



EE4801 – Lecture 21

The IA-64 Architecture

The IA-64 Architecture

- ◆ Developed as a collaboration between Hewlett-Packard and Intel.
- ◆ Attempts to push (past) the limits of superscalar architectures.
- ◆ Marks the end of architectures like the P4.
- ◆ Tries to exploit instruction-level parallelism by using long instruction words.
- ◆ Currently marketed as the Intel Itanium.

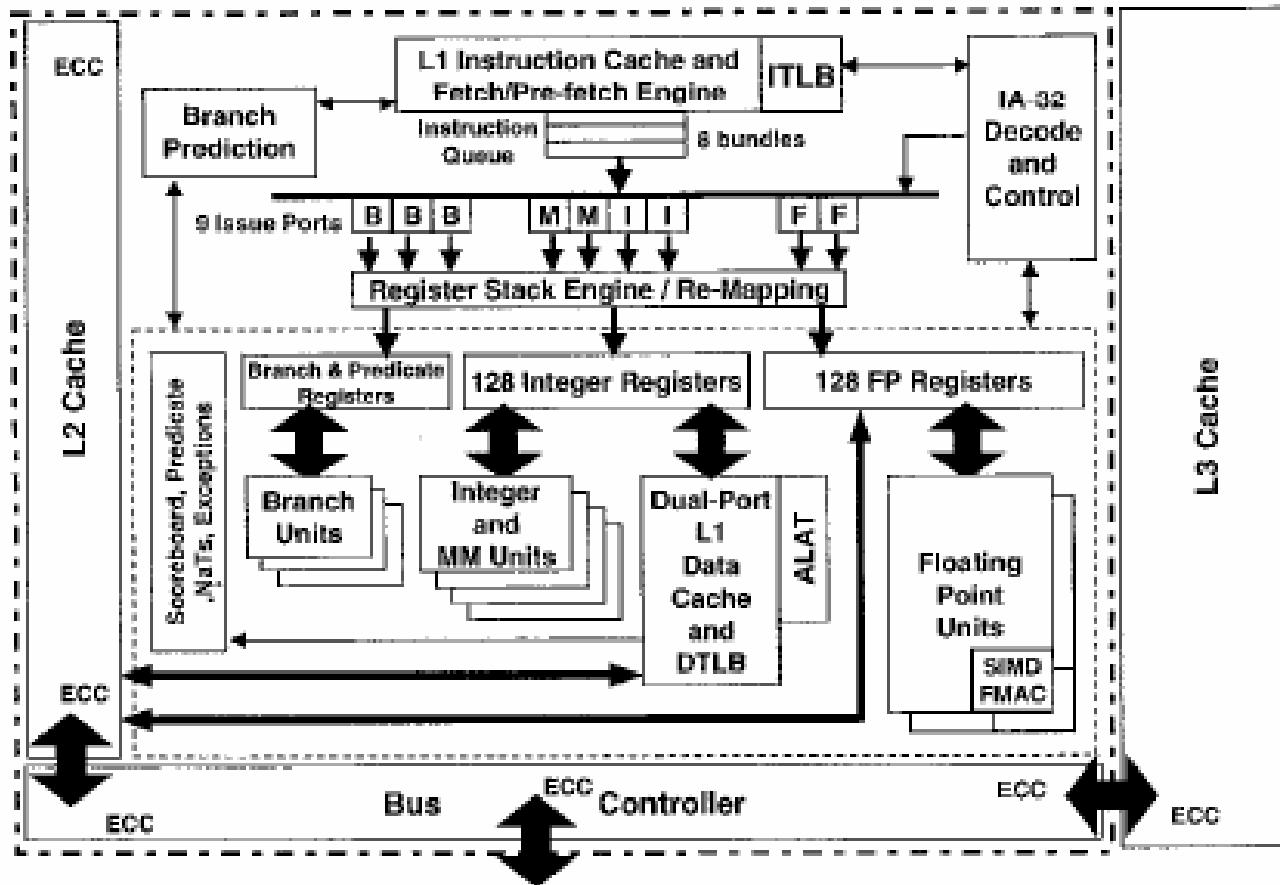
Two Main Goals

- ◆ The IA-64 Architecture attempts to improve performance by addressing the two main issues that slow a processor down:
 - Branches – IA-64 attempts to eliminate a large fraction of program branches.
 - Memory Latency – IA-64 attempts to minimize the effect of memory access.

IA-64 Hardware Architecture

- ◆ The IA-64 contains more of everything:
 - 128 General purpose registers
 - 128 Floating-point registers
 - 4 Integer and 4 MMX processing units
 - 2 single precision and 2 double precision floating-point units.
 - 2 Load/Store units
 - 3 Branch Units
 - L1, L2 and L3 caches

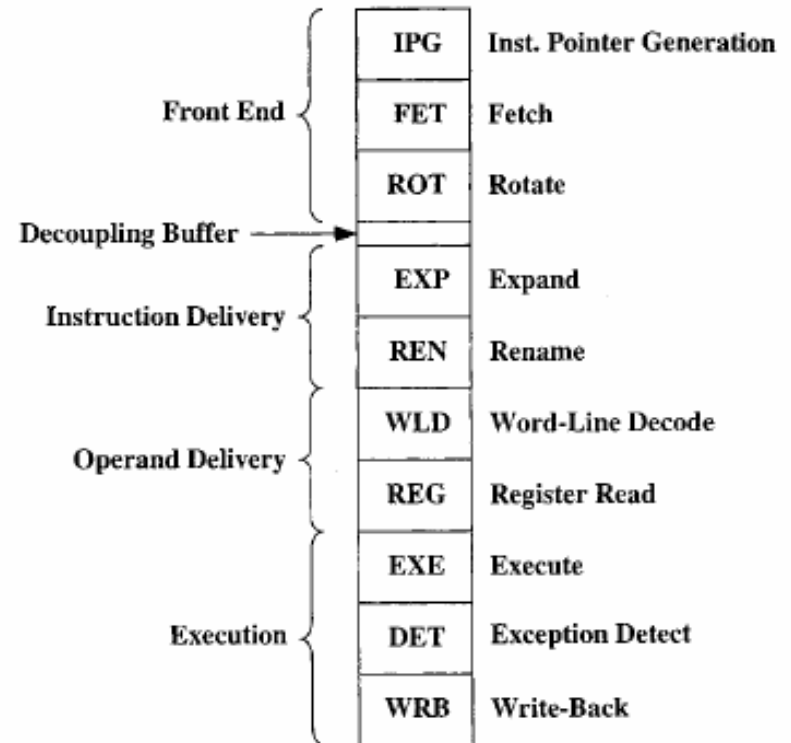
IA-64 Hardware



Source: Singer, "The First IA-64 Microprocessor: A Design for Highly-Parallel Execution"

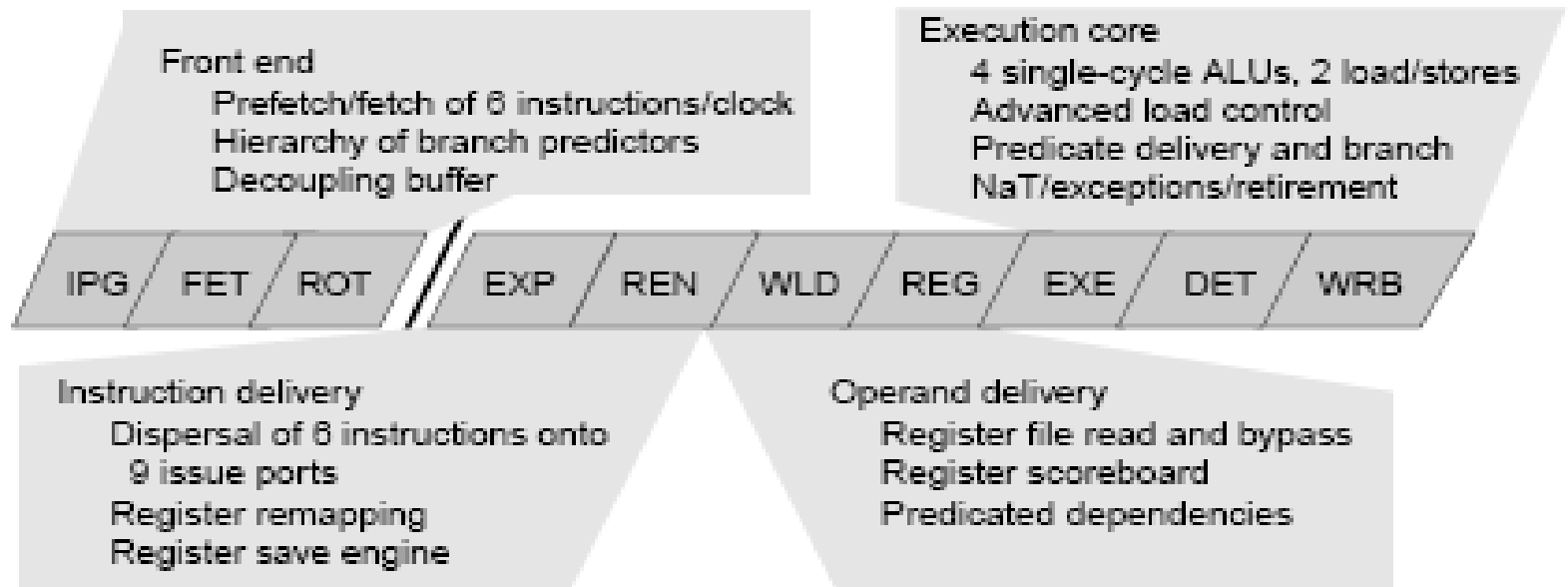
IA-64 Instruction Pipeline

- ◆ 10 Stage Pipeline
- ◆ As many as six IA-64 instructions can execute per clock cycle.
- ◆ The compiler works closely with the hardware to extract parallelism prior to execution.



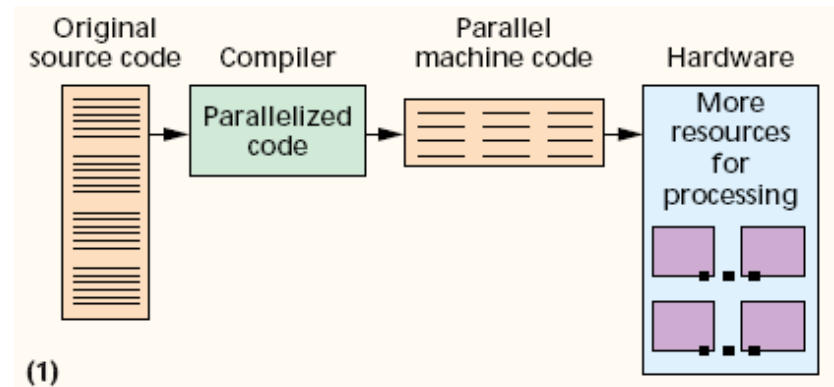
Pipeline Description

- ◆ While the high-level functions of the pipe are as expected, each of these functions is also pipelined:



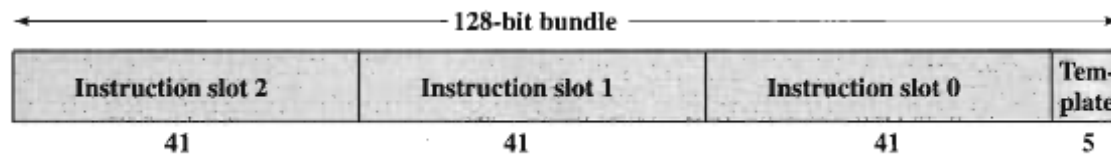
EPIC – Explicitly Parallel Instruction Computing

- ◆ The compiler for the IA-64 analyzes the original source and generates explicitly parallel instructions.
- ◆ “Predication” and “Control Speculation” are used to help the compiler identify instruction level parallelism.

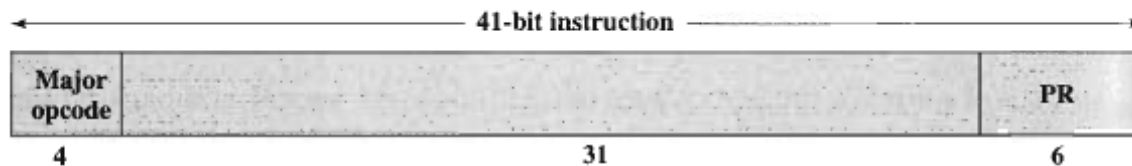


IA-64 – VLIW Instruction Format

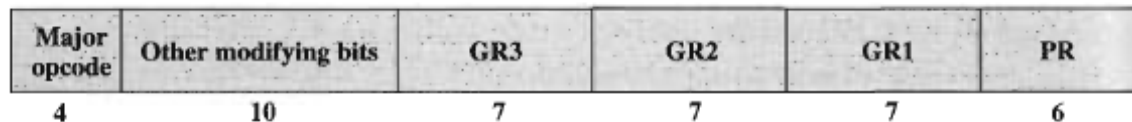
- ◆ Each 128 bit instruction contains 3 instructions:



(a) IA-64 bundle



(b) General IA-64 instruction format



(c) Typical IA-64 instruction format

PR = Predicate register
GR = General or floating-point register

Instruction Types

- ◆ Each instruction, or syllable, in a 128-bit bundle is associated with an execution unit type.
- ◆ There are four types of execution units:
 - I-Unit – Integer arithmetic.
 - M-Unit – Loads/Stores and some Integer ops.
 - B-Unit – Branch handling
 - F-Unit – Floating point.

Instruction Associations

- ◆ Each instruction is associated with particular execution units.
- ◆ This allows the compiler to select instruction sequences to optimize use of processor resources.

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
X	Extended	I-unit/B-unit

The Template Field

- ◆ Each fetch obtains a bundle of three instructions and a template.
- ◆ The template field indicates:
 - Resource allocations
 - Dependency relationships
- ◆ Templates work across bundles – it is possible (but unlikely) to have 20 operations being performed concurrently.

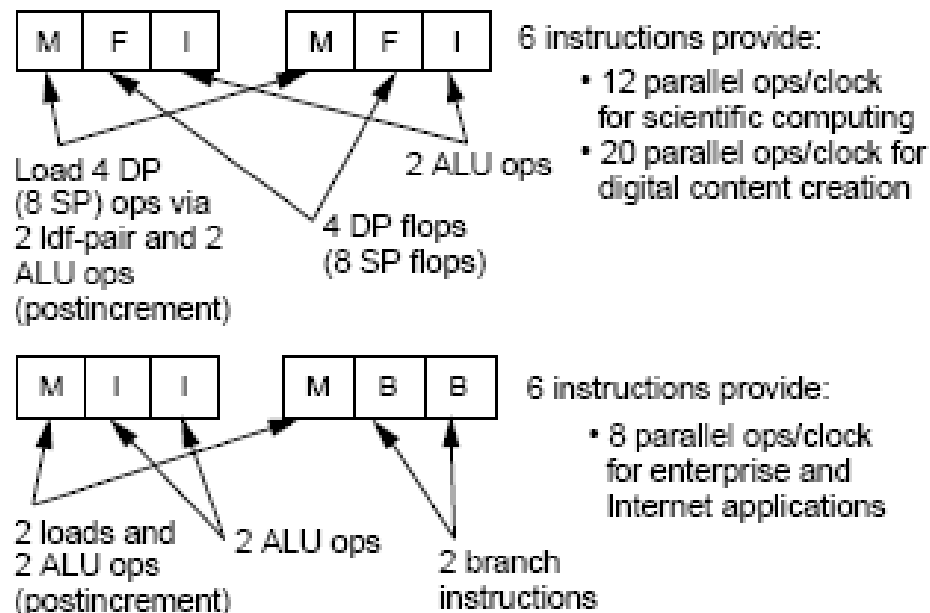
Use of Templates

- ◆ Templates are used to aid in scheduling the processing of instructions.
- ◆ This allows the compiler to communicate information to the processor about instruction execution.

Template	Slot 0	Slot 1	Slot 2
00	M-unit	I-unit	I-unit
01	M-unit	I-unit	I-unit
02	M-unit	I-unit	I-unit
03	M-unit	I-unit	I-unit
04	M-unit	L-unit	X-unit
05	M-unit	L-unit	X-unit
08	M-unit	M-unit	I-unit
09	M-unit	M-unit	I-unit
0A	M-unit	M-unit	I-unit
0B	M-unit	M-unit	I-unit
0C	M-unit	F-unit	I-unit
0D	M-unit	F-unit	I-unit
0E	M-unit	M-unit	F-unit
0F	M-unit	M-unit	F-unit
10	M-unit	I-unit	B-unit
11	M-unit	I-unit	B-unit
12	M-unit	B-unit	B-unit
13	M-unit	B-unit	B-unit
16	B-unit	B-unit	B-unit
17	B-unit	B-unit	B-unit
18	M-unit	M-unit	B-unit
19	M-unit	M-unit	B-unit
1C	M-unit	F-unit	B-unit
1D	M-unit	F-unit	B-unit

Multiple Issue of Instructions

- ◆ Each instruction can process one 64-bit or two 32-bit quantities.
- ◆ 6 instructions can be issued concurrently.



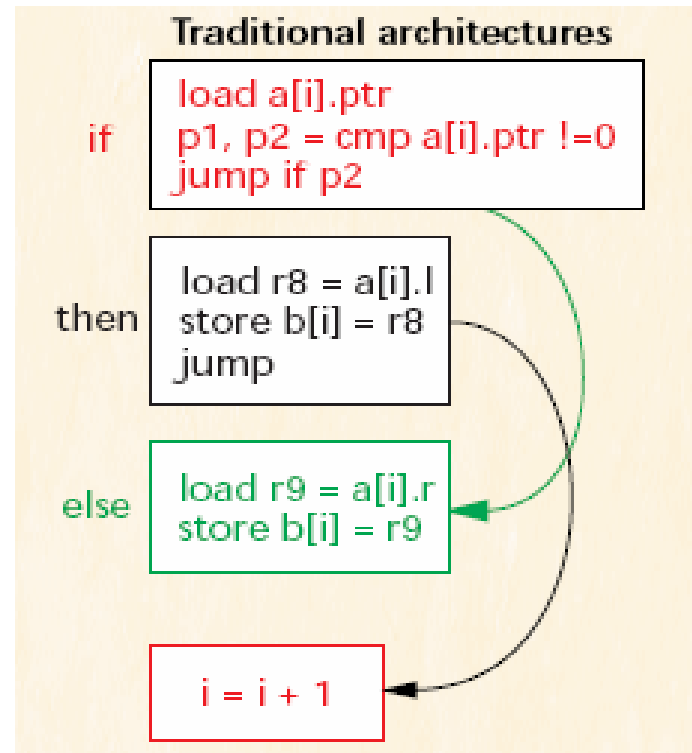
Predication

- ◆ One techniques used in the IA-64 architecture is called *predication*
- ◆ The compiler adds a predicate to each instruction
- ◆ These predicates are tags which depend on the result of a conditional statement.
- ◆ The results of an instruction are only written to memory if the predicate for that instruction is true.

If-then-else statement

- ◆ Consider a traditional system:

```
if a[i].ptr != 0
    b[i] = a[i].l;
else
    b[i] = a[i].r;
i = i + 1
```



Predicated Execution

- ◆ In the IA-64, both branches can be issued in parallel.
- ◆ The value of the predicate will ultimately determine which path will be committed.
- ◆ The other path is discarded.

```
load a[i].ptr  
p1, p2 = cmp a[i].ptr !=0  
  
<p1>load a[i].l    <p2> load a[i].r  
<p1> store b[i]    <p2> store b[i]  
  
i = i + 1
```

Control Speculation

- ◆ The IA-64 also allows speculative execution of load instructions.
- ◆ This reduces memory latency by increasing the probability that required data has been loaded by the time it is needed.
- ◆ The IA-64 allows data to be loaded even before the branch which would cause its load is issued (hoisting).

Example of Hoisting

- ◆ By speculatively loading $a[t1-t2]$ it is possible to avoid an extra load delay.

