IA-64, P4 HT and Crusoe Architectures Ch 15

IA-64 General Organization
Predication, Speculation
Software Pipelining
Example: Itanium
Pentium 4 HT
Crusoe General Architecture
Emulated Precise Exceptions

General Organization

- EPIC - Explicit Parallel Instruction Computing
  - parallelism visible at instruction level, not "secrectly" implement in processor
    - new instruction stream semantics
  - compiler prevents many "hazards" (dependency problems), hardware can depend on it
- VLIW (Very Long Instruction Word)
- Branch predication – many speculative execution tracks
- Speculate on memory data loads

Fig 15.1

IA-64 General Organization

- 128 64-bit (+ Not a Thing bit) registers
  - integer, logical, general purpose
- 128 82-bit registers
  - floating point (IEEE double extended)
  - graphics
- 64 1-bit predicate registers
- 8 64-bit branch registers

f0: 0.0
f1: 1.0

Fig 15.1

Slide 9 [Lamb00]

Instruction Format

Slide 8 [Lamb00]

- Instruction (41 bits)
  - operation & predicates
  - up to 6 instruction executions in parallel
- Instruction bundle (128 bits)
  - three instructions & template
  - smallest unit to fetch instructions from memory
- Instruction group
  - machine instructions that could be issued in parallel
  - end of group marked with ";;" in symbolic assembly language code

Fig 15.2

Tbl 15.3

Predicated Execution

- Execute each branch
  - if-then-else gives two predicates, and each path will advance with its own predicate
- Predicate values known only after branch instruction completes
- Discard "wrong" path, commit "right" path
  - known always before commit time?

Fig 15.3 (a)

Slide 18 [Lamb00]

Speculative Loading, I.e., Control Speculation

- Start loading from memory in advance so that data is available earlier
  - load instruction "hoisted" earlier in code, before some branch instruction
  - interrupts are delayed (via NaT bit in register), and handled only at the time when they would have been handled normally

```
je  L2                    ld8.s  r1 = [r5]
ld8  r1 = [r5]     ⇨      je  L2
                          chk.s  r1, recovery
```

Fig 15.3 (b)

Slides 27, 28 [Lamb00]

Data Speculation

Slide 26 [Lamb00]

- Start loading from memory in advance so that data is available earlier
  - load instruction "hoisted" earlier in code, before a store instruction that might alter just that memory location
  - Advanced Load Address Table (ALAT, special hardware) keeps track of data speculation addresses
  - each store will clear target address in ALAT (if any)
  - at original load instruction time, a new load is initiated if ALAT entry was cleared

Slide 31 [Lamb00]

alias problem:

```
je L1                    ld8.a  r1 = [r5]
st8 [r3] = r13    ⇨      je L1
ld8  r1 = [r5]           st8 [r3] = r13
                         ld8.c r1 = [r5]
```

Slides 29-30 [Lamb00]

IA-64 Register Set

Fig 15.7

- 128 general purpose regs (stacked, rotated)
- 128 floating point regs (rotated)

Tbl 15.5

- 64 predicate regs
- 8 branch regs
- instruction pointer (bundle address)

Slides 15-17 [Lamb00]

Software Pipelining

- Unwrap loops in hardware, so that multiple iterations are done in parallel

code p. 559

  - code is not unrolled, but action is

code p. 560

  - each iteration done with different registers (automatic register renaming)

Slide 25 [Lamb00]

  - beginning and end of loop handled as special cases (with predicates)
  - each iteration execution is spread enough to make room for ILP

Fig 15.6

  - loop branches are replaced with special loop terminating instructions that control sw pipelining
  - why is this called software pipelining?

Itanium

Slide 40 [Lamb00]

- 1st implementation of IA-64 architecture
- "Simpler" than conventional superscalar
  - no reservation stations, reorder buffers
  - no large renamed register set for architecture registers
  - no dependency issue logic
  - dependencies solved by compiler, and explicitly solved in code
- Very large memory address space
  - explicit control over memory hierarchies
  - explicit memory op fences

Slides 10-12 [Lamb00]

Itanium

- Powerful cache hierarchy
  - split L1: 16KB + 16KB, 4-way set assoc, 32B lines
  - unified L2: 96KB, 6-way set assoc, 64B lines
  - off-chip unified L3: 4MB, 4-way set assoc
- TLB hierarchy

Slide 42 [Lamb00]

  - instruction TLB: 64 entry full assoc
  - data L1 TLB: 32 entry direct assoc
  - data L2 TLB: 96 entry full assoc
  - Hardware Page Walker – use mem hierarchy to locate address mapping
- 10-stage in-order pipeline

Slides 43-44 [Lamb00]

Itanium 2

- Upgraded cache hierarchy
  - split L1: 16KB + 16KB, 4-way set assoc, **64B** lines
  - unified L2: **256KB, 8-way** set assoc, **128B** lines
  - **on-chip** unified L3: **3MB**, **12-way** set assoc
- TLB hierarchy
  - instruction **L1 TLB**: 32 entry full assoc
  - instruction **L2 TLB**: 128 entry full assoc
  - data L1 TLB: 32 entry **full** assoc
  - data L2 TLB: **128** entry full assoc

Itanium 2

- Max 6 issues per cycle
  - 11 issue ports
- Many functional units, all fully pipelined
  - 6 general purpose ALU's
  - 4 data cache memory ports
  - 6 multimedia FU's
  - 4 FPU's
  - 3 branch units
- Perfect loop prediction
- Lots of branch prediction hints in code

IA-64 Summary

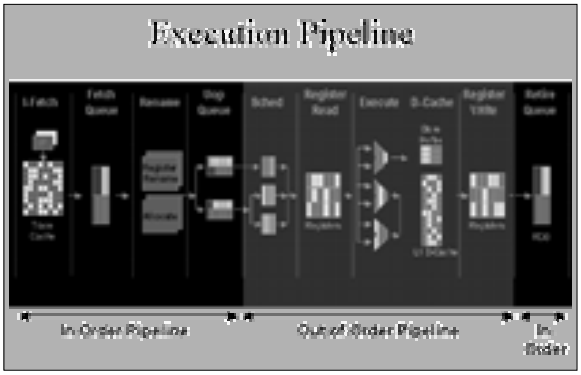
- Parallel semantics for ISA (Instr Set Arch)
- Lots of explicit ILP (Instr Level Parallelism)
- Memory hierarchy (cache) controls in ISA
- Memory synchronization primitives in ISA
  - normal access temporal locality hint (E.g., ifetch.t1) suggests to keep data in L1D, L2, and L3
  - less important hint (E.g., Fpload.nt1) suggests to keep data only in L2 and L3.

IA-64 Summary (contd)

- Lots of speculative work, that may be wasted
  - predicated execution
  - miss-prediction costs mostly avoided
  - branch prediction hints in ISA
  - load speculation: "hoist" loads above branch or store
- Large visible register set – no hidden rename regs
  - automatic stack frame save/restore
- HW-controlled software pipelining

Pentium 4 HT

- HT – Hyper-threading
- 2 logical processors in one physical prosessor
- OS sees it as symmetric 2-processor system
- Use wait cycles to run the other thread
  - memory accesses (cache miss)
  - dependencies, branch miss-predictions
- Utilize usually idle int-unit, when float unit in use
- 3.06 GHz + 24%(?)
  - GHz numbers alone are not so important
- 20 stage pipeline
- Fall 2003: CS dept new PCs with P4 HT processor



Crusoe Architecture

Major Ideas
General Architecture
Emulated Precise Exceptions
What to do with It

Background

- Transmeta Corporation
  - Paul Allen (Microsoft), George Soros (Soros Funds)
  - David R. Ditzel (Sun) Orig. CEO, now CTO
  - Edmund J. Kelly, Malcolm John Wing, Robert F. Cmelik
  - Linus B. Torvalds, February 1997 → 2003
- Patent 5832205
  - applied August 20, 1996
  - granted November 3, 1998
  - many (a few) other patents …
- Crusoe processor
  - published January 19, 2000

Basic Idea(s) (5)

faster

cheaper

- Create a new processor which, when coupled with "morph host" emulator, can run Intel/Windows code faster than state-of-the-art Intel processor, *or* with same speed but with less electric power
- New processor can be implemented with significantly fewer gates than competitive processors
- Compete with Intel, friendly with Microsoft
  - sell chip with emulator code to system manufacturers (Dell, IBM, Sun, etc etc)
- X86 (IA-32) binary is new binary standard
- Native OS not so important
  - services from target OS: E.g., Windows or Linux

Major General Ideas

- Emulation can be faster than direct execution
- TLB used to solve new problems
  - track memory accesses for memory mapped I/O
  - track memory accesses for self-modifying code
- Most of executed code generated "on-the fly"
  - not compiled before execution begins
  - extremely optimized dynamic code generation
- Optimized code allows for simpler machine
  - smaller, faster, uses less power?

Major General Ideas (contd)

- Self-modified code (dynamically created code) can be generated so that it is extremely optimized for execution
  - issue dependencies, reorder, reschedule problems solved at code generation (not in HW)
  - processor HW does not need to solve these
- Optimize for speed, but only when needed
  - do not optimize for speed when exact state change is required (this is the tricky part!)
- Alias detection to assist keeping global variables in registers

Major General Ideas (contd)

- NOT: faster and with less power

Class action suit (5.7.2001) ... stating that ... a revolutionary process that delivered longer battery life in Mobile Internet Computers while delivering high performance ....

Settled 13.3.2003 for 5.5 million dollars

http://www.lieffcabraser.com/transmeta.htm

Major Emulation Ideas

- Target processor (I.e., Intel processor) state kept in dedicated HW registers
  - working state ("speculated" state?), committed state
- Memory store buffer keeps uncommitted ("speculated") emulated memory state
- Specific instructions support emulation
  - commit, rollback (exact exceptions)
  - prot (aliases)
- TLB (and VM) designed to support emulation
  - A/N-bit (mem-mapped I/O), T-bit (self-mod. code)

General Architecture

- VLIW implementation
  - VLIW = Very Long Instruction Word
  - 4 simultaneous RISC instructions in "molecule"
    - one each of float, int, load/store, branch
  - large L3 Translation Cache for VLIW "molecules"
    - 8-16 MB
    - similar to Pentium 4 Trace Cache?
  - no circuitry for issue dependencies, reorder, optimize, reschedule
    - compiler takes care of these
    - data & structural dependencies under compiler control?

General Architecture (contd)

- Large register set
  - native regs: 64 INT, 32 FP
    - extra regs for renaming
  - target architecture regs: complete CPU state
    - INT, FP, control Reax, Recx, Rseq, Reip
    - working regs for normal emulation
    - committed regs for saving emulated processor state

General Architecture (contd)

- TLB
  - new features to solve new problems
    - before used to solve also memory protection problems in addition to plain VM address mapping
  - A/N-bit for memory-mapped I/O detection
    - trap to emulator, which creates precise code
    - memory-mapped I/O requires precise emulated processor state changes
  - T-bit for self-modifying code detection
    - trap to emulator, which recreates emulating code in instruction cache ("translation buffer")

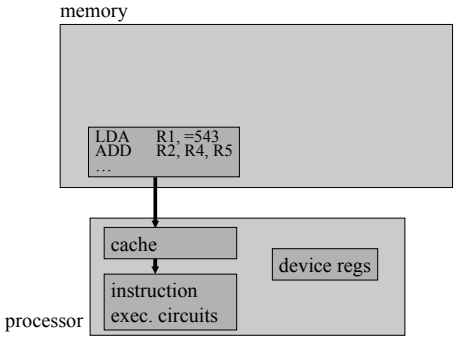
General Architecture (contd)

- Target memory store buffer
  - implemented with special register(s) to support emulation
  - keep track on which target processor memory stores are committed and which are not
  - uncommitted memory stores can be discarded at rollback
    - modify HW registers implementing it
    - commit & rollback controlled from outside of the processor, not internally as is usual with speculative instructions

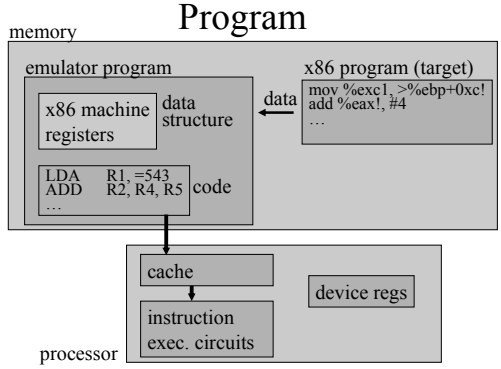
General Architecture (contd)

- RISC instruction set
  - explicitly parallel code (VLIW)
  - *commit* instruction supports emulation
    - commits emulated processor and memory state
    - use when coherent target processor (Intel) state!
  - *rollback* instruction (?) supports emulation
    - some or all of it can be in emulator code
    - recover latest committed emulated target register state
    - delete uncommitted writes from store buffer
    - retranslate emulation code for precise state changes
      - *commit* now after every emulated instruction?
  - *prot* instruction for alias detection

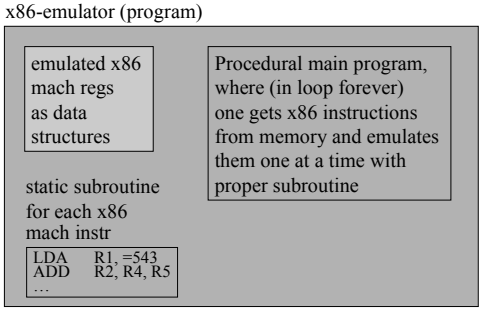
Ordinary Program Execution



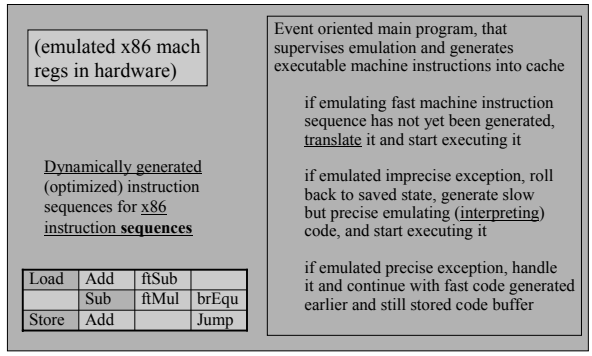
Execution of Ordinary Emulator Program



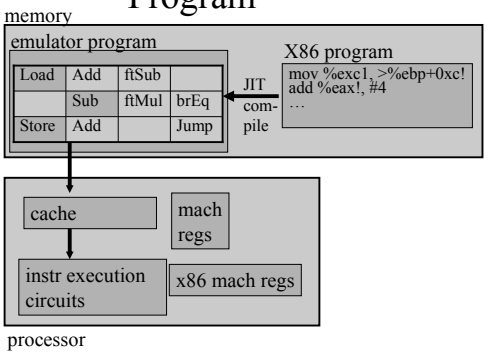
Ordinary Emulator



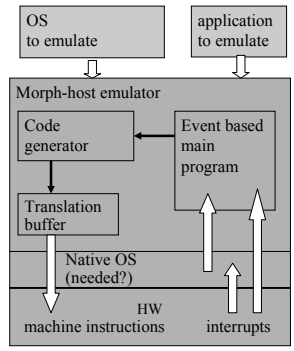
Crusoe Emulator



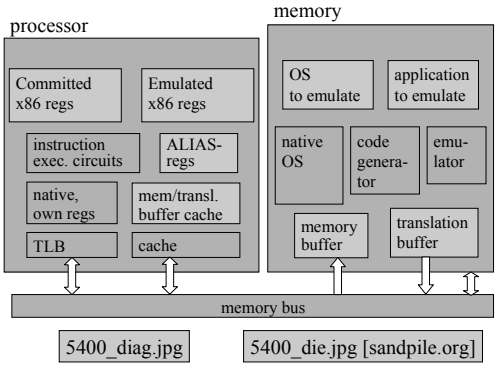
Execution of Crusoe Emulator Program



Crusoe Logical Structure



Crusoe Physical Structure



Crusoe Summary

- Emulation can be done faster or with less energy than the ”real thing”
- VLIW (EPIC?) core architecture
- Special HW to speed up emulation
  - x86 regs
  - memory-mapped I/O detection
  - alias and self-modifying code detection
- Special HW for precise interrupts
  - 2nd set of x86 regs
  - target memory store buffer
  - commit and rollback instruction in ISA

Crusoe Summary (contd)

- Complex overall structure
- ”Code Morphing Software”
  - JIT optimized code generation
  - compiler and interpreter resident in memory
  - fast but imprecise, or slow and precise emulation
- Optimize for speed or size (power, electricity)?
  - Small size ⇒ cheaper, less power

TM3200, TM5400, …, TM5600 low power
TM5800 high speed

Efficeon processor

- Follow-up for Crusoe
  - published 16.10.2003
  - Same manufacturing technology (.13 micron)
- More parallelism
  - 8 cells per molecule (Crusoe: 4 cells)
  - 256 bit molecule (Crusoe: 128 bits)
- Wait - you get more on the same chip
  - L1 data cache, L1 instr. cache
  - 1 MB L2 unified cache
  - DDR and AGP 4x graphics interface controllers
  - HyperTransport Bus Interface Controller
    - 12x speed as compared to PCI

-- IA-64 and Crusoe End --

”Aqua 3400 Portable Wireless Internet Access Device, Transmeta 400MHz, 8.4" TFT touch-screen”

”NEC Versa DayLite combines the power-saving 600 Mhz Crusoe TM5600 processor with dual battery systems that NEC claims will extend battery life to up to 7.5 hours on a single charge”