

- Built from 5 board types
  - Pairs of Integer/Float boards (I-F)
  - Global/memory/IO controller
  - 45x45cm each
  - Based on numerous gate arrays (8000 gates)
- Configurations with 1,2 or 4 I-F boards
  - Issuing up to 28 operations per cycle

I-F Board Pair



Source: P. Faraboschi, HP Research

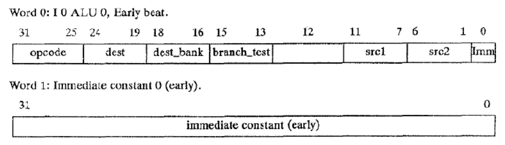
Instruction Set Architecture

- Register files
  - 64 32-bit integer registers
  - 64 32-bit float registers (paired)
  - Clustered, with explicit copy
- Instructions
  - Integer select - partial predication
  - “Fast moves” for register file-register file transfers
  - Speculative loads
  - Non-trapping floating point instructions

Instruction Set Architecture (2)

- Two-step branches
  - 7 branch registers
  - 1 delay slot
  - Multiway branches (up to 4 in parallel)
- Encoding
  - Fixed-overhead with 32-bit syllables
  - 7 syllables for each I-F pair
  - 256-bit to 1024-bit instruction word
  - Expanded at cache refill

Instruction Encoding


Word 0: I 0 ALU 0, Early beat.
31
25
24
19
18
16
15
13
12
11
7
6
1
0
opcode
dest
dest_bank
branch_test
src1
src2
Imm
Word 1: Immediate constant 0 (early).
31
0
immediate constant (early)

Memory

- Virtual memory
  - Sophisticated paging and TLB
  - Full Unix OS support
- No data caches, but
  - Interleaved memory banks
  - Allowing 4 accesses to be initiated in parallel
  - Bank stalls on actual conflicts
- Instruction cache

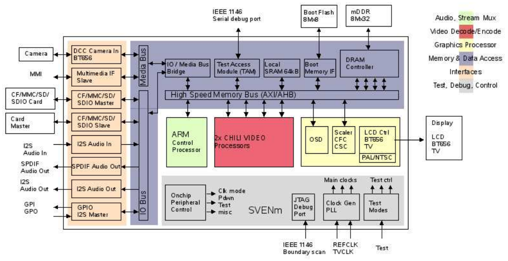
OnDemand Chili

- 4-way VLIW architecture
- 64 32-bit general purpose register file
- Instruction Set Architecture
  - Full set of Micro-SIMD instructions
  - Distinctive predication model
  - No floating point support

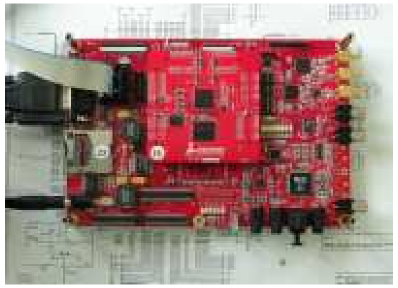
OnDemand Chili (2)

- SVENm - SoC
  - 2 Chili v1.0 cores
  - ARM control processor, running Linux
- Applications
  - Mobile multimedia processing
  - Focus on video decoding
  - H264, MPEG, VC-1, etc.
- Developed in Austria
  - Built in China though

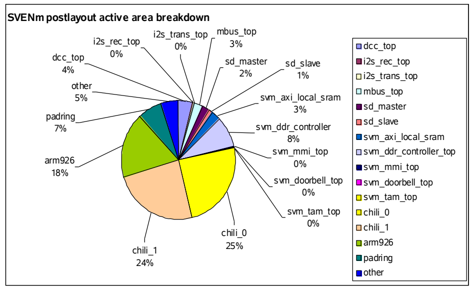
SVENm - Overview



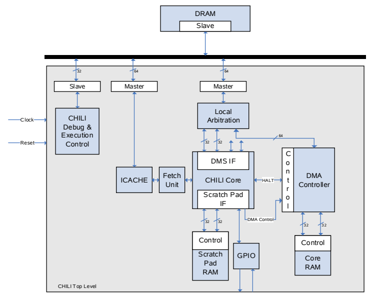
SVENm – Board



SVENm - Area



Chili Core Overview



Chili - Pipeline

- 4 identical pipelines in parallel
- 11 stages
  - Fetch/Expand/Decode
  - Forward/Execute/Memory access
  - Writeback
  - Limited hazard detection and stalling
- Typical latency of 1 cycle
  - Multiplication 3 cycles
  - Loads up to 40 cycles!

Chili - ISA

- Special instructions
  - Clip integer value to upper/lower bound
  - Align and round, for fractional data types
  - Population count, leading 0s, leading 1s, etc.
  - Sum of absolute differences (SAD)
  - Multiply accumulate (MAC)
- Full 2x16-bit Micro-SIMD
  - Powerful permute instructions
- No division

Chili – Predication

- Full predication
  - Not all operations can be predicated
  - Unfortunate: loads/stores can not be predicated
- Requires two operations
  - Test and predicated operation execute in parallel
  - Saves extra cycles
  - May increase code size

Chili – Branches

- May only reside in the first slot
- 5 delay slots
  - Up to 23 operations are execute after the branch
- May be predicated
  - No unbundling possible

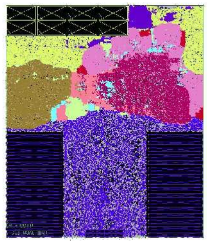
Chili – Encoding

- 32-bit syllables
- Operations require 1-2 syllables
- Variable-length bundles
  - Neither vertical nor horizontal NOPs
  - Always encodes 4 operations
  - Requires 16-32 bytes
- May cross cache boundaries

Chili – Memory

- Several memories
  - Slow, shared DDR RAM
  - Faster, shared SRAM
  - Fast, local scratchpad memory
- Issue up to 4 memory operations
  - Accesses executed out-of-order
  - Interleaved memory banks

Chili - Floorplan

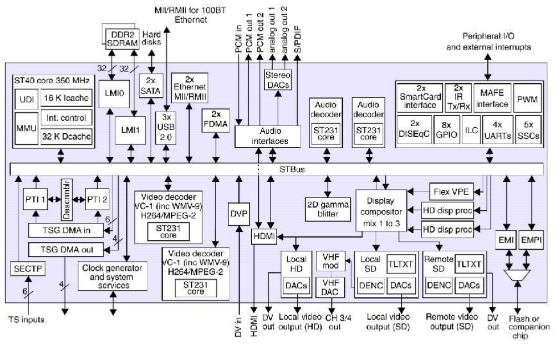


🟦	Fetch
🟥	Decode
🟪	Execution Units
🟦	Comparators
🟫	Register File
🟥	Forwarding
🟨	Memory Subsystem (DMS)
🟪	DMS Bus Interface
🟩	DMS Core Interface
🟥	Multiplexer

ST 2xx

- Joint development by HP and STMicro.
  - Based on the HP-LX research project
- 4-way VLIW architecture
- 64 32-bit general purpose register file
- Instruction set architecture
  - Dismissible loads
  - Fractional arithmetic
  - Partial predication (select)

ST231 - Applications



ST231 - Pipeline

- No details available
  - Not completely symmetric
  - 4 integer execution units
  - 2 multiply units
  - 1 load/store, branch unit
  - Hazard detection with stalling
- Typical latency of 1 cycle
  - Loads and multiplications take 3 cycles

ST231 - ISA

- Special instructions
  - Min/max
  - Count leading 0s
  - Multiplies for fractional data types
  - Division step
- Partial predication
- Dismissible loads
  - Prefetching to dedicated buffer

ST231 - Branches

- May only reside in the first syllable
- Two-step branches
  - 8 branch registers
  - 2 bundle distance between condition and branch
- No delay slot
  - All branches incur 1 stall cycle

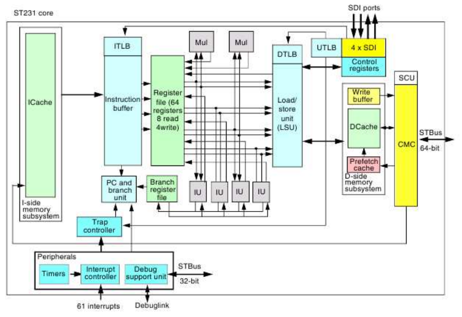
ST231 - Encoding

- Bundles of up to 4 syllables
- Distributed encoding
  - Positional dispatch
  - 32 bit syllables, 1 stop bit
- Immediates
  - 9-bit short immediate
  - 9+23-bit extended immediates
- Restrictions for
  - Branches, multiplies, extended immediates

ST231 - Memory

- Issue one single access per cycle
- Caches
  - 32kb instruction cache
  - 32kb lockable data cache (8kb-24kb local RAM)
  - 256 byte prefetch buffer
- Streaming data interface (SDI)
  - Fast streaming I/O
  - Does not pollute caches

ST231 – Block Diagram



Outlook

- Overview of traditional compilers
- Introduction to ILP aware compilation
- Profiling techniques
- Phase ordering problems