

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

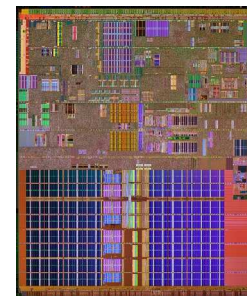
- Product life cycle
 - Product volume (“long tail” vs. “head”)
- Constraints
 - Performance/power/size
 - Production costs/Development costs
 - Market/Time to market

Last Lecture (2)

- Characterization
 - Main purpose is not computing
 - Applications (consumer, comm., automotive)
 - Processor architecture
 - Work load
- Compatibility
- Custom solutions (ASIP, DSP, SoC)

In Today's Lecture

- VLIW principles
 - ILP
 - VLIW vs. Superscalar
- Instruction Set
 - Execution model
 - Extensions
- Instruction Encoding



VLIW Principles

Expose ILP in the architecture design.

If you can do it in software, do it in software!

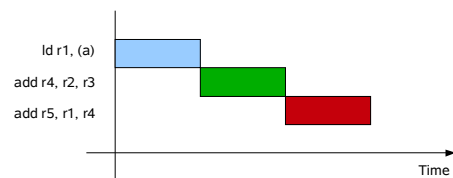
Clean successor of RISC.

Forms of Parallelism

- Pipelining
- Data parallelism (SIMD, Vector processing)
- Instruction level parallelism (ILP)
- Thread level parallelism (TLP)

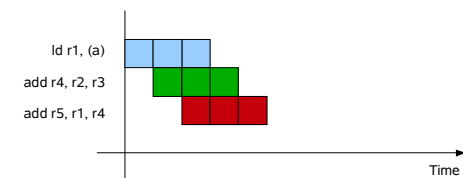
Sequential Execution

- Process one instruction at a time
- No parallelism at all



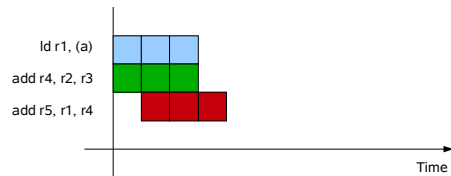
Pipelined Execution

- Divide instructions into stages
- Parallel processing of independent stages

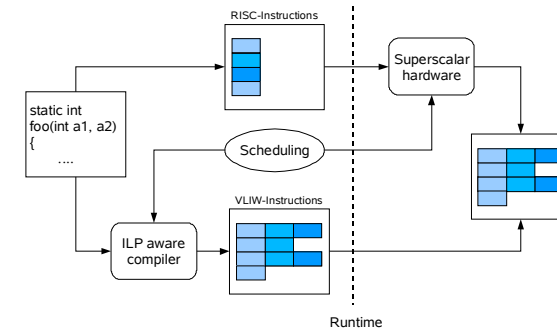


Parallel Execution

- Group independent operations
- Parallel processing of operations



VLIW vs. Superscalar



Example: MIPS 74k

```
void vec_sum(int *a, int *b, int*c, int n) {
    for (int i = 0; i < n; i++)
        (*c++) = (*a++) + (*b++);
}
```

	IEU	IEU/AGEN
(0) .BB1_2:		
(1) lw \$10,0(\$4)	(2) addiu \$3,\$3,1	(1) lw \$10,0(\$4)
(2) addiu \$3,\$3,1	(4) addiu \$4,\$4,4	(3) lw \$9,0(\$5)
(3) lw \$9,0(\$5)	(5) addiu \$5,\$5,4	(6) addu \$9,\$9,\$10
(4) addiu \$4,\$4,4	(9) addiu \$6,\$6,4	(7) sw \$9,0(\$6)
(5) addiu \$5,\$5,4		
(6) addu \$9,\$9,\$10		
(7) sw \$9,0(\$6)		
(8) bne \$3,\$7,.BB1_2		
(9) addiu \$6,\$6,4		

IEU ... Integer execution unit
IEU/AGEN ... Shared integer unit and address generator

Example: ST231

- Compiler generates 2 loops

- first captures initial (n & 3) iterations
- second is 4 times unrolled

```
(0) L?_0_9:
(1) ldw $r9= 0[$r17]
(2) add $r16 = $r16,4
(3) add $r17 = $r17,4
(4) add $r20 = $r20,-1;;
(5) ldw $r10 = -4[$r16]
(6) convib $b1 = $r20
(7) add $r18 = $r18,4
(8) add $r15 = $r15,1;;
(9) add $r9 = $r9,$r10;;
(10) stw -4[$r18] = $r9
(11) br $b1, L?_0_9;;
```

Architecture Classification

	Sequential Architectures <i>Superscalar</i>	Dependence Architectures <i>Dataflow</i>	Independence Architectures <i>VLIW</i>
Dependence information in the program	Implicit via register names	Exact description of all dependences	Explicit description of some independent operations
How are dependent operations typically exposed	By the hardware's control unit	By the compiler (and they are embedded into the program)	By the compiler (and they are implicit in the program)
How are independent operations typically exposed	By the hardware's control unit	By the hardware's control unit	By the compiler (and they are implicit in the program)
Where is the final operation scheduling typically done	In the hardware's control unit	In the hardware's control unit	In the compiler
Role of the compiler	Rearrange code to make parallelism more evident and accessible to the hardware	Replace some of the analysis hardware found in superscalars	Replaces virtually all hardware dedicated to parallelism exposure and scheduling

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #13

VLIW vs. Superscalar

Superscalar

- Scheduling is done by hardware
- Sequential stream of scalar operations
- Allows for In-order and out-of-order execution
- Number of issued instructions is dynamically determined by a hardware dispatch unit
- **Microarchitecture** technique

VLIW

- Scheduling is done by Software
- Sequential stream of parallel operations
- Only allows for in-order issue
- Number of issued instructions is statically determined by the compiler
- **Architecture** technique

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #14

The Role of the Compiler

- Compilers are important
 - Especially for VLIW architectures
 - Less for superscalar
- Extracting parallelism
 - Instruction scheduling
 - Predication
 - Speculation
 - Loop optimizations

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #15

Instruction Set

- Interface to the programmer/compiler
- Explicitly hide or expose architectural features
- Instruction encoding
 - Bundles/groups of operations
 - Legal combinations
- Binary compatibility

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #16

Architectural Features

- Execution model
 - Operation latencies
 - Computational resources
 - Semantics of parallel execution
- Exceptions/Interrupts
- Extensions
 - Predication
 - Speculation

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #17

Execution Model

- Many details are exposed to the programmer/compiler
 - Which operations are executed in parallel
 - How an operation is executed
 - When an operation is finished
 - Handling of hazards
- Complicates compilers
- Simplifies hardware

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #18

Semantics

- What is the value of r1?

```
mov $r1 = 1
mov $r2 = 2;;
mov $r2 = 3
mov $r1 = $r2;;
```

- Is this valid?

```
{
  mov r1, 0
  if (r2 == 3) mov r1, 1
}
```

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #19

Exceptions/Interrupts

- Exceptions may be raised in parallel
 - Precise vs. imprecise exceptions
 - Which instruction(s) caused the exception(s)
 - In which order are they processed
- Restarting execution
 - Which operations need to be reexecuted

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #20

Predication

- Conditionally nullify the effect of operations
- Full predication
 - All (almost all) operations can be predicated
- Partial predication
 - Only a few instructions can be predicated
 - Conditional move (cmov)
 - Select

Speculation

- Speculatively execute operations
 - Even if the calculation is useless
 - Even if the calculation may be incorrect
- May require compensation
 - Suppress exceptions
 - Undo incorrect calculations

Instruction Encoding

- Bridges between software and hardware
 - Programs are transformed to binary code
 - Hardware executes programs based on binary code
- Strong connection between architectural style and encoding
 - Encoding for RISC machines
 - CISC encoding techniques
 - Special techniques for VLIW

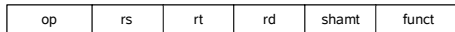
RISC Encoding

- RISC advocates simplicity and regularity
 - Encoding uses a fixed length
 - Few encoding formats
 - Reserved space for future extensions
- Simple decoding hardware
- Some code size overhead

Example: MIPS

- fixed width of 32 bit
- 3 encoding formats:

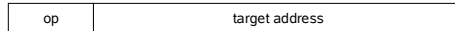
R-Format:



I-Format:



J-Format:

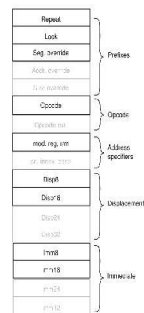


CISC Encoding

- Variable length
 - Frequent instructions get short encoding
 - Infrequent instructions get longer encodings
 - Easier to add extensions
- Complex structure
 - Encode length of the instruction
 - Many instructions and instruction variants
 - More complex hardware

Example: x86-32

- 7 addressing modes
- 5 different prefixes
- 4 displacement variants
- 4 immediate variants
- 1 – 17 bytes in size



VLIW Encoding

- Inspired by RISC encoding
- Decompose a VLIW into fixed sized junks
 - Fixed encoding width for operations
 - Few encoding formats for operations
 - Variable encoding width for instructions
- Simple encoding scheme for bundles

Terminology

- Instruction/group
 - Independent operations that can be executed in parallel
- Bundle
 - Group of operations that are encoded in the same VLIW
 - Not necessarily independent

Terminology (2)

- Operation
 - Basic operation of the execution pipeline
 - Similar to RISC operations/instructions
- Syllable
 - Basic unit for the instruction encoding
 - Fixed bit width
 - Typically encodes one single operation

Example: Intel Itanium

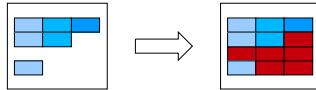
- 41 bit syllable
- 41 bit operations (exactly one syllable)
- Bundle
 - 3 syllables/operations
 - 5 bits template and stop bit
- Instruction/group
 - Several bundles
 - Variable length

VLIW Encoding Schemes

- Uncompressed Encoding
- Fixed-overhead Encoding
- Distributed Encoding
- Template-based Encoding

Uncompressed Encoding

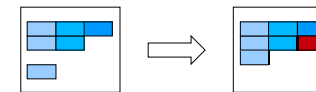
- Explicitly encode all operations
 - Including explicit NOPs if no useful operations could be found
- Allows for simple decoding
- Very bad code size
 - Negative effect on instruction cache



03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #33

Saving NOPs

- Horizontal NOPs
 - Replace consecutive NOPs in a bundle
- Vertical NOPs
 - Replace consecutive bundles of NOPs
- Dynamically expanded during decoding



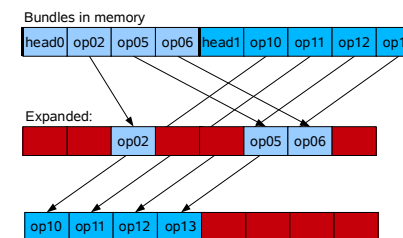
03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #34

Fixed-overhead Encoding

- Variable length bundles
 - Based on horizontal NOPs
- Prepend a header to each bundle
 - Count of operations
 - Map operations to functional units
- Adopted by early architectures
 - e.g. Multiflow

03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #35

Fixed-overhead Encoding (2)

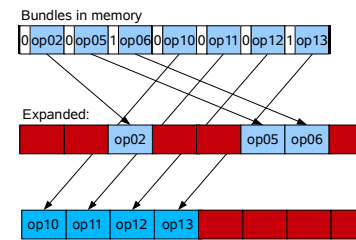


03/28/08 Ebner, Brandner | Compilation Techniques for VLIWs | SS08 Slide #36

Distributed Encoding

- Header information distributed
 - Either using a stop-bit or parallel-bit
 - Encoded with operations
- Reduces code size
- More complex decoding
 - Requires a search for the stop-bit
- Adopted by ST231

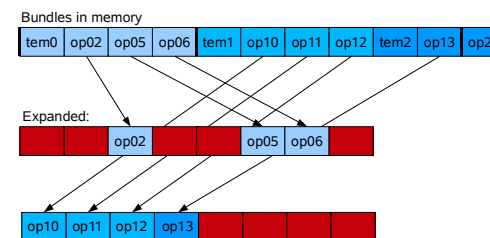
Distributed Encoding (2)



Template-based Encoding

- Similar to fixed-overhead encoding
 - Limits number of combinations
- Space efficient
- Low hardware overhead
- Adds complexity to compiler
- Adopted by Intel Itanium

Template-based Encoding (2)



Dispatching

- Assign operations to computational units
- Explicitly encoded in the simple scheme
 - 1:1 mapping of operations and functional units
- The mapping is lost for the other schemes

Dispatch (2)

- Unit identifiers within operations
- Explicit mapping
 - Fixed-overhead encoding
 - Within templates
- Positional encoding
 - Based on syllable ordering within instructions

Encoding Tricks

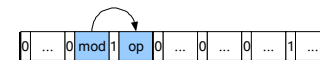
- Embedding large immediates
 - Special immediate operation
 - Used by some parallel operation



- Adopted by ST231

Encoding Tricks (2)

- Modifying the effect of parallel operations
 - Special test/modifier operations



- Adopted by Chili

Outlook

- Historical Perspective
- Architectural Structures
- Microarchitectural Design Issues
- Clustered Architectures
- Examples: Multiflow, Chili, ST231