Intel/HP EPIC/IA-64 Architecture

55:132/22C:160
High Performance Computer Architecture

Intel/HP EPIC/IA-64 Architecture

- EPIC (Explicitly Parallel Instruction Computing)
  - An ISA philosophy/approach

    e.g. CISC, RISC, VLIW
  - Very closely related to but not the same as VLIW
- IA-64
  - An ISA definition

    e.g. IA-32 (was called x86), PA-RISC
  - Intel's new 64-bit ISA
  - An EPIC type ISA
- Itanium (was code named Merced)
  - A processor implementation of an ISA

    e.g. P6, PA8500
  - The first implementation of the IA-64 ISA

IA-64 Architecture

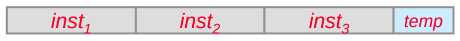
- 128 general-purpose registers
- 128 floating-point registers
- Arbitrary number of functional units
- Arbitrary latencies on the functional units
- Arbitrary number of memory ports
- Arbitrary implementation of the memory hierarchy

Needs retargetable compiler and recompilation to achieve maximum program performance on different IA-64 implementations

IA-64 Instruction Format

- IA-64 "Bundle"
  - Total of 128 bits
  - Contains three IA-64 instructions *(aka syllables)*
  - Template bits in each bundle specify dependencies both within a bundle as well as between sequential bundles
  - A collection of independent bundles forms a "group"

    A more efficient and flexible way to encode ILP then a fixed VLIW format



- IA-64 Instruction
  - Fixed-length 40 bits long
  - Contains three 7-bit register specifiers
  - Contains a 6-bit field for specifying one of the 64 one-bit predicate registers

IA-64 EPIC vs. Classic VLIW

- Similarities:
  - Compiler generated wide instructions
  - Static detection of dependencies
  - ILP encoded in the binary (a group)
  - Large number of architected registers
- Differences:
  - Instructions in a bundle can have dependencies
  - Hardware interlock between dependent instructions
  - Accommodates varying number of functional units and latencies
  - Allows dynamic scheduling and functional unit binding

    Static scheduling are "suggestive" rather than absolute

  ⇒Code compatibility across generations

    but software won't run at top speed until it is recompiled so "shrink-wrap binary" might need to include multiple builds

Additional Features of IA64

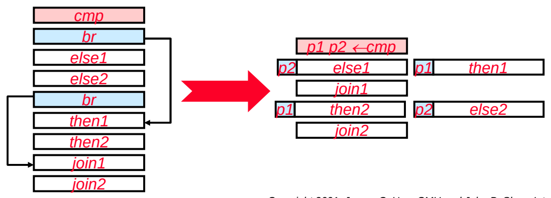
- Predicated execution
- Speculative, non-faulting Load instruction
- Software-assisted branch prediction
- Register stack
- Rotating register frame
- Software-assisted memory hierarchy

See "Understanding the IA-64 Architecture" by G. Doshi, Intel

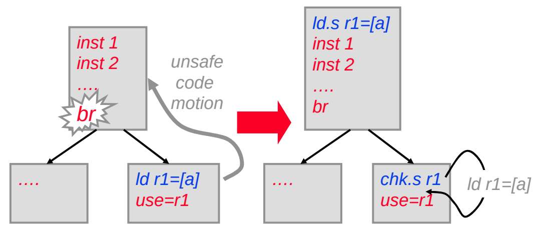
Predicated Execution

- Each instruction can be separately predicated
- 64 one-bit predicate registers

  Each instruction carries a 6-bit predicate field
- An instruction is effectively a NOP if its predicate is false
- Assumes IA-64 processors have lots of spare resources
- *Converts control flow into dataflow*



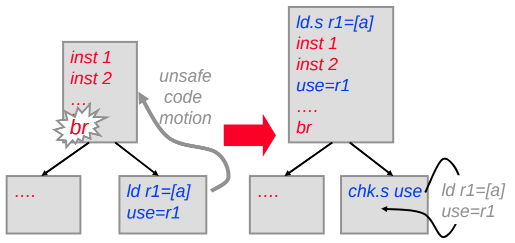
Speculative, Non-Faulting Load



- *ld.s* fetches *speculatively* from memory

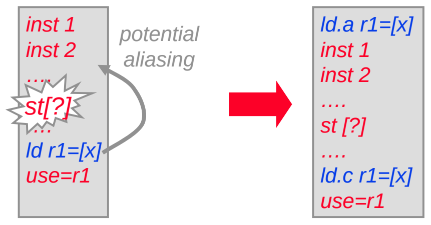
  i.e. any exception due to *ld.s* is suppressed
- If *ld.s r* did not cause an exception then *chk.s r* is an NOP, else a branch is taken (to some compensation code)

Speculative, Non-Faulting Load



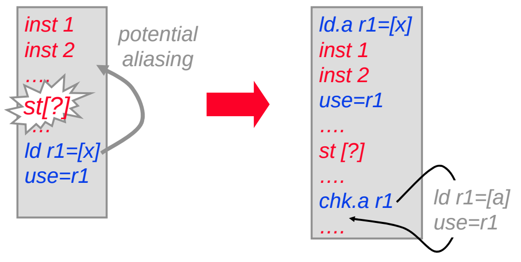
- Speculative load data can be consumed prior to check
- “speculation” status is propagated with speculated data
- Any instruction that uses a speculative result also becomes speculative itself (i.e. suppressed exceptions)
- *chk.s* checks the entire dataflow sequence for exceptions

Speculative “Advanced” Load



- *ld.a* starts the monitoring of any store to the same address as the advanced load
- If no aliasing has occurred since *ld.a*, *ld.c* is a NOP
- If aliasing has occurred, *ld.c* re-loads from memory

Using Speculative Load Results

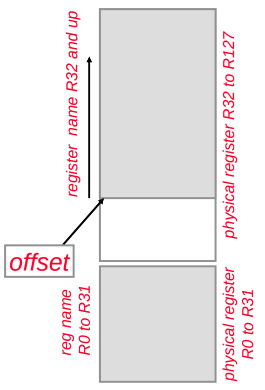


Branch Prediction

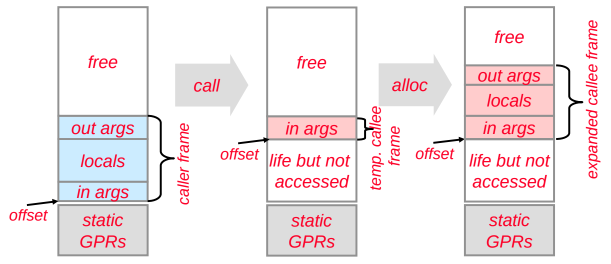
- Static branch hints can be encoded with every branch
  - taken vs. not-taken
  - *whether to allocate an entry in the dynamic BP hardware*
- SW and HW have joint control of BP hardware
  - “brp” (branch prediction) instruction can be issued ahead of the actual branch to preset the contents of BPT and BTAC
  - *Itanium uses a 512-entry 2-level BPT and 64-entry BTAC*
- TAR (Target Address Register)
  - a small, fully-associative BTAC-like structure
  - contents are controlled entirely by a “prepare-to-branch” inst.
  - a hit in TAR overrides all other predictions
- RSB (Return Address Stack)
  - Procedure return addr is pushed (or popped) when a procedure is called (or when it returns)
  - Predicts nPC when executing register-indirect branches

Register Renaming

- 128 general purpose physical integer registers
- Register names R0 to R31 are static and refer to the first 32 physical GPRs
- Register names R32 to R127 are known as “rotating registers” and are renamed onto the remaining 96 physical registers by an offset
- Remapping wraps around the rotating registers such that when offset is non-zero, physical location of R127 is just below R32



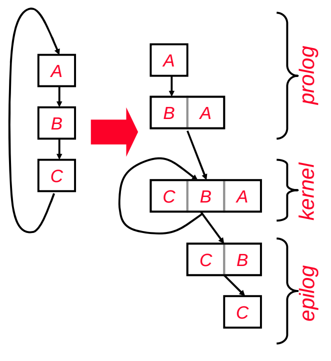
Register Stack for Procedure Calls



- On a procedure call, the rename offset is bumped to the beginning of output argument registers
- Callee can then allocate its own working frame (up to 96 regs)
- If there isn’t enough free regs to be allocated, HW automatically frees up space by *spilling* life contents not in the current frame to memory

Register stack appears infinite to SW

Rotating Loop Frames for Loop Pipelining



Suppose B_i is is only data dependent (through data stored in registers) on A_i; and C_i only on B_i

- The “pipelined” kernel block (containing independent computation from C_i, B_{i+1} and A_{i+2}) potentially has better ILP

What happens if C_i is also data dependent on A_i

- The result placed in register by A gets clobbered by the next execution of A (in the next cycle) before C can use it two cycles from now

Nice Loop Pipelining Example

```
    i=0
    while (i<99) {
        ;; a[ i ]=a[ i ]/10
A       Rx = a[ i ]
B       Ry = Rx  / 10
C       a[ i ] = Ry
        i++
    }
```

```
i=0
while (i<99) {
    Rx = a[ i ]
    Ry = Rx  / 10
    a[ i ] = Ry

    Rx = a[ i+1 ]
    Ry = Rx  / 10
    a[ i+1 ] = Ry

    Rx = a[ i+2 ]
    Ry = Rx  / 10
    a[ i+2 ] = Ry

    i=i+3
}
```

```
i=0
Ry=a[ 0 ] / 10
Rx=a[ 1 ]

while (i<97) {
    a[i]=Ry          C

    Ry=Rx / 10       B

    Rx=a[i+2]        A

    i++
}

a[97]=Ry
a[98]=Rx / 10
```

Loop Pipelining Requiring Renaming

```
   i=0
   while (i<99) {
        ;; a[ i ]=a[ i ]/10+a[ i ]
A       Rx = a[ i ]
B       Ry = Rx  / 10
C       a[ i ] = Ry+Rx
        i++
   }
```

```
i=0
while (i<99) {
      Rx = a[ i ]
      Ry = Rx  / 10
      a[ i ] = Ry+Rx
WAR
      Rx = a[ i+1 ]
      Ry = Rx  / 10
      a[ i+1 ] = Ry+Rx
WAR
      Rx = a[ i+2 ]
      Ry = Rx  / 10
      a[ i+2 ] = Ry+Rx

      i=i+3
}
```

```
i=0
Ry=a[ 0 ] / 10
Rx=a[ 1 ]

while (i<97) {
      a[i]=Ry+Rx'

      Ry=Rx / 10

      Rx'=Rx
      Rx=a[i+2]

      i++
}

a[97]=Ry + Rx'
a[98]=Rx / 10 + Rx
```

Renaming with Rotating Registers

```
i=0
Ry=a[ 0 ] / 10
Rx=a[ 1 ]

while (i<97) {
      a[i]=Ry+Rx'

      Ry=Rx / 10

      Rx'=Rx
      Rx=a[i+2]

      i++
}

a[97]=Ry + Rx'
a[98]=Rx / 10 + Rx
```

```
i= -2

while (i<99) {
      pred(i>-1):
            a[i]=Ry+RR(x-2)

      pred(i>-2 && i<98):
            Ry=RR(x-1) / 10

      pred(i<97):
            RR(x)=a[i+2]

      `increase RR offset by 1'
      i++
}
```

Itanium Specifics

- 6-wide 10-stage pipeline
- Fetch 2 bundles per cycle with the help of BP into a 8-bundle deep fetch queue
- 512-entry 2-level BPT, 64-entry BTAC, 4 TAR, and a RSB
- Issue up to 2 bundles per cycle some mixes of 6 instructions
  e.g. (MFI,MFI) or (MIB,MIB$_h$)
- Can issue as little as one syllable per cycle on RAW hazard interlock or structural hazard (scoreboard for RAW detection)
- 8R-6W 128 Entry Int. GPR, 128 82-bit FPR, 64 predicate reg's
- 4 globally-bypassed single-cycle integer ALUs with MMX,
  2 FMACs, 2 LSUs, 3 BUs
- *Can execute IA-32 software directly*
- Intended for high-end server and workstations

Itanium Performance

	SPEC2000	
system	int	fp
hp workstation i2000 800MHz, 2-way, 2MB cache, Windows XP		658
hp workstation i2000 800MHz, 1-way, 2MB cache, HP-UX	365*	610*
hp workstation i2000 733MHz, 1-way, 2MB cache, Windows XP		623
hp workstation i2000 733MHz, 1-way, 2MB cache, HP-UX	335*	577*

How does it compare?

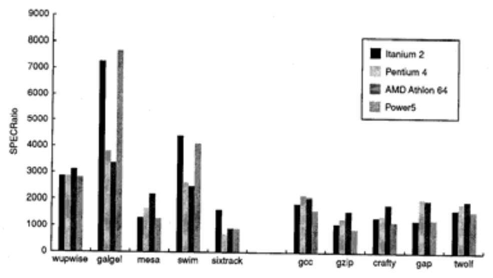
	SPEC_rate2000 (peak)	
system	int	fp
hp workstation x4000 uni-processor	6.36	6.43
hp workstation x4000 dual processor	11.70	10.50

FYI

	Alpha 21264	AMD Athlon	Intel P4	MIPS R12000	IBM Power3	HP PA-8600	SUN Ultra-III
Clock (MHz)	833	1200	1500	400	450	552	900
SPECint 2000	518		524	320	286	417	438
SPECfp 2000	590	304	549	319	356	400	427

Microprocessor Report, December 2000

Preformance : Itanium 2 vs. Superscalar



Source: Hennessey and Patterson, *Computer Architecture, A Quantitative Approach, Fourth Edition*, Appendix G