Very Long Instruction Words

Brian Halbert
David Pyenson
What is a Very Long Instruction Word?

- Instructions that are created by a compiler/preprocessor
- The compiler breaks multiple instructions that can be executed simultaneously by the processor
- Complex compiler allows for simple hardware
- Can be considered the next level of RISC
  - multiple parallel RISC instructions
History

- Alan Turing’s parallel computing studies in 1946 as well as Maurice Wilkes’s microprogramming work done in 1951, had key roots in the Very Long Instruction Word model.
- Joseph Fisher while working on horizontal microcoding for a CDC-6600 emulator in 1979 developed “trace scheduling.” A key element in generating VLIW code. Trace scheduling is a global code motion taking a loop to straight-line code sequences by utilizing loop unrolling and static branch prediction. Thus generating a sequential set of instructions with no need for a branch.
- The Multiflow compiler was developed by Fisher and colleagues in 1984 which never became a success due to the lacking marketplace.
- Further development slowed until it was discovered that the VLIW architecture is ideal for quick computation of extremely repetitive and complicated algorithms. Therefore emerging new waves of media processing chips. The first publicly successful implementation came quick as the Texas Instrument C6X series in the late 1980’s.
- Texas Instrument’s huge success was also due to the technology being utilized at the appropriate time. The memory expense was too large to accommodate in the 1970’s requiring memory development to catch up.
Benefits

- Lower power consumption due to the less intensive processor needed for a VLIW Architecture.
- VLIW compiler generates packets containing multiple operations giving a complete “plan of execution.”
- Compiler determines the dependencies and schedules its instruction packets according to latencies.
  - This creates code with no hazards and is fully scheduled not relying on the hardware to drive the code to discover and dependencies or reschedule instructions.
The largest implementation difficulty is also VLIW largest advantage. Writing a compiler that effectively predicts complex code branches developing one instruction for them is extremely complex. This is an example of the “Law of Conservation of Complexity”

Binary compatibility is difficult to achieve. If a piece of hardware was built to utilize five execution units, than a piece of software working with four or any other amount would not operate.

A compiler-first design has issues with its inflexibility. Instructions are ordered at compile time, leaving any memory issues, latency, cache misses, to happen after compile. Thus a slight deviation from a pure VLIW design may be needed for implementation.
How Does the Compiler Optimize Code?

- Trace Scheduling for additional parallelism
- Memory Bandwidth increases via Memory Reference and Memory Bank Disambiguation
- Complex Code Generation Algorithms
- Speculative Execution
- Software Pipelining
A trace is an acyclic path through a series of basic code blocks.
Attempts to find the most likely path.
Each trace can be compacted/optimized by performing code motion without regard to basic block boundaries.
- Code Motion - Reordering of instructions to move loop invariant code out of loops.
Main advantage is the ability to optimize across branches.
Handles loops poorly so code must be unrolled first.
Acyclic Path Through Code Blocks

One possible path that can be optimized.
Memory Bandwidth Increases

- Memory Reference Disambiguation
  - Do memory pointers point at the same location?
  - Can be aided by assertions from programmer
  - Easier to do on array references than other pointers

```
x1 = a[i]
a[i] += 1
x2 = a[j]
a[j] += 1
```

- Memory Bank Disambiguation
  - Loop unrolling to allow the program to work with multiple blocks of memory at once
  - Memory banks can be accessed via ‘a shortcut’ or the main memory controller
  - At compile time the banks must be known to form ‘shortcuts’ and to perform MBD

*This code block can double its execution time if i != j*
Complex Code Generation Difficulties

- **VLIW compiler** requires complex algorithms due to the constant attempt to fully utilize all functional units. In long data paths the compiler must generate cluster operations to limit any data moving between elements, this is known as **operation placement**.
- **Data Routing** is done by choosing data paths using buses and registers that transfer data between elements in the architecture. Since there are usually many paths, the compiler must analyze and select the path with the least interference with any other instructions.
- **Register Allocation** is difficult with a VLIW, making the compiler decide when to move data between registers and memory, as well as which banks should hold the value.
Speculative Execution

- General rules of upward code motion past a branch
  1. Must not cause an exception
  2. Must not overwrite the value of a register that is needed by some other successor of the branch
  3. Must not alter system memory

- Code past a branch is executed
  - If branch is taken any results after the branch are squashed
  - If branch is not taken then the results can be pushed to main registers

- General vs Boosting Hardware Model
  - General Model hardware provides non-trapping instructions to relax rule 1
  - Boosting provides full hardware support for speculative execution
    - Suppresses exceptions
    - Shadow Register File/ Shadow Store Buffer
  - Boosting doesn’t have significant gains since early stores are rare and rule 2 can often be avoided using software to do register renaming and loop induction variable optimization
Software Pipelining

Figure 6: Simple loop before and after software pipelining
The Efficeon processor developed by Transmeta was a 256-bit VLIW architecture. Efficeon was regarded as a processor with low power consumption and thermal footprint, as well as computational efficiency.

A 1.6GHz model was released in 2004 with an integrated Northbridge which lowered power consumption to compete with Intel's 1.6GHz Atom that only utilized an external northbridge.

The efficeon was also significantly smaller than other processors from the same era, only 60% of what the Pentium 4 was.

The Efficeon processor was used in Sharp and Sony laptops, as well as other desktops and tablets.
Future

- Streaming and media processing is a possible future for VLIW technology. This application holds large amounts of ILP with regular predictable branch prediction. Though superscalar processors dominate the market for now, VLIW has solid potential to migrate into main stream processors.
- VLIW will mainly be used for research and analysis
  - Ability to perform repetitive and intensive tasks very efficiently
- While a mainstream VLIW CPU may never come around, billions of VLIW chips are manufactured and sold every year by companies such as Qualcomm.
  - These are primarily DSP cores that will be utilized in devices such as televisions, blu-ray players, smartphones, and even a car’s computer
Questions?