ITANIUM

An EPIC Architecture

Marco Barcella
Karthik Sankaranarayanan
Ganesh Pai

Introduction

- EPIC: Explicitly Parallel Instruction Computing
- Combination of features of RISC and VLIW
- VLIW features and flaws
  - Groups of independent instructions
  - Simple hardware
  - Exploit ILP with compiler
  - Large increase in code size
  - Blocking caches

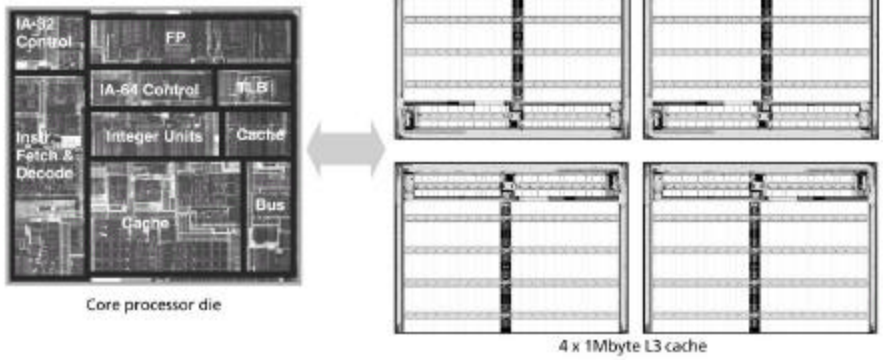
Introduction

- 733 - 800 MHz clock
  0.18-micron CMOS process technology
- 2 extended, 2 single precision FMACs
  Execution up to 8 SP flops/cycle - 6 GFLOP
- >20x Pentium Pro
- 3-level cache hierarchy
  - Split L1 and Unified L2 on die
  - Unified L3 on separate die but same container

Introduction

- 64-byte line size
- Page Sizes up to 256MB
- Full 64-Bit computing
- Full IA- 32 binary compatibility in hardware
  - Shared Resources: ALU, registers, Data Cache
  - IA-32 Engine: Dynamic execution
- Instruction set architecture (Marco)
  Instruction stream (Ganesh)
  Data stream and IA-32 Compatibility (Karthik)

Die Plot



Instruction Set Architecture

The Software Interface

Marco Barcella

Outline

- Introduction to the ISA
- Expressing parallelism
- Creating parallelism
- Techniques and instructions
- Compatibility
- Observations

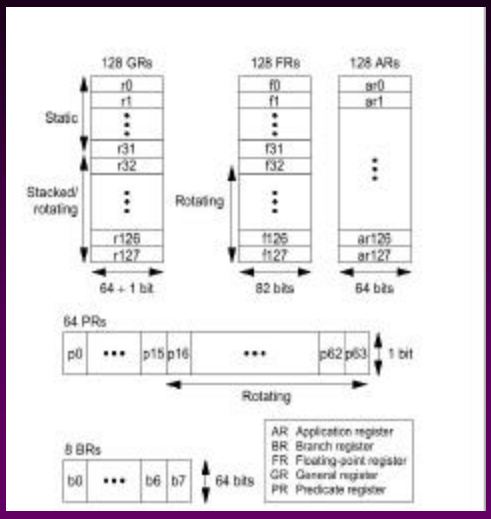
Why & How

- Goal
  - Bring ILP features to a general purpose microprocessor, flexibility
- Techniques
  - Predication
  - Speculation
  - Large register files
  - register rotation
  - HW exception deferral
  - Software pipelining
- RISC/CISC basic architecture of HP's PA-RISC, but …

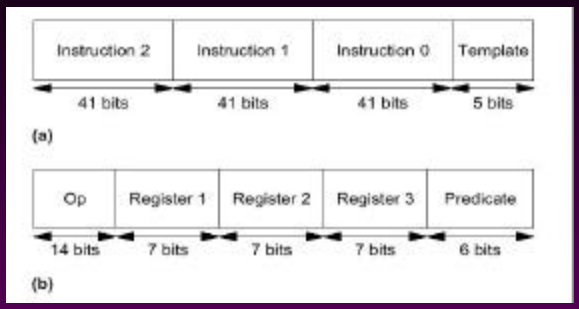
Register Resources

- NaT
- Predicate Registers
- Branch Registers

- AR: PFS(PFM, PEC)
  - UM
  - 8 Kernel Registers
  - LC, EC, CCV
  - AR 16-19
  - Future definition



Encoding



- Bundles: More than one per cycle
- Template: MII, MIB – other combinations
- Compiler based reordering
- No Register analysis
- Instruction compared to 32-bit

Instructions

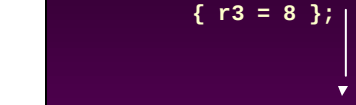
- 6 types, 4units
- L+ X : Long branches, long immediate integer

Instruction Type	Description	Execution Unit Type
A	Integer ALU	I-unit or M-unit
I	Non-ALU integer	I-unit
M	Memory	M-unit
F	Floating-point	F-unit
B	Branch	B-unit
L+X	Extended	I-unit/B-unit

Expressing Parallelism

- Not only bundles, but also
  - Compound Conditionals

```
If ((a==0)|| (b<=5) ||
   (c!=d) || (f & 0x2)
{ r3 = 8 };
```



```
cmp.ne p1 = r0, r0;
add t = -5, b;;

cmp.eq.or p1 = 0,a
cmp.ge.or p1 = 0,t
cmp.ne.or p1 = 0,d
tbit.or p1 = 1,f,1;;

(p1) mov r3 = 8
```

  - Multi-way branches

```
{ .mii
   cmp.ne p1,p2 = r1,r2;
   cmp.ne p3,p4 = 4, r5;
   cmp.lt p5,p6 = r8,r9;
}
{ .bbb
  (p1) br.cond label1
  (p3) br.cond label2
  (p5) br.call b4 = label3
}

// Fall through code here
```

Creating Parallelism

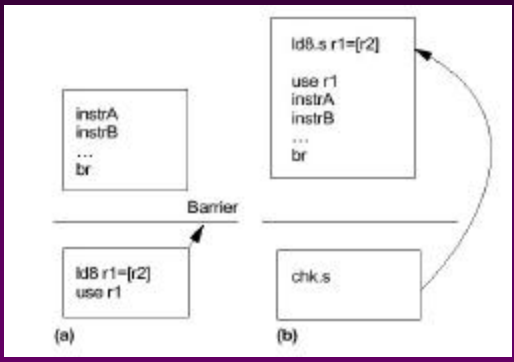
- Predication
  - Uses CMP instructions and predicate registers
  - Converts control dependencies to data dependencies
  - Motivation

```
if (r1==r2)                    cmp.eq p1,p2 = r1, r2;;
    r9 = r10 – r11;
else                  ——————→  (p1) sub r9 = r10, r11
    r5 = r6 + r7;              (p2) add r5 = r6, r7
```

- Speculation + Predication
  - Basic blocks in a single group
  - Barriers between basic blocks
  - Compiler

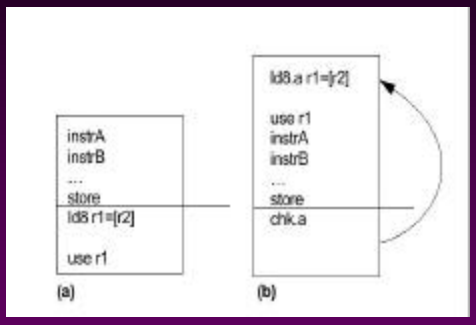
Control Speculation

- Importance of loads
- ld.s and chk.s and handling exceptions
- Propagation of token and fix-up



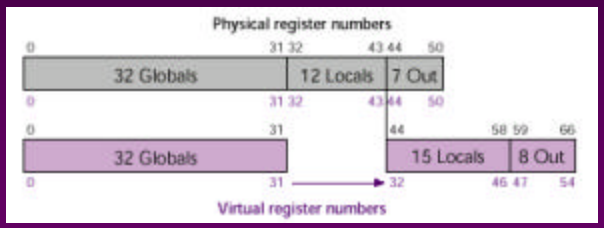
Data Speculation

- Ambiguous dependencies, ld.a
- How it works
  - ALAT, two tags
- Two recoveries
  - ld.c, ldf.c, ldfp.c
  - chk.a (chk.s)



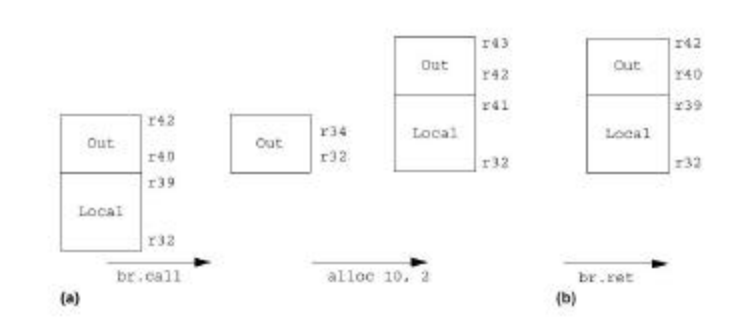
Procedure Calls

- Criticism: Large registers
- GR: 32 static + 96 stack
- Frames(SPARC), local, output
- br.call, brl.call & then br.ret
  - CFM in PFM (PFS), RRB, alloc (sof, sol)



Procedure Calls

- RSE speculatively fills and spills in the background
- Result: Vs. PA-RISC 30%, 5% (Database)



Context Switch Instructions

- Specific control on stack and backing store
  - Flushrs to spill previous stack frames
  - Cover to create a new frame above
  - Ladrs to fill from backing store

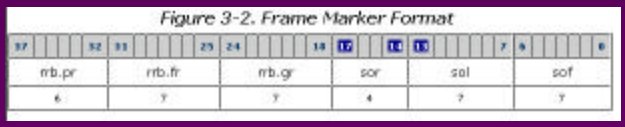
Branch Instruction

- Three categories
  - IP-relative (21 bit) ; Long (60 bit) ; Indirect (in BRs)

Table 4-23. State Relating to Branching

Register	Function
BRs	Branch registers
PRs	Predicate registers
CFM	Current Frame Marker
PFS	Previous Function State application register
LC	Loop Count application register
EC	Epilog Count application register

Figure 3-2. Frame Marker Format



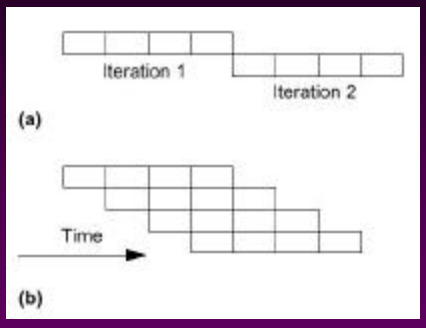
Branch Instructions

Table 4-22. Branch Types

Mnemonic	Function	Branch Condition	Target Address
br.cond or br	Conditional branch	Qualifying predicate	IP-rel or Indirect
br.call	Conditional procedure call	Qualifying predicate	IP-rel or Indirect
br.ret	Conditional procedure return	Qualifying predicate	Indirect
br.ia	Invoke the IA-32 instruction set	Unconditional	Indirect
br.cloop	Counted loop branch	Loop count	IP-rel
br.ctop, br.cexit	Modulo-scheduled counted loop	Loop count and Epilog count	IP-rel
br.wtop, br.wexit	Modulo-scheduled while loop	Qualifying predicate and Epilog count	IP-rel
brl.cond or brl	Long conditional branch	Qualifying predicate	IP-rel
brl.call	Long conditional procedure call	Qualifying predicate	IP-rel

Software Pipelining

- Motivation
- Vs HW
- Parallelism
- 3 phases
- Rotating FR, PR
- LC, EC



Software Pipelining

- 2 categories
  - Counted,
  - While (top, exit)
- Counted
  - Ends with EC=1 and LC=0, no qualifying predicate
- While
  - No LC, ends when QP=0 and EC=1

Branch Prediction Hints

- Hints, Branch Predict Instructions (brp)
- Hints:
  - strategy

Table 4-26. Whether Prediction Hint on Branches

Completer	Strategy	Operation
.spnt	Static Not-Taken	Ignore this branch, do not allocate prediction resources for this branch.
.sptk	Static Taken	Always predict taken, do not allocate prediction resources for this branch.
.dpnt	Dynamic Not-Taken	Use dynamic prediction hardware. If no dynamic history information exists for this branch, predict not-taken.
.dptk	Dynamic Taken	Use dynamic prediction hardware. If no dynamic history information exists for this branch, predict taken.

Branch Prediction Hints

- Prefetch

Table 4-27. Sequential Prefetch Hint on Branches

Completer	Sequential Prefetch Hint	Operation
.few	Prefetch few lines	When prefetching code at the branch target, stop prefetching after a few (implementation-dependent number of) lines.
.many	Prefetch many lines	When prefetching code at the branch target, prefetch more lines (also an implementation-dependent number).

- Deallocate

Table 4-28. Predictor Deallocation Hint

Completer	Operation
.none	Don't deallocate
.clr	Deallocate branch information

Branch Prediction Hints

- Branch prediction instructions
  - LOCATION
  - TARGET
  - IMPORTANCE
  - STRATEGY

Memory Instructions

- Simple (GR or FR, memory access order)
- Variants for speculative, spilling
- Semaphore instructions

Memory Instructions

Table 4-12. Memory Access Instructions

Mnemonic			Operation
General	Floating-point		
	Normal	Load Pair	
ld	ldf	ldfp	Load
ld.s	ldf.s	ldfp.s	Speculative load
ld.a	ldf.a	ldfp.a	Advanced load
ld.sa	ldf.sa	ldfp.sa	Speculative advanced load
ld.c.nc, ld.c.clr	ldf.c.nc, ldf.c.clr	ldfp.c.nc, ldfp.c.clr	Check load
ld.c.clr.acq			Ordered check load
ld.acq			Ordered load
ld.bias			Biased load
ld.fill	ldf.fill		Register fill
st	stf		Store
st.rel			Ordered store
st.spill	stf.spill		Register spill
cmpxchg			Compare and exchange
xchg			Exchange memory and GR
fetchadd			Fetch and add

Integer and Shifting

- Add, add…1, addp (32bit)
- Shift Left Mask Merge: dep, dep.z
  - Position and field by immediate
  - Simple shl (amount)

Compare Instructions

- Two predicate registers
- Deferred token (tnat)
- 5 types
  - Normal,
  - Unconditional
  - 3 “parallel” compares

Floating Point Architecture

- FSR: precision modes, 4 status fields
- All with FMAC= A*B+C: simple,divide
- XMA
- 82 bits: 2+ 32(if single), 64(double), 80(double extended)
- Two singles in one register

Compatibility

- X86: direct execution
- BR.IA, JMPE, overhead of register set saving
- SSE included (128), “new media”
- MMX parallel arithmetic: 128 not 8
- HP dynamic translator
- CMP4

Code Density

- Causes
  - Avg. 43 bit (32 of RISC)
  - Added (alloc, chk)
  - Fix-up
- Biggest impact
  - Decreasing hit rate on caches

Observations

- Synergetic
  - ld.sa, data dependences in software pipelining
- Compiler
  - Template
  - Grouping
  - Explicit prefetching
  - ld.a
- X86 common SW base (aggressive)
- 20/30% improvement over RISC is claimed

Instruction Stream

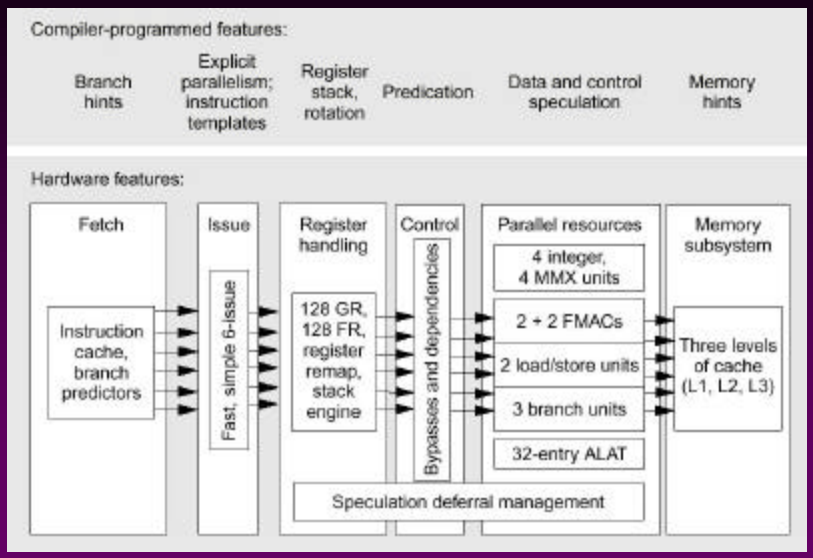
The Processor Front-end

Ganesh Pai

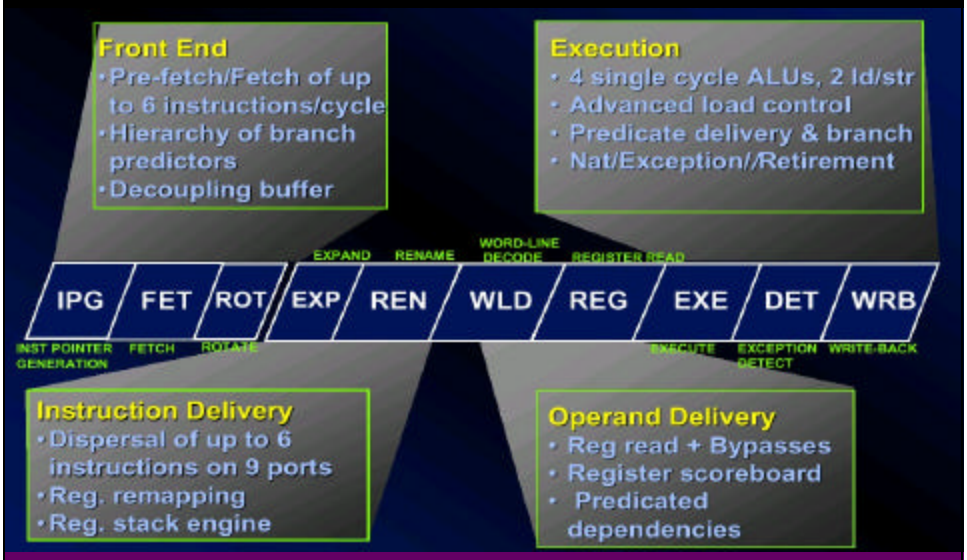
Instruction Stream

- Overview of EPIC hardware
- I-Stream
  - Pipeline
  - I-Cache
  - Prefetch & Fetch
  - Branch prediction
  - Issue (Instruction dispersal & delivery)

Overview of EPIC Hardware



10 Stage In-order Core Pipeline



Pipeline Features

- 6-wide EPIC hardware under precise compiler control
- 10-stage in-order pipeline
- Dynamic support for run-time optimization
  - Ensure high throughput
- Register scoreboard to enforce dependencies

I – Cache ; I – TLB

- 16 Kb
- 4-way set associative
- Fully pipelined
- 64-entry I-TLB
- Single cycle
- Fully associative
- On-chip page walker

- I-Cache filters prefetch requests
- Both enhanced with an additional port
  - To check for a miss

Fetch & Prefetch

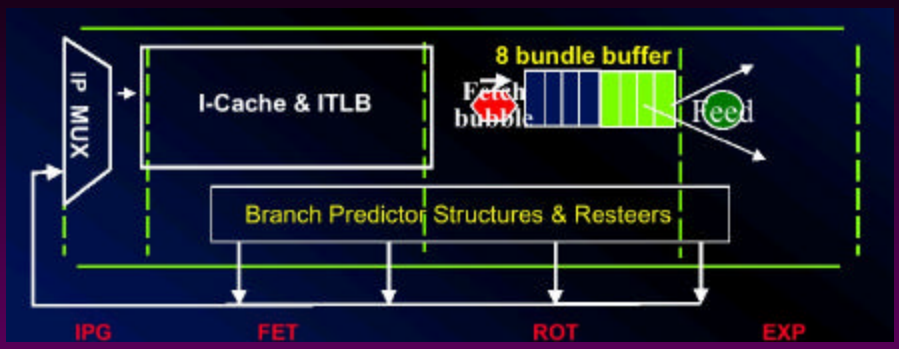
- Speculative fetching

- Both hardware and software prefetching

- Software initiated instruction prefetch
  - Triggered by BPR hints
  - Fetch from L2 into instruction-streaming buffer (ISB)
  - Eight 32-byte entries in the ISB
  - Short 64-byte bursts / long sequential stream

- Eliminate I-fetch bubbles

Fetch & Prefetch

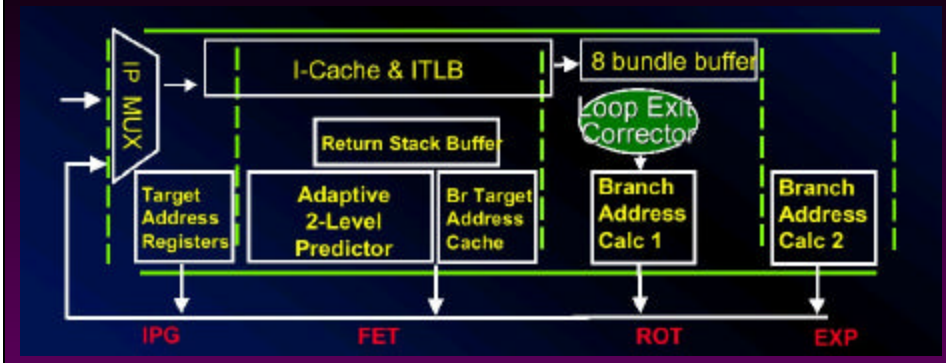


- Decoupling buffer
  - 8 bundles deep
  - Hides stalls, cache misses, branch mispredictions

Branch Prediction

- First emphasis on compiler
  - Reducing branches by predication
- Branch Prediction for remaining cases
  - Assisted by branch hint directives i.e
  - branch target addresses
  - Static hints on branch direction
  - Indications for use of dynamic predictor
- Hierarchy of branch predictors

Branch Prediction



- Branch hints + Predictor Hierarchy
  - Four progressive Resteers
  - Improved branch prediction

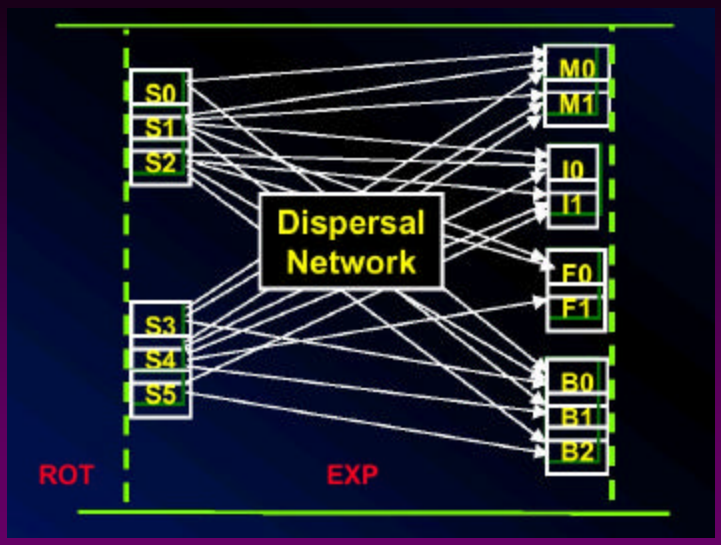
Branch Prediction

- Resteer1 : Single Cycle Predictor
  - 4 TAR s programmed by compiler with “important” hints
  - TAR is a 4 deep FIFO
  - On a “hit” branch is predicted taken
- Resteer2: Adaptive multi-way return predictor
  - 2 level prediction scheme (Yeh and Patt)
  - 512 (128 x 4) entry branch prediction table (BPT)
  - 2 bit saturating up-down counter to predict direction
  - Enhanced by 64-entry multi-way BPT
  - 64-entry branch target address cache (BTAC)
  - 8-entry return stack buffer (RSB)

Branch Prediction

- Resteer3 & 4
  - Two branch address calculators (BAC1 and BAC2)
  - Correction to earlier predictions (if any)
  - A special “perfect-exit-loop-predictor”
- In case of misses in earlier structures
  - Use of a static prediction information from bundles

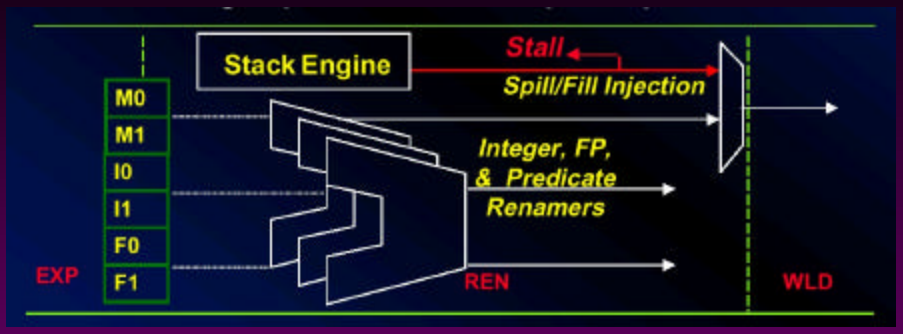
Instruction Dispersal



Instruction Dispersal

- Stop bits eliminate dependency checking
- Templates simplify routing
- Map instructions to first available of 9 issue ports
  - Keep issuing until stop bit
  - Resource over-subscription or asymmetry
- Re-map virtual register to physical register
- Instruction granular

Instruction Delivery



- Register Stacking
- Achieved transparently to the compiler
- Register re-mapping via parallel adders

Data Stream

The Execution Core

Karthik Sankaranarayanan

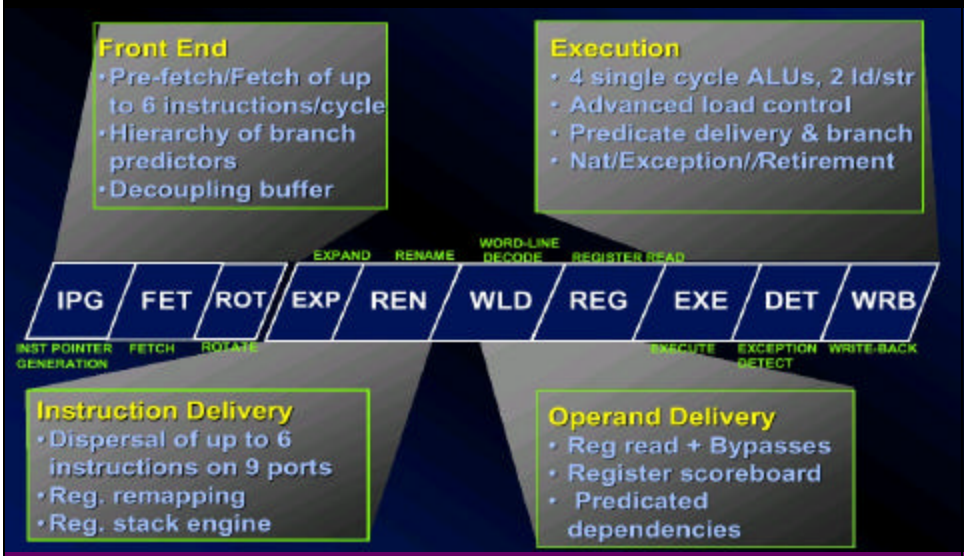
Recap - Execution Units

- 17 units + ALAT
  - 4 ALU
  - 4 MMX
  - 2 + 2 FMAC
  - 2 Load/ Store
  - 3 branch
- Issue Ports
  - 2 I
  - 2 M
  - 2 F
  - 3 B

Register Files

- Integer
  - 128 64-bit
  - 8 read ports (2 x 2 I units, 2 x 2 M units)
  - 6 write ports (1 x 2 I units, 2 x 2 Loads - A.I)
- Floating Point
  - 128 82-bit (double extended)
  - 8 read ports (2 x 2 F units, 2 x 2 M units)
  - 4 write ports (2 x 2 F units, 2 x 2 M units)
- Predicate
  - 64 1-bit , “broadside” R/W
  - 15 read ports (2 x 6 - M, F, I units & 3B units)
  - 11 write ports
    - (2 x 2 M units, 2 x 2 I units, 2 x 1 F unit, 1 x 1 Reg. Rot.)

Recap - 10 Stage Pipeline



Operand Delivery - WLD/REG Stages

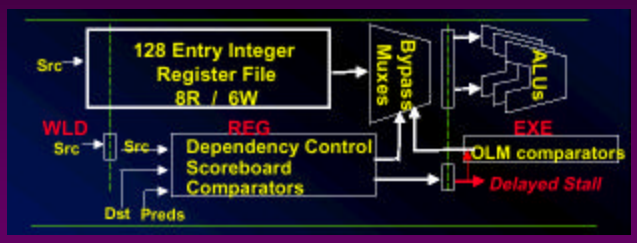
- Register Read
  - WLD (Word Line Decode) - begin access
  - REG - Read Registers
  - WLD - frequency increase?
- Register Scoreboard
  - Hazard detection
  - Stall only dependent instructions
  - Include predicates

```
cmp.eq r1,r2 --> p1,p3
(p1) ld4[r3] --> r4
add r4, r1 --> r5 (no dependence if p1=0)
```

  - Defer stalls

Operand Delivery

- Deferred Stall
  - Stall actually in EXE stage
  - Clock frequency
  - Operand read over - can't re-read
  - Snoop the register bypass network



  - OLM - Operand Latch Manipulation

Execution

- Deferred Stall
- Execute
  - Writes turned off at retirement for false predicates
  - Different latencies - Out Of Order “Execution”
  - In-order retire - scoreboard

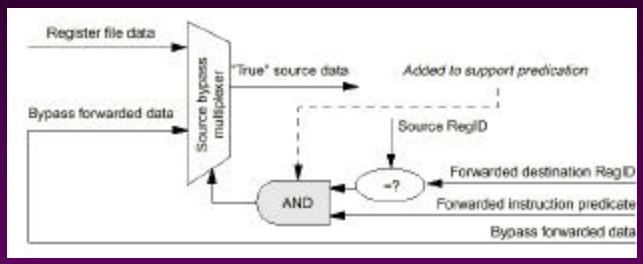
```
cmp.eq r1,r2 --> p1,p3
cmp.eq r7,r8 --> p5,p7

(p1) ld4[r3] --> r4  (reads p1 in EXE)
(p5) add r4, r1 --> r5 (reads p5 in REG)
```

- Predicates
  - ‘Producer’ reads in EXE
  - ‘Consumer’ reads in REG

Execution

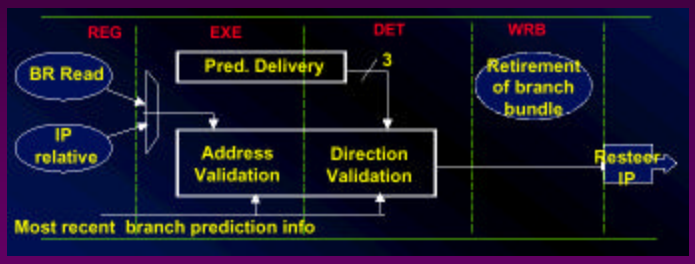
- Predicates
  - Forward as soon as possible
  - Minimize forwarding logic
  - Predicate generation - deterministic latency



  - Separate Register file
    - Speculative, Architectural (SPRF, APRF)
    - Shadow state
  - Bypass paths to eliminate ‘false stalls’

DET/ WRB - Parallel Branches

- Multi-way branches - speculation + predication
- B units - up to 3 branches' parallel execution
- Execution in DET stage
- Can use predicates in the same bundle

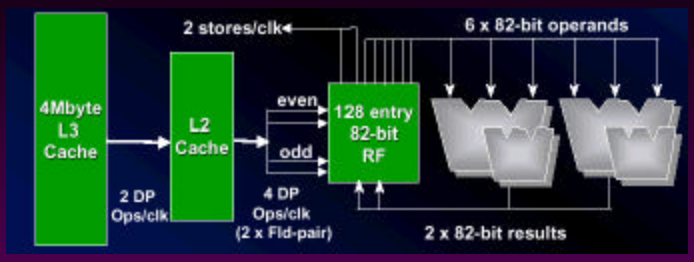


- Software pipeline support - LC, EC

DET/ WRB - Parallel Branches

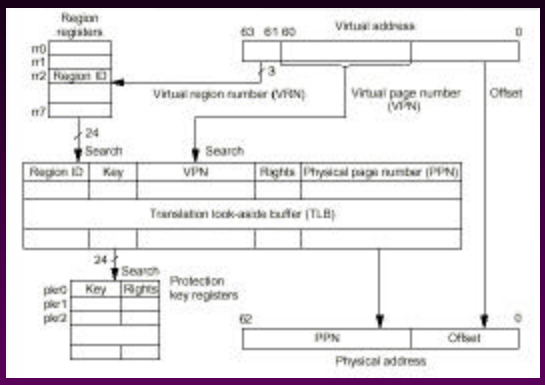
- Control Speculation
  - ld.s, chk.s
  - Exception Deferral - NaTs, NaTVals (poison bits!)
  - Store NaTs? - store.spill, ld.fill (context switch)
  - UNaT, RNaT
- Data Speculation
  - ld.a, chk.a, ld.c
  - ld.c can be issued with dependent instructions
  - ALAT - 32 entries, Register ID, Address, Size
- In-order retirement (branch misprediction/ flush).

FPU Details



- Pipelined FMACs (A*B + C) (5 cycles)
- 4 DP ops/ 8 SP (SIMD) ops per cycle
- Divide/ Square root - S/W pipeline
- FP CMP operations (2 cycles)
- direct L2 cache contact - 2 ldf pair / cycle
- setf, getf, XMA, status registers

Memory Subsystem



- Address translation
  - 32 entry L1 DTLB, 96 entry L2 DTLB, Page size 4K - 256 M
  - Regions for sharing, , Keys for protection
  - Hardware page walker

Memory Subsystem

- L1 Data
  - 16 K, 4-way, 32 byte lines
  - write through, no write allocate
  - dual ported, 2 cycle load latency
- L2, on chip, unified
  - 96 K, 6 way, 64 byte lines, Write back, write allocate
  - Dual ported, 6 cycles Int, 9 cycles FP load latencies
  - MESI protocol for coherence
- L3, off chip, on package, unified
  - 4 M, 4-way, 64 byte lines
  - 21-24 cycle latency, 128 bit bus

Memory Subsystem

- Caches
  - Hints
  - FP NT1 = Int NT2
  - Bias - Easier MESI

Table 2. Implementation of cache hints.

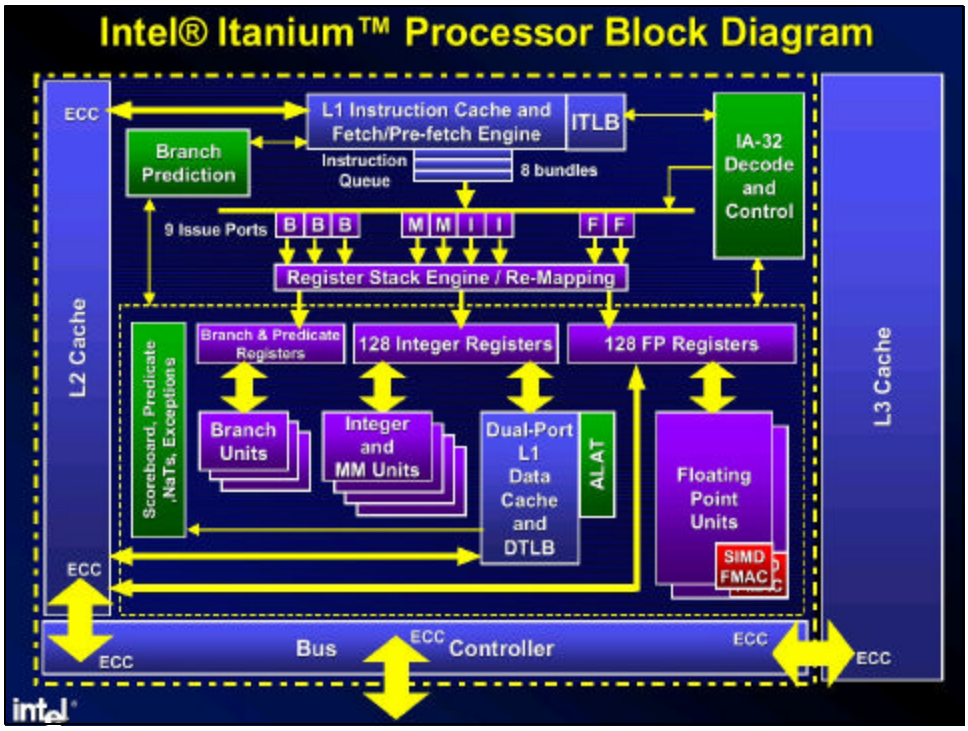
Hint	Semantics	L1 response	L2 response	L3 response
NTA	Nontemporal (all levels)	Don't allocate	Allocate, mark as next replace	Don't allocate
NT2	Nontemporal (2 levels)	Don't allocate	Allocate, mark as next replace	Normal allocation
NT1	Nontemporal (1 level)	Don't allocate	Normal allocation	Normal allocation
T1 (default)	Temporal	Normal allocation	Normal allocation	Normal allocation
Bias	Intent to modify	Normal allocation	Allocate into exclusive state	Allocate into exclusive state

Rest of the Processor

- System Bus
  - 64 bit, 2.1GB/s,
  - Multidrop , Split transaction bus
  - Up to 56 outstanding transactions
  - Optimized MESI protocol
  - Glue-less multiprocessor support (Up to 4)
- IA 32 control
- ECC/Parity coverage of processor and bus
  - Read only structures - parity
  - Data - ECC.

Putting It All Together

The Block Diagram



Conclusions

To Sum Up

Conclusions

- Complexity shift to compilers
- Methods to express compile time information
- Large register files, EPIC specific Hardware
- Optimized FPUs for multimedia applications
- Large L3 cache
- Reliability and performance - server side

"Neat design, Let us see if it succeeds"