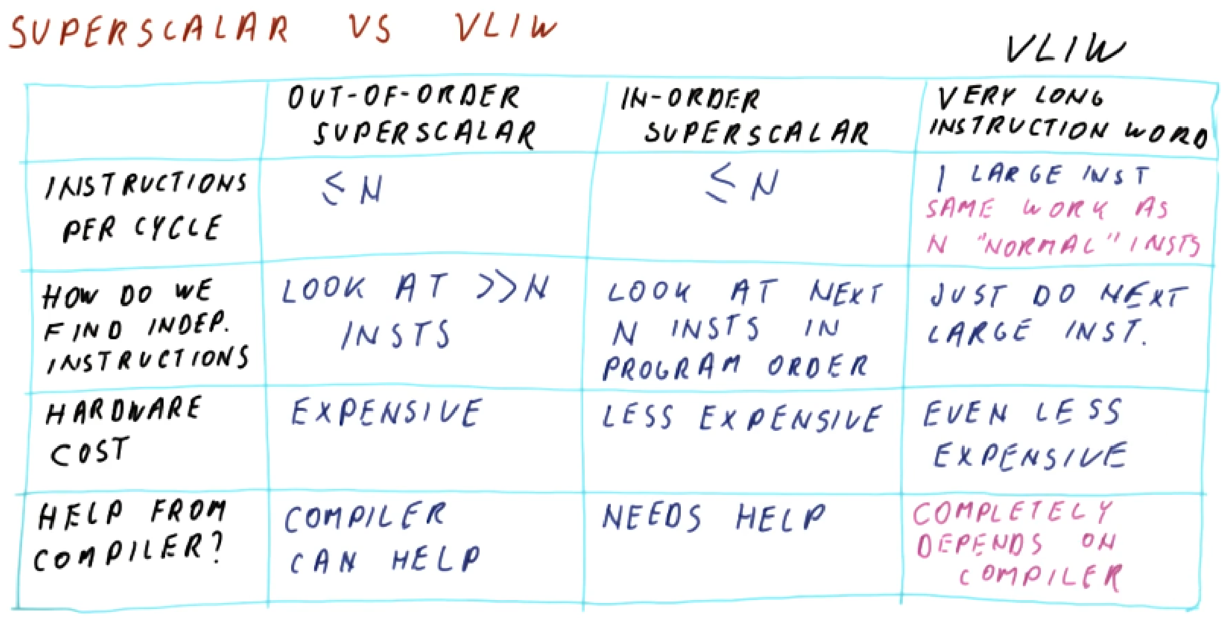
Very Long Instruction Word

Introduction

1. VLIW processors are different from out-of-order processors in that they do not try to identify ILP on their own; they simply do what the compiler tells them to do

Superscalar vs VLIW

1. Superscalar: Attempt to execute more than one instruction per cycle
2. VLIW: Attempt to achieve the same thing in a different way



SUPERSCALAR VS VLIW

VLIW

	OUT-OF-ORDER SUPERSCALAR	IN-ORDER SUPERSCALAR	VERY LONG INSTRUCTION WORD
INSTRUCTIONS PER CYCLE	$\leq N$	$\leq N$	1 LARGE INST SAME WORK AS N "NORMAL" INSTS
HOW DO WE FIND INDEP. INSTRUCTIONS	LOOK AT >>N INSTS	LOOK AT NEXT N INSTS IN PROGRAM ORDER	JUST DO NEXT LARGE INST.
HARDWARE COST	EXPENSIVE	LESS EXPENSIVE	EVEN LESS EXPENSIVE
HELP FROM COMPILER?	COMPILER CAN HELP	NEEDS HELP	COMPLETELY DEPENDS ON COMPILER

VLIW and Superscalar Comparison

Superscalar vs VLIW Quiz

1. Out-of-order superscalar processor
   - 32-bit instruction
2. VLIW processor
   - 128-bit instruction, each specifying 4 operations
3. For a program size of 4000 byes, the VLIW program size is between 4000 and 16000 bytes
   - At best, the program size can't be smaller than 4000 bytes
   - However, if we can't fit multiple instructions, we might have to fill 3 of the VLIW operations with no-ops, causing the size of 16000 bytes

The Good and the Bad

1. Good
   - Compiler does the hard work
   - Compiler is only once; plenty of time to optimize
   - Simpler HW than a comparable OOO processor
   - Can be more energy efficient
   - Works well on loops and "regular" code

    - Traversing arrays, multiplying matrices, etc.
2. Cons:
    - Latencies of instructions are not always the same (e.g. cache miss)
    - Irregular applications (pointer-intensive code, AI applications)
        - VLIW struggles in cases where many decisions are required
    - Code bloat
        - VLIW can introduce lots of no-ops

VLIW Backward Compatibility Quiz

1. Simple VLIW processor
    - 64-bit instructions (2 ops)
    - Fetch, decode, execute 1 instruction/cycle
2. Better VLIW
    - 4 operations/cycle
    - Fetch, decode, execute 2 64-bit instructions/cycle
3. Is this a VLIW processor?
    - No. We cannot "widen" a VLIW processor without recompiling the code. The compiler is promising the the entire N-instruction word is free of dependencies. If we combine 2 such instructions, this guarantee is broken.

VLIW Instructions

1. Instruction set has all the "normal" ISA opcodes
2. Full predication support
    - Relies on compiler to expose parallelism; one way compiler does this is through scheduling
3. Lots of registers
    - Scheduling optimizations require additional registers
4. Branch hints: Compiler specifies to hardware what it thinks the branch will do
5. VLIW instruction "compaction"
    - Eliminate no-ops by adding "end-of-instruction" markers

VLIW Examples

1. Intel Itanium: Example of how not to do VLIW
    - Tons of ISA features
    - Hardware was very complicated (probably most complicated Intel ever built)
    - Still not great on irregular code
2. Digital signal processing processors
    - Regular loops, lots of iterations with floating-point operations
    - Excellent performance
    - Very energy-efficient

VLIW Target Market Quiz

1. VLIW is best for:
    - Add many numbers together (best)
    - Figures out best path in a maze (worst)
    - Counts elements of a linked list (mid)

Conclusion

1. People had high hopes for Intel Itanium, but they weren't able to deliver on these promises