Switch Connection

Switch design for fast processor-to-memory transfer without contention.
Crossbar Switch Operation

A crossbar switch is *non-blocking*, being able to concurrently switch any input line to any (available) output line.

The crossbar requires \( n \times n = n^2 \) *individual switches* for a square switch.
**Straight Through X-Bar**

If you would rather view the X-bar as a symmetric in-out switch, we can draw it this way:

And hide the insides and put on the switch symbol:
Multiple Parallel Switches

We actually need to switch *multiple parallel lines*. For example, if we have a 32-bit address bus, a 64-bit data bus, and 20 control lines, we need 32 + 64 + 20 = 116 *parallel cross-point switches*. Following shows our 6 x 6 cross-bar with 4 parallel switches:

An 8 x 8 crossbar with 116 parallel lines would require:
- $8 \times 8 \times 116 = 7424$ individual switches.
Omega (Shuffle-exchange) Network Operation

The full crossbar switch is non-blocking and fast – only one switch from input to output. However, it takes a lot of hardware to implement it. Can we relax our requirements somewhat and produce a cheaper switch?

We look at the perfect shuffle:

![Diagram of perfect shuffle]

Drawn another way – and its inverse:

![Diagram of perfect shuffle inverse]
2-way crossbar switch boxes

Each 2-way crossbar requires 4 individual switches.

The shuffle with exchange boxes
**IA-64 Architecture**

Over the past several years, strategies to increase processor performance have focused on finding more instruction-level parallelism (ILP). ILP is basically the idea of finding several instructions to execute at the same time.

However, difficult problems limit ILP:

- Branch instructions, which introduce control dependencies,
- Data dependencies among neighboring instructions.
- Memory latency, the time it takes to retrieve data from memory.

In the IA-64 architecture, the compiler plays a major role in using predication, instruction grouping, many registers, and control speculation to expose more ILP.

The machine instruction-set architecture exposes machine characteristics to the source program by providing:

- “Predicate” instructions to reduce the number of branches.
- Branch “hints” to reduce the number of mispredicts.
- Multiple instructions in a “group” that can be executed in parallel.
- Many registers (128 GPRs, 128 FPRs & 64 predicate registers).
- A speculative load instruction with deferred exception processing to pre-fetch a cache line.
- Cache control hints.

Intel and HP are naming this concept EPIC (*Explicitly Parallel Instruction Computing*).
Instruction Bundles and Groups

Instruction bundles are sets of three instructions with a template.

<table>
<thead>
<tr>
<th>Instruction 2</th>
<th>Instruction 1</th>
<th>Instruction 0</th>
<th>Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>41 bits</td>
<td>41 bits</td>
<td>41 bits</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

128-bit Instruction Bundle

The Template

The template field describes the type of instructions (memory, integer, floating point, branch) in the three instruction slots, including an extended instruction type that occupies two slots (long immediate operands).

The template also specifies stops that bound a group of instructions that have no RAW, WAW or WAR dependencies among them, and can be explicitly executed in parallel. The assumption is that the compiler generates code such that any sequential dependencies are located in separate groups.

Groups can span bundles, and can begin and end within a bundle.
**Branch Instructions**

Any (or all) of the instructions in a bundle may be a branch instruction, and execution ordering is from instruction 0 to 2 in the group. However, the target of a branch instruction is always the first instruction of a bundle.

Branches can be relative to the instruction counter or absolute. Absolute branches always use an address contained in a branch register as the target address.

Branch *prediction hints* can also be specified in the branch instruction. These hints specify that the processor should assume the branch is:

- always not taken,
- always taken,
- not taken if no history,
- taken if no history.

**Predicate Registers versus Condition Codes**

The IA-64 uses 64 binary predicate registers instead of condition codes for specifying conditional execution.

Compare instructions set two predicate registers – one for true and one for false. Qualifiers define the type of compare. For example: CMP.EQ, CMP.GT, CMP.NE, etc.

*All instructions are conditionally executed* depending on the setting of a predicate register. If the specified predicate register is true, the instruction is executed. If it is false, the instruction is not executed. Predicate register 0 is defined to be always true.
Use of predicates

Predicates are used to reduce the number of branch instructions needed.

Normal machine code expansion for an If, Then, Else clause:

If (A == B) then C = D else E = F;

       ------- ; Instruction
       Cmp  A,B ; Compare A to B
       BE    TrueBlk ; Branch = to the true block
       E  F ; Execute only if False
       BR    EndBlk ; Around then block
TrueBlk C  D ; Execute only if True
EndBlk       ------- ; Instruction

Expansion using predicate instructions:

If (A == B) then C = D else E = F;

       ------- ; Instruction
p1,p2  Cmp.Eq A,B ; Compare Equal & set pred's
  (p2)   E  F ; Execute only if p2 (False)
  (p1)   C  D ; Execute only if p1 (True)
       ------- ; Instruction
Register Sets

General Registers
♦ 128 registers
♦ 64 bits

Floating Point Registers
♦ 128 registers
♦ 82 bits

Predicate Registers
♦ 64 registers
♦ 1 bit

Branch Registers
♦ 8 registers
♦ 64 bits
Cache Control Hints

Load and store instructions (and speculative load instructions) provide for giving spatial and temporal cache hints to the processor.

♦ Spatial hint – will (not) likely need the next memory block.
♦ Temporal hint – will (not) likely need this data item in the near future.

These hints specify the condition through two levels of cache. That is, you can specify the hint for level 1 cache, level 2 cache, or all levels of cache.

When a branch register is loaded with an address, the load instruction can hint that it is likely that the address will be used shortly, so bring the target instruction up into the cache.

There is also an “Invalidate cache line” instruction and a hint to obtain exclusive control of a cache line.

♦ The compiler can statically specify these hints based on program flow expected at compile time.
♦ The IA-64 architects also envision program trace data being used to “fix up” the hint information in object code for frequently-executed, compute-intensive applications.

Names – the difference between EPIC, IA-64 and Merced

♦ EPIC is a technology, like RISC or CISC
♦ IA-64 is an architecture, like PowerPC or x86
♦ Merced is a processor, like Pentium III or K6-III