Itanium

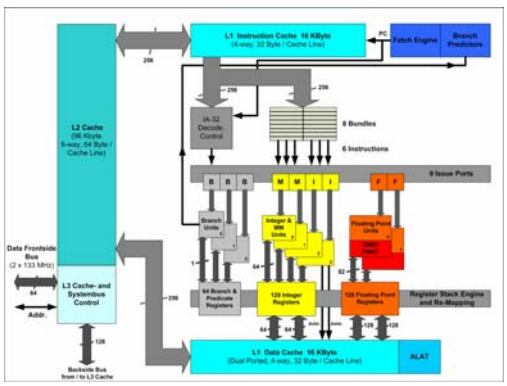
cs9244

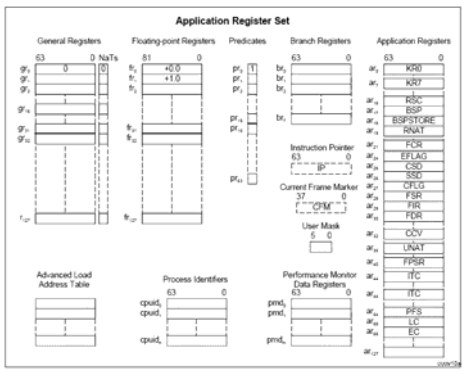
Outline

- History of Itanium
- Itanium Architecture Features
  - Registers
  - Instructions
  - Speculation
  - Predication and Hints
- Itanium 2 vs. Itanium
  - Execution Units
  - Cache system
  - Pipeline

History

- 1994: Intel and HP began working on Merced
- 2001: released Itanium processor at 733, 800MHz
- 2002: released Itanium 2 processor at 900MHz and 1GHz, codenamed McKinley
- 2003:
  - Madison was introduced with three version
  - Hondo was announced as the HP mx2 dual-processor module
  - Deerfield was released as the first low voltage Itanium processor
- 2004:
  - released first processor in Madison 9M series
  - Fanwood core debuted
- Upcoming: Montecito



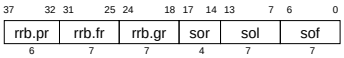


Application Register State

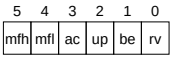
- 128 General Registers
  - GR0 = 0 (read-only)
  - GR0-GR31 are *static,* GR32-GR127 are *stacked*
- 128 Floating-point Registers
  - FR0 = 0.0 (read-only), FR1 = 1.0 (read-only)
  - FR0-FR21 are *static,* FR32-FR127 are *rotating*
- 64 Predicate Registers
  - PR0 = 1 (read-only)
  - PR0-PR15 are *static,* PR16-PR63 are *rotating*
- 8 Branch Registers
  - Holds target address for indirect branches
- 128 Application Registers

Application Register State (cont)

- Current Frame Marker

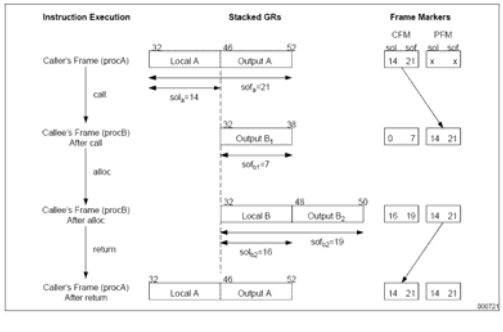


- Instruction Pointer
- User Mask



rv: Reserved
be: IA-64 big-endian memory access enable
up: User performance monitor enable
ac: Alignment check for data memory references
mfl: Lower floating-point registers written
mfh: Upper floating-point registers written

Register Stack



Register Stack (cont)

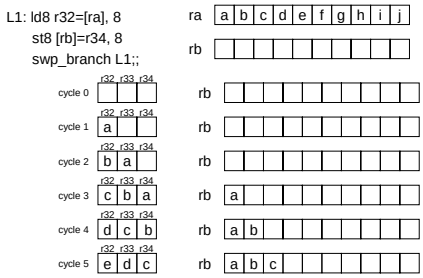
- Register stack frame can be resized using *alloc* instruction
- Register Stack Engine (RSE)
  - Automatically save and restore register stack without explicit software intervention
  - Use spare memory bandwidth in the background
  - Spill and fill may cause RSE traffic
  - RSE traffic degrades performance

Register Rotation

- Register renaming mechanism that enables the concurrent execution of multiple iterations of a loop
- Rotating register
  - PR32-PR127, FR32-FR127, programmable sized GR starting from GR32
  - Size of rotating area in GR file determined by *alloc* instruction (size either be 0 or 8x)
  - Rotate toward larger register number
  - Renaming register number = rotate register number + value of rrb

Register Rotation (cont)

```
L1: ld8 r32=[ra], 8
    st8 [rb]=r34, 8
    swp_branch L1;;
```



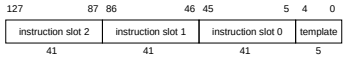
Instruction Types

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

Example:
A : add, or
I : mov, shl
M : load
F : fadd
B : br, brp
L+X: brl

Bundle and Template

- Three instructions are grouped together into a 128-bit container called *bundle*



- Template
  - 12 basic types: MII, MI|I, MLX, MMI, M|MI, MFI, MMF, MIB, MBB, BBB, MMB, MFB
  - Mapping of instruction to execution unit
  - *Stop*

Control Speculation

- Overlap long load latency
- NaT/NaTVal support

……	……	recovery:
if (a > b) {	ld.s target, addr	ld target, addr
ld target, addr	if (a > b) {	br next
……	chk.s target, recovery	
}	next :	
	……	
	}	

ld.s – control speculative load
chk.s – check instruction

Data Speculation

//other instruction
 	
st8 [r4] = r12
ld8 r6 = r[8];;
add r5 = r6, r7;;
st8 [r18] = r5
	ld8.a r6 = r[8];;
	//other instruction
	add r5 = r6, r7;;
	//other instruction
	st8 [r4] = r12
	chk.a.clr r6, recovery back:
	st8 [r18] = r5
	
	// somewhere else in program
	Recovery:
	ld8.a r6 = r[8];;
	add r5 = r6, r7
	br back

- Advanced load (ld.a, ldf.a and ldfp.a)
  - compute ALAT register tag
  - if ALAT entry exists, removes it
  - allocate a new ALAT entry
  - load the value into target register
- Advanced load check
  - looks for a matching ALAT entry, if found, falls through to next instruction otherwise branches to the recovery
- Two kinds of compiler-generated recovery
  - check load instructions: ld.c, ldf.c or ldfp.c
  - Advanced load check: chk.a

- ALAT (advanced load address table)
  - Invalidating ALAT entries: by advanced load, certain instructions ,or events that alter memory state

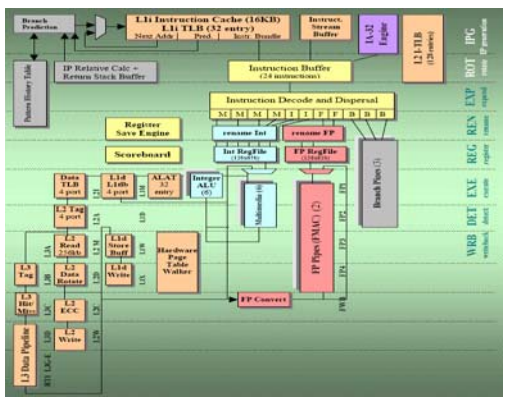
Predication

- Conditional execution of an instruction base on a qualifying predicate
  - Convert branch conditions to predicate registers
  - Convert control dependences to data dependences
- Most instructions can be predicated

Branch Hints

- Mechanism to decrease the branch misprediction rate
- Do not affect the functional behavior of the program and may be ignored by the processor

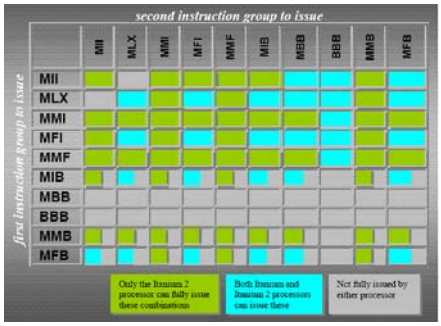
spnt	**Static Not-Taken**	**Ignore this branch, do not allocate prediction resources for this branch.**
sptk	**Static Taken**	**Always predict taken, do not allocate prediction resources for this branch.**
dpnt	**Dynamic Not-Taken**	**Use dynamic prediction hardware. If no dynamic history information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



Execution Resources

Itanium 2 Proc. Execution Units	# Units	Latency
Memory Load Ports	2	1 cycle (L1)
Memory Store Ports	2	NA
ALUs (integer)	6	1 cycle
Integer Units	2	1 cycle
Integer Shift	1	1 cycle
Multimedia ALUs	6	2 cycles
Parallel Multiply Units	1	2 cycles
Parallel Shift-Mask Units	2	2 cycles
FP FMAC (multiply-accumulate)	2	4 cycles
FP FMISC (compares, merge, etc)	2	4 cycles
Branch Unit	3	0-2 cycles

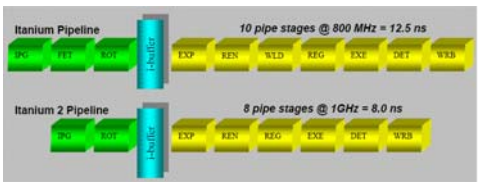
Issue Combinations for 2 Bundles



Cache System Distinction

- Cache Latency
  - L1i/L1d 2 cycles => 1 cycle
  - L2 (I, FP) 6, 9 cycles => 5, 6 cycles
  - L3 (I, FP) 21, 24 cycles => 12, 13 cycles
- Virtual address and physical address
  - 50-bit => 64-bit for virtual address
  - 44-bit => 50-bit for physical address
- Cache line size
  - Doubled for every level of cache
- Page size
  - Up to 4GB, used to be up to 256MB
- Cache line transfer bandwidth
  - Doubled

Pipeline



- In-order pipeline
- Pipeline deduction leads to 4-6% performance improvement

IPG=instruction pointer, **FET**=Fetch, **ROT**=Rotate, **EXP**=Expand, **REN**=Rename, **WLD**=Word-line decode, **REG**=Register read, **EXE**=Execute, **DET**=Exception detect, **WRB**=Write-back