Itanium (old name: IA-64)

Stallings: Ch. 15
IA-64 overview
Predication, speculation
software pipelining
Itanium 2
Intel Multi-core and STI Cell

IA-64 overview
Predication, speculation
software pipelining

Itanium vs. Superscalar

SUPERSCALAR
Multiple parallel execution units
Remand and speculation instruction counts at run time
Branch prediction with speculative execution of one path
Loops that do not vary rapidly track guarded, and jump to body due to the order they
Execute roughly simultaneous but not in parallel

Itanium new design and architecture
Not backward compatible
HP and Intel co-operated

Instruction format

Fetch from memory one (or more) bundle at a time
Template (multiple, liaison)
Indicates what instructions can be executed parallel
Indicates what execution types each instruction slot needs

Template and Execution units

Template: info for parallel execution
Several bundles can be combined
More parallel execution
No need for NOP instructions
Up to 6 instructions per clock cycle (source: Itanium data sheet)

EPIC (Explicit Parallel Instruction Computing)

- Parallelism explicit in machine instructions, not hidden inside the hardware (processor)
- New semantics on machine instructions
- Compiler solves dependencies and decides the parallel execution issues, processor just trusts it
- VLIW (Very Long Instruction Word)
- Handle instructions in bundles (nippu)
- Branch predication, control speculation
- Speculative execution of both (all) branch targets
- Spekulative loading of data

Itanium Organization

- Lot of registers --- no renaming or dependency analysis
- Minimum 8 execution units
  - Function of the number of translators available on chip
  - GR registers associated with NaT-bit (Not a Thing)
  - Used in speculations

GR = General purpose or scratch registers
PR = Predication register or guard

Load/Store architecture

Comp. Org II, Spring 2009
Assembly-language format

(symbolinen konekieli)

[qp] mnemonic [.comps] dests = srcs

- qp    qualifying predicate register
- if predicate register value = 1 (true), commit
- mnemonic name of the instruction operation
- comps completers, separated by periods
- Some instructions have extra parts to qualify it
- dests destination operands, separated by commas
- srcs source operands, separated by commas

Instruction group boundaries (stops) are marked ;;
Instruction bundle template has the "black line"
Hint: the instructions of a group can be executed parallel
No data or output dependency within group
- no read after write (RaW) or
- no write after write (WaW)
What about antidependency (WaR)???

ld8 r1 = [r5] // first group
sub r6 = r8, r9 ;;
add r3 = r1, r4 // second group
st8 [r6] = r12              // memory address in r6

Predicated execution

A graph about the execution

Speculate:
- Both branches executed until the right one can be selected
- Selection known at the latest when the branch instruction (3) commits
Speculative loading

- Start data load in advance
  - speculative load (even if unclear if the data is needed)
  - Ready at processor when needed, no latency
- Straightforward unless branch or store between load and use
- Branching? – control speculation
  - Speculative loading could cause an exception (or page fault) that should not have happened at all
- Store? – data speculation
  - Speculative loading could be for the same memory location, the store is about to change

Control speculation

- Compiler adds the speculative load earlier in the code and chk.s instruction in its original place
- CPU delays the (possible) exception from speculative load to the chk.s instruction

Data speculation

- Hoist (nostal) load instruction earlier in the code before the branch instruction
- Mark it speculative (.s)
- If speculative load cause exception, delay it (NaT bit)
- There is a possibility that exception should not happen
- Add chk.s instruction to the original location. It checks for exceptions and starts recovery routine

Software pipeline (Ojhelmoitu liukuhihna)

- Hardware support to allow parallel execution of loop instructions
- Parallel execution can be achieved by executing instructions of different iteration cycles
- Each iteration cycle uses different registers
- Automatic register renaming
- Prolog (alku) and epilog (loppu) are special cases handled by rotating predicate register
- "Loop jump" replaced by special loop termination instr. that controls the pipeline
- Rotate registers, decrease loop count
mov lc = 199 // set loop count register
mov pr16 = 1 // set epilog count register
mov pr16 = 0 // empty usage

L1: (p16) ld5 r32 = [r5], 4 // cycle 0
     (p17) --- // empty stage
     (p18) add r35 = r34, r9 // cycle 0
     (p19) st4 [r6] = r36, 4 // cycle 0
     br.ctop L1 ;; // cycle 0

Koodi p.555

Application register set
(Sovelluksen rekisterit)

General registers (128), FP-registers (128), Predicates (64)
Some static, some rotational (automatic renaming by hw)
Some general registers used a stack (pino)
Branch registers (8)
Target address can be in a register (indirect jump!)
Subroutine return address normally stored in register br0
If ner call before return, br0 stored in register stack
Instruction pointer
Bundle address of current instruction
Reg address of single instruction
User mask
Flags (single-bit values) for traps and performance monitoring
Performance monitor data registers
Supports monitoring hardware
Information about hardware, e.g. branch predictions, usage of register stack, memory access delays, ...

Rekisteripino, Register Stack Engine

r0..r31 for global variables
r32..r127 (total 96) for subroutine calls
Call reserves a frame (set of regs in a register window)
• parameters (inputs/outputs) + local variables
• Size set dynamically (alloct instruction)
Registers automatically renamed after the call
Subroutine parameters always start from register r32
Allocated in a circular-buffer fashion (renkaana)
If area full, hardware moves register contents of oldest frame to memory (= backing store). Restored when subroutine returns
Memory address in register BSP, BSPSTORE
(backup store pointer)
Procedure Call and Return

Application registers

Itanium 2
(again just called Itanium!)

Itanium

Wide and fast bus: 128b, 6.4 Gbps
Improved cache hierarchy
- L1: split instr, data 16K + 16K, set-ass. (4-way), 64B line
- L2: shared 256KB, set-ass. (8-way), 128B line
- L3: shared, 9MB, set-ass. (12-way), 64B line
- A4 on-chip, smaller latencies
- TLB hierarchy
  - t-TLB L1: 32 hams, associative
  - L2: 128 items, associative
  - D-TLB L1: 32 hams, associative
  - L2: 128 items, associative

Memory management

- Memory hierarchy visible to applications also
- Fetch order: make sure, that earlier ops have committed
- Locality: fetch a lot a little lines to cache
- Prefetch: when moved closer to CPU
- Clearing: line invalidation, writes to cache
- Implicit control (exclusive access)
- Switching memory and register content
- Increasing memory content by a constant value
- Possibility to collect performance data
- To improve hints...

Computer Organization II

Itanium

- First implementation released in 2001
- Second, at that time called Itanium 2, released in 2002
- Simpler than conventional superscalar CPU
  - No resource reservation stations
  - No reorganization buffers (ROB)
  - Simpler register remapping hardware (versus register aliasing)
  - No dependency-detection logic
  - Compiler solved dependences and created explicit parallelism directives
- Large address space (suuri osoiteavaruus)
  - Smallest addressable unit: 1, 2, 4, 8, 10, 16 bytes
  - Recommendation: use natural boundaries
- Support both: Big-endian and Little-endian
Itanium

- 11 instruction issue port (like selection window)
- Max 6 instructions to execution in each cycle
- In-order issue, out-of-order completion
- 8-stage pipeline
- More execution units (22):
  - 6 general purpose ALU's (1 cycle)
  - 6 multimedia units (2 cycles)
  - 3 FPU's (4 cycles)
  - 3 branch units
- 4 data cache memory ports (L1: 1/2 cycle load)
- Improved branch prediction
- Application is allowed to give hints
- Used to reduce cache miss

Itanium Processor

- Improved branch prediction
- Application is allowed to give hints
- Used to reduce cache miss

Current State (2006-08)

- Intel hyper-thread and multi-core
- STI multi-core

Intel Pentium 4 HT (IA-32)

- HT - Hyper-threading
- 2 logical processors in one physical processor
- 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% (7)
- GHz numbers alone are not so important
- 20 stage pipeline
- Dual-core hyper-thread processor
- Dual-core Itanium-2 with Hyper-threading

Intel Multi-Core Core-Architecture

- 2 or more (> 100?) complete cores in one chip
- Hyper-threading still in use
- Simpler structure, less power
- Private L1 cache
- Private or shared L2 cache?
- Intel Core 2 Duo E6700
  - 128-bit data path
  - Private 32 KB L1 data cache
  - Private 32 KB L1 instr. Cache (for micro-ops)
  - Shared/private
  - 4 MB L2 data cache

STI Cell Broadband Engine

- Several dedicated vector-processing engines (SPE), controlled by one main, general purpose processor (PPE)
  - 1 PowerPC PPE
  - Power Processing Element
  - RISC, 2 hyper-threads, in-order, simple prediction logic (needs compiler support)
  - 32 KB L1 data and instr. caches
  - 256 KB L2 cache
  - "normal programs"
  - 8 SPE's
  - Synergistic Processor Elements
  - 256 KB local data/instr memory, no cache
  - Receive/execute packets from off-chip main memory (as DMA transfer)
  - 128 registers of 128 bit, 64 GB/s
  - 2 pipelines: even, odd
  - No branch prediction, "Branch hint" instr.

STI Cell (Cell B.E.)

- Sony
  - Playstation 3 (4 cells)
- IBM
  - Roadrunner supercomputer
    - (installed 2008)
    - 1$10M, 1100 m², Linux
    - Peak 1.4 petaflops (1.6 * 10^15 flops)
    - Sustained 1 petaflops
    - Over 16000 AMD Opteron for file ops and communication (e.g.)
    - Normal servers
    - Over 16000 Cells for number crunching
    - Blade centers

STI Cell Broadband Engine

- Programming Models for SPE use
  - Function offload Model
    - Run some functions at SPE's
  - Device Extension Model
    - SPE as front-end for some device
  - Computational Acceleration Model
    - SPE's do most of computation
  - Streaming Models
    - Data flow from SPE to SPE
  - Shared-mem multiprocessor Model
    - Local store as cache
    - Cache coherent shared memory
  - Asymmetric Thread Runtime Model

- Normal programs
  - 8 SPE's
  - Synergistic Processor Elements
  - 256 KB local data/instr memory, no cache
  - Receive/execute packets from off-chip main memory (as DMA transfer)
  - 128 registers of 128 bit, 64 GB/s
  - 2 pipelines: even, odd
  - No branch prediction, "Branch hint" instr.
Computer Organization II

ARM -

ARM architecture
- ARM architecture family
  - 32-bit embedded RISC microprocessor
  - ARM7 family accounts for approximately 90% of these.
  - The most widely used 32-bit architecture in the world.
  - The ARM architecture is used in about 3/4 of all 32 bit
    processors sold. (source: Intellitech)
- Architecture
  - Extremely simple (ARM6 only 35,000 transistors)
  - 32-bit data bus, a 26-bit (64 Mbyte) address space and
    sixteen 32-bit registers.
  - Low power usage, hardwired control
  - ARM: no cache, ARM4: cache
  - ARM8: 5-stage pipeline, static branch prediction, double-
    bandwidth memory

ARM architecture
- Conditional execution of most instructions
  - 4-bit condition code in front of every instruction
  - Arithmetic instructions alter condition codes only when desired
- Indexed addressing modes
- 2-priority-level interrupt subsystem

while (i != j)
{
  if (i > j)
    i -= j;
  else j -= i;
}

loop CMP Ri, Rj ; set condition "NE" if (i != j)
  ; "GT" if (i > j),
  ; or "LT" if (i < j)
SUBGT Ri, Rj, Rj ; if "GT", i = i-j;
SUBLT Ri, Rj, Ri  ; if "LT", j = j-i;
BNE loop             ; if "NE", then loop

Things progress...
- X86 => Pentium => Core => Nehalem (Core i7) => Westmere
  - Superscalar
    - More efficient use of pipelining
    - Parallel pipelines
    - Branch prediction
    - Out-of-order execution
    - CISC => RISC translations
    - Hyperthreading
  - Chip-level multiprocessing -> multi-core
  - Vector instruction codes (in vector processors)
    - Parallel data processing
  - Cache: more levels, larger cache
    - OX9650: 12 MB L2

To different directions...
- Power consumption (Virrankulutus)
  - Mobile and portable devices
  - Density => heating up
- Superscalar improvements used?
  - Improvements prediction logic gives less and less benefit =>
    simpler CPU
  - => software based (Transmeta Crusoe tried this)
  - => compiler does and gives better ordered instructions (IA-
    64, Itanium2, CELL, ..)
- More cores on one chip
  - Different tasks (like Westmere integrated GPU)
- Coordination of the cores (processors) becomes an issue

Review Questions / Kertauskysymyksiä
- EPIC
- Why does the instruction bundle have a template?
- What is predicated execution? How does it work?
- What means control speculation? Data speculation?
- How registers are used in subroutine calls?
- Difference of hyper-threading and multi-core?