Computer Science 703

Advance Computer Architecture

2006 Semester 2

Lecture Notes

25May06

VLIW & EPIC Architectures

James Goodman

Department of Computer Science

Background

Sources for this lecture:

- Intel Itanium Architecture Software Developer's Manual
- Mark Smotherman, "Understanding EPIC architectures and implementations", from ACM Southeast Conference, 2002

- Itanium Architecture (IPF : Itanium Processor Family)
- HP: Explicitly Parallel Instruction Computing (EPIC)

- History: came through HP: 2 separate histories
  - Bob Rau/Mike Schlanskar: Cydrome
  - Josh Fisher: Multiflow

VLIW, 1982

Josh Fisher, Yale University, Multiflow

- Want to group multiple instructions into a single "long instruction" that is executed on different functional units in parallel.
- Effect is very similar to a very long pipeline: branches (and cache misses) are deadly
- Ignored cache misses by ignoring caches
- Greatest innovation: *Trace scheduling*–capturing larger blocks of parallelism by predicting most likely path through multiple basic blocks, then adding fixup code where wrong branch was predicted.

Three steps for capturing ILP

1. Check dependencies between instructions to determine which instructions can be grouped together for parallel execution
2. Assign instructions to the functional units on the hardware
3. Determine when instruction begins execution

Tasks for ILP Execution

Each of these tasks can be performed at least partially at compile time

1. Compiler indicates which instructions can be executed concurrently (or hardware infers it from the order).
2. Compiler designates a functional unit for each instruction (or the hardware dynamically assigns a free one).
3. Compiler indicates exactly which instructions should be initiated in each cycle (or hardware assures that resources are/will be free and issues when ready).

Four Classes of Architecture

- VLIW : Compiler determines which instructions are assigned to which FU (a very long instruction word)
  - Highly restricted; implementation is architecture (# of functional units determines code!)
- Dynamic VLIW : Compiler does grouping, FU assignment; hardware determines execution time
  - Can respond to events that cannot be anticipated by compiler (like data caches)
- EPIC Compiler does grouping; FU assignment, initiation determined by hardware
  - Functional units dynamically scheduled, so architecture not tied to implementation
  - Still major benefit of compiling to specific implementation.
- Superscalar processors : all three done in hardware

Four Levels of Compiler Contribution

	Grouping	Fn unit asgn	Initiation
Superscalar	Hardware	Hardware	Hardware
EPIC	Compiler	Hardware	Hardware
Dynamic VLIW	Compiler	Compiler	Hardware
VLIW	Compiler	Compiler	Compiler

Table 1. Four Major Categories of ILP Architectures.

Mark Smotherman, "Understanding EPIC architectures and implemntations," Southeast ACM Conference 2002

Four Architectural Models

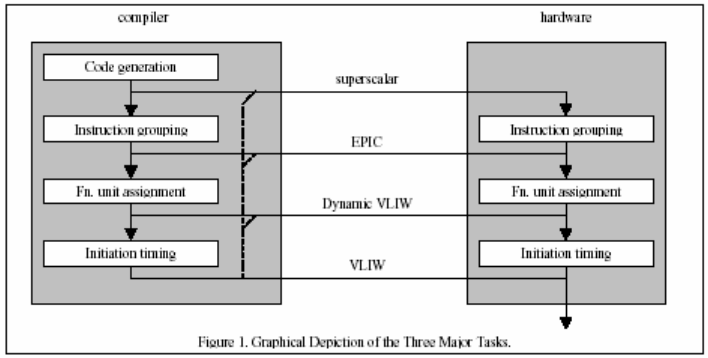


Figure 1. Graphical Depiction of the Three Major Tasks.

Mark Smotherman, "Understanding EPIC architectures and implemntations," Southeast ACM Conference 2002

- Example sequence: C = A + B

```
Load R1, A
Load R2, B
Add R3, R1, R2
Store C, R3
```

- Instructions 1 & 2 can be executed concurrently; 3 depends on 1 & 2; 4 depends on 3

VLIW Bundles

Ld/St unit 0	Ld/St unit 1	integer ALU	branch unit
`Load R1, A`	`Load R2, B`	nop	nop
nop	nop	nop	nop
nop	nop	`Add R3, R1, R2`	nop
`Store C, R3`	nop	nop	nop

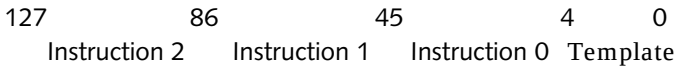
- Multiflow improved instruction size by compressing instructions to save space

EPIC Specification of Bundles

```
(2) Load R1, A
(1) Load R2, B
(1) Add R3, R1, R2
(.) Store C, R3
```

- Tag indicates distance to first dependent instruction
- Allows hardware to do FU assignment

IPF Format



Instructions are one of six types:

1. integer alu
2. non-alu integer
3. memory
4. floating-point
5. branch
6. extended

- Template tells which type is in which field (not all combinations allowed)

Five other features of EPIC Architectures

1. Predicated execution
2. Unbundled branches
3. Compiler control of the memory hierarchy
4. Control speculation
5. Data speculation

Predicated execution

- 64 Predicate registers indicate true or false
- Instruction execution is predicated on value
- Efficient implementation of short if-then-else without branches

Unbundled branches

- Branches consist of 3 parts:
  1. Branch decision
  2. Provide target address
  3. Transfer of control (PC)
- Within a bundle, multiple branch instructions can specify parallel tests, with multiple branch targets

Compiler hints to memory hierarchy

- Compiler can predict temporal locality quite well
- Provides hints:
  - Indicate temporal locality at L1
  - Indicate no temporal locality at L1
  - No temporal locality at L2
  - No temporal locality at all levels

Control speculation

- Hoist loads ahead of branches: if you didn't need it, not much lost
- Problem: what if load causes an exception?
- Solution: explicitly speculative load
  - Load causing exception returns tagged result (NaT: not a Thing or NaTVal: not a Thing Value for FP)
  - Speculation check instruction raises exception if NaT still around

Data speculation

- Hoist load instructions earlier
- Problem: aliasing: compiler often can't disambiguate pointers: how to avoid passing a store?
- Solution: Explicitly speculative load
  - Advanced Load Address Table (ALAT) has addresses
  - Followed by data-verifying load instruction
  - If store has occurred, data-verifying load re-executes load instruction

Two Variations of Check

Case 1: Check occurs before loaded value is used (ld.c)

- Load is repeated and execution continues

Case 2: Check occurs after loaded value has been used to generate other values (chk.a)

- If unsuccessful, chk.a branches to compiler-generated recovery code.

ALAT

4.4.5.1 Data Speculation Concepts

An ambiguous memory dependency is said to exist between a store (or any operation that may update memory state) and a load when it cannot be statically determined whether the load and store might access overlapping regions of memory. For convenience, a store that cannot be statically disambiguated relative to a particular load is said to be ambiguous relative to that load. In such cases, the compiler cannot change the order in which the load and store instructions were originally specified in the program. To overcome this scheduling limitation, a special kind of load instruction called an advanced load can be scheduled to execute earlier than one or more stores that are ambiguous relative to that load.