COMP4611: Design and Analysis of Computer Architectures

Pipelining

EPIC and IA-64

Lin Gu

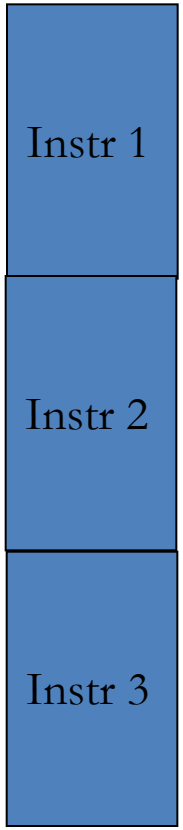
CSE, HKUST

Modern RISC processors

- Complexity has nonetheless increased significantly
- Superscalar execution (where CPU has multiple functional units of the same type e.g. two add units) require complex circuitry to control scheduling of operations
- What if we could remove the scheduling complexity by using a smart compiler…?

VLIW & EPIC

- VLIW – very long instruction word
- Idea: pack a number of *non-interdependent* operations into one long instruction
- Strong emphasis on ***compilers*** to schedule instructions
- Natural successor to RISC – designed to avoid the need for complex scheduling in RISC designs
- Example: IA-64



3 instructions scheduled
into one long instruction word

Today's Architectural Challenges

Performance barriers :

- Memory latency
- Branches
- Loop pipelining and call / return overhead
- Hardware-based instruction scheduling
- Unable to efficiently schedule parallel execution
- Too few registers
- Unable to fully utilize multiple execution units

Improving Performance

To achieve improved performance, Itanium(R) architecture code accomplishes the following:

- Increases instruction level parallelism (ILP)
- Improves branch handling
- Hides memory latencies

Instruction level parallelism (ILP)

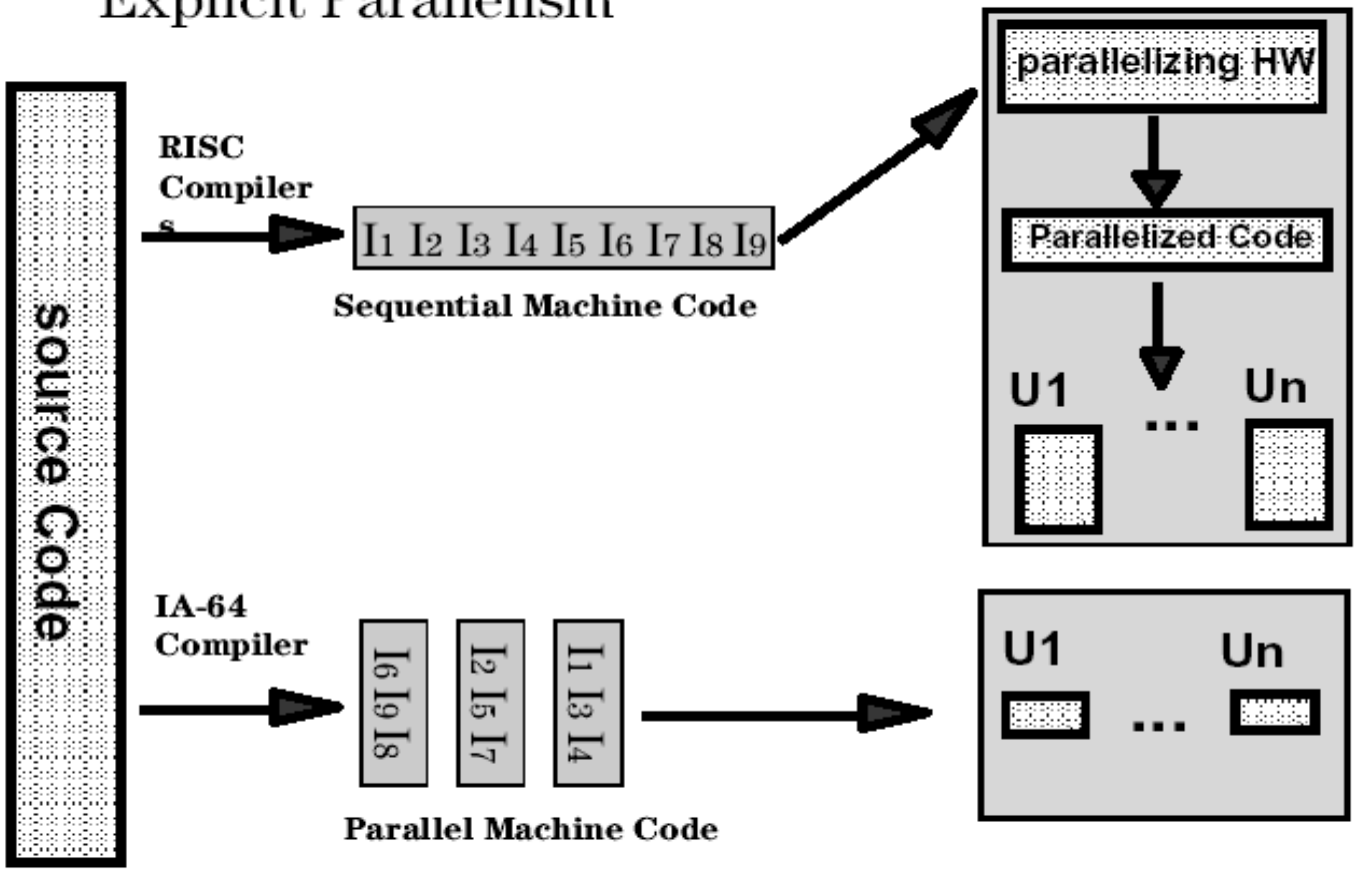
Increase ILP by:

- More resources
  - Large register files
  - Avoiding register contention
- 3-instruction wide word
  - Bundle
  - Facilitates parallel processing of instructions
- Enabling the compiler/assembly writer to explicitly indicate parallelism

IA- 64: The Itanium Processor

- Intel and Hewlett-Packard Co. designed a new architecture, IA-64, that they expected to be much more effective at executing instructions in parallel
- IA-64 is brand new ISA, derived from EPIC (Explicitly Parallel Instruction Computing)
- A radical departure from the traditional paradigms.

Explicit Parallelism



IA - 64

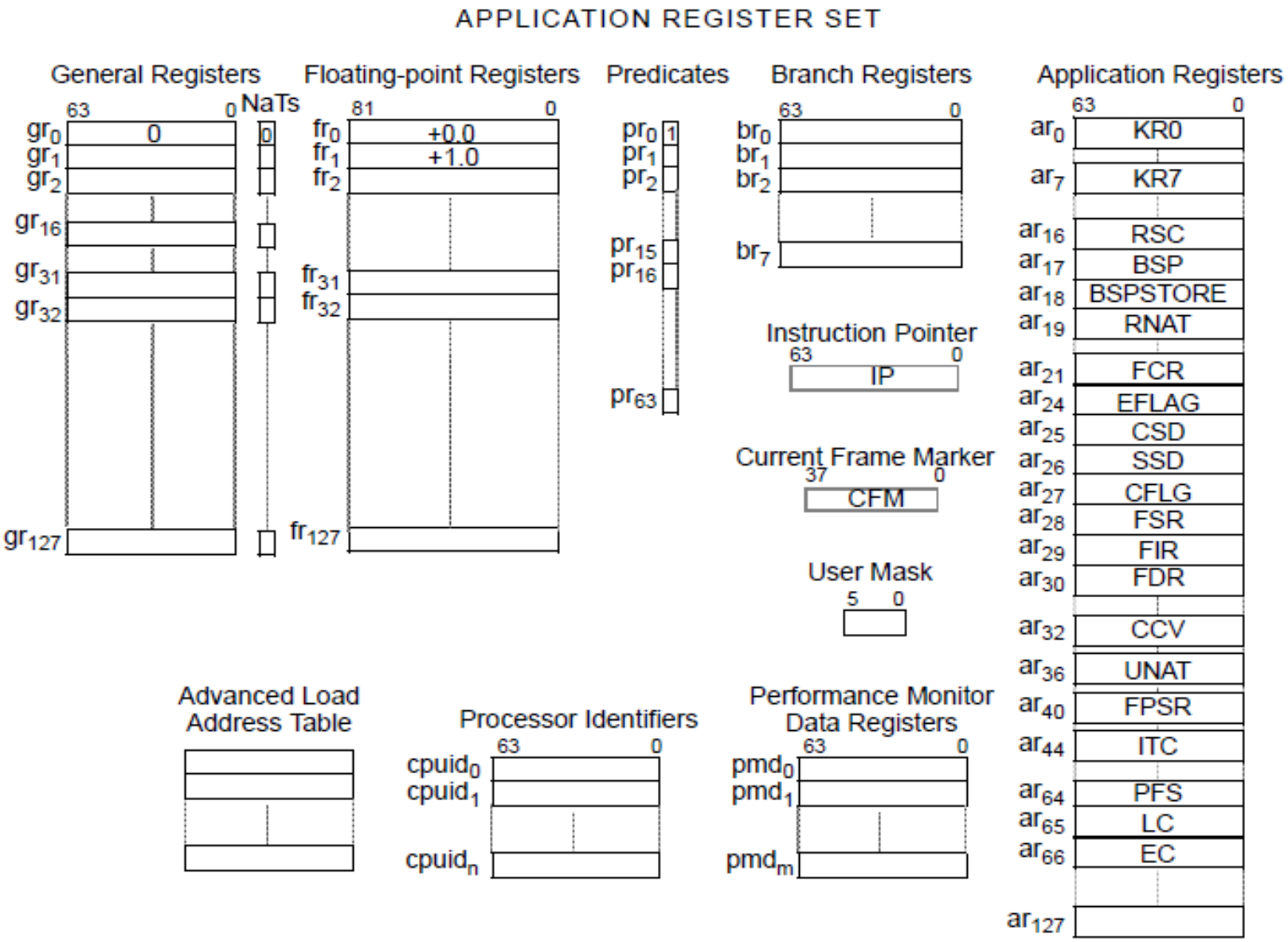
- 64-bit ISA
- Instructions are scheduled by the compiler, not by the hardware
- Much of the logic that groups, schedules, and tracks instructions is not needed thus simplifying the circuitry and promising to improve performance

Intel IA-64

- Massive resources
  - 128 GPRs (64-bit not including the NaT/Not a Thing bit)
  - 128 FPRs (82-bit)
  - 64 predicate registers
  - Also has *branch registers* for indirect branches

- Contrast to:
  - RISC: 32 int, 32 FP, handful of control regs
  - x86: 8 int, 8 fp, handful of control regs
    - x86-64 bumps this to 16, SSE adds 8/16 MM regs

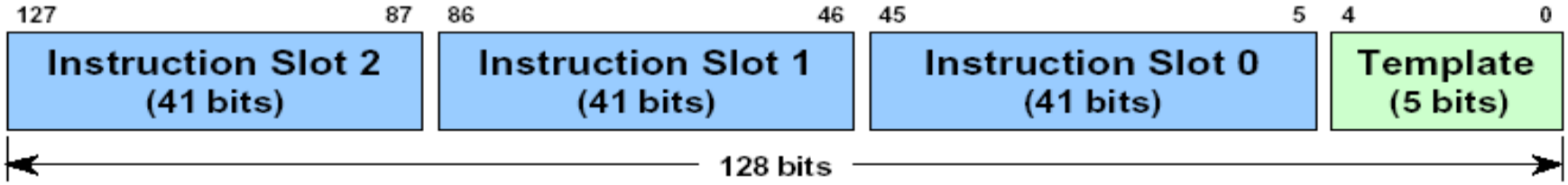
IA-64 Registers



IA-64 Groups

- Compiler assembles *groups* of instructions
  - No register data dependencies between instructions in the same group
    - Memory dependence may exist
  - Compiler explicitly inserts “stops” to mark the end of a group
  - Group can be arbitrarily long

IA-64 Bundles



- Bundle == The “VLIW” Instruction
  - Templates define the types of instructions in a bundle
    - 5-bit template encoding
    - also encodes “stops”
  - Three 41-bit instructions
  - Bundles can be connected together and executed simultaneously

- 128 bits per bundle
  - average of 5.33 bytes per instruction
    - x86 only needs 3 bytes on average

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[a]

a. L+X Major Opcodes 0 - 7 execute on an I-unit. L+X Major Opcodes 8 - F execute on a B-unit.

- Instructions are divided into different types
  - the type determines which functional units the instruction operates on
  - templates are based on these types

Bundle Templates

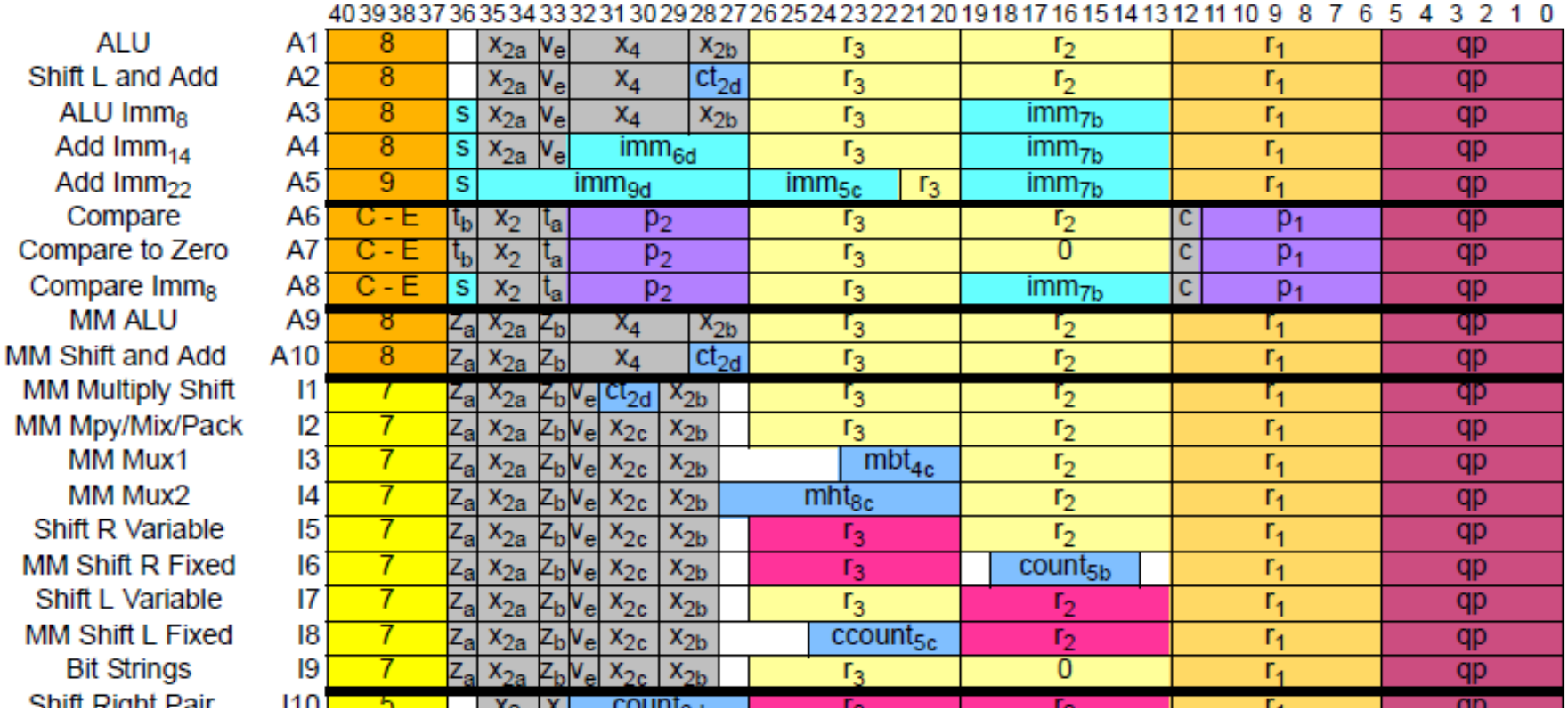
Template	Slot 0	Slot 1	Slot 2
00	M-unit	I-unit	I-unit
01	M-unit	I-unit	I-unit
02	M-unit	I-unit	I-unit
03	M-unit	I-unit	I-unit
04	M-unit	L-unit	X-unit[a]
05	M-unit	L-unit	X-unit[a]
06			
07			
08	M-unit	M-unit	I-unit
09	M-unit	M-unit	I-unit
0A	M-unit	M-unit	I-unit
0B	M-unit	M-unit	I-unit
0C	M-unit	F-unit	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
14			
15			
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
1A			
1B			
1C	M-unit	F-unit	B-unit
1D	M-unit	F-unit	B-unit
1E			
1F			

- Not all combinations of A, I, M, F, B, L and X are permitted
- Group “stops” are explicitly encoded as part of the template
  - can’t stop just anywhere

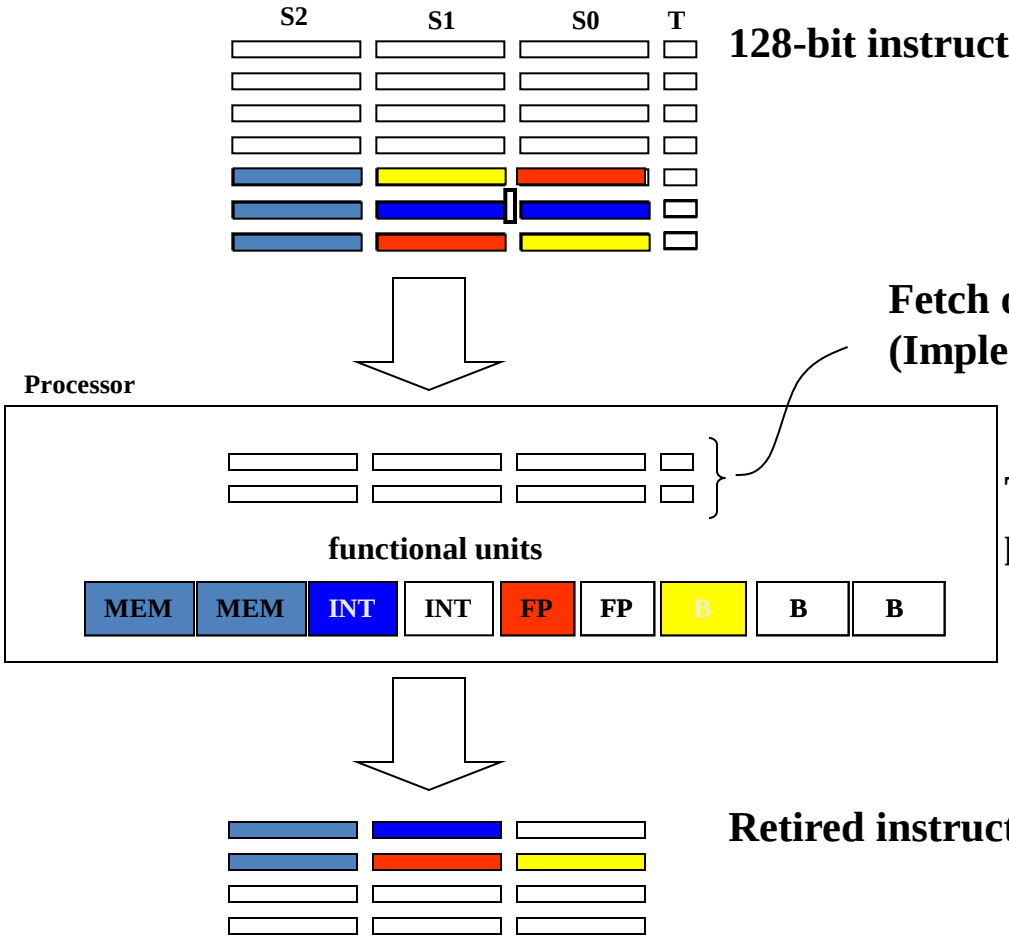
Some bundles identical except for group stop

Individual Instruction Formats

- Fairly RISC-like like
  - easy to decode, fields tend to stay put



Explicitly Parallel Instruction Computing (EPIC)



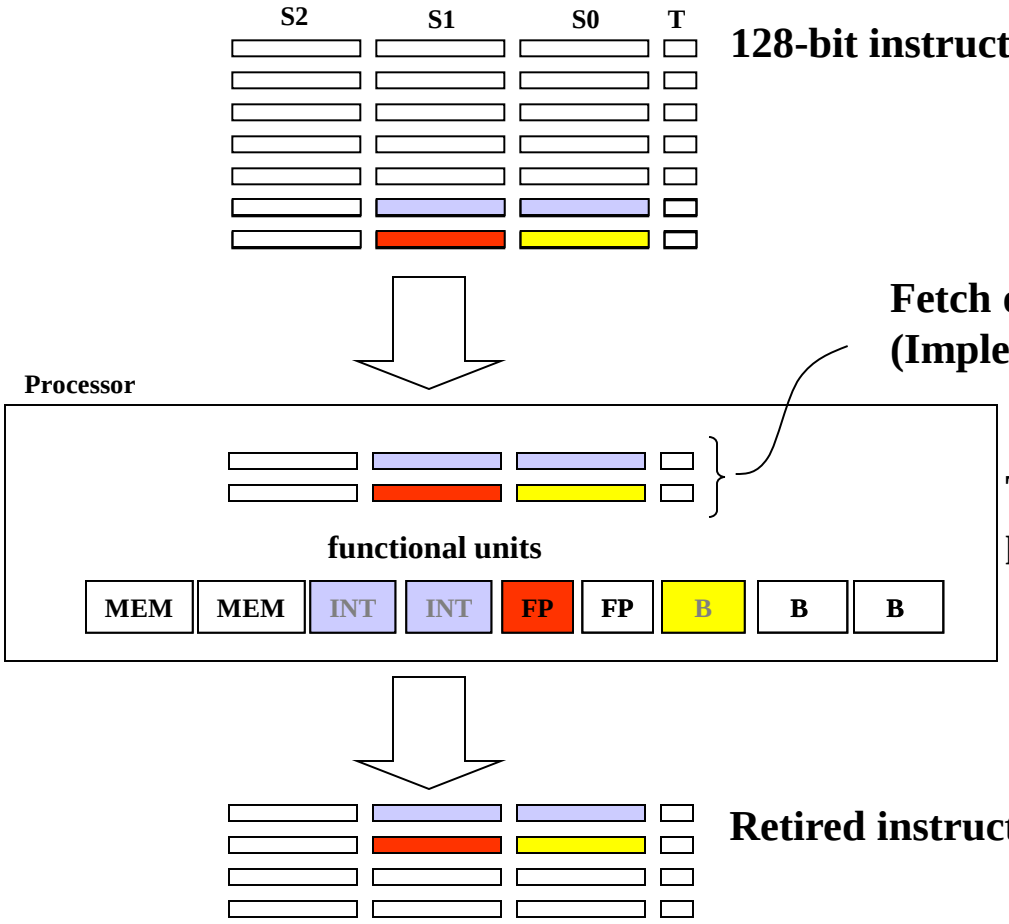
Commercial VLIW Processors

Itanium

Itanium® Processor Family Architecture

- **EPIC**: Explicitly Parallel Instruction Computing
- Instruction encoding
  - Bundles and templates
- Large register resources
  - 128 integer
  - 128 floating point
- Support for
  - Software pipelining
  - Predication, and speculation

Explicitly Parallel Instruction Computing EPIC



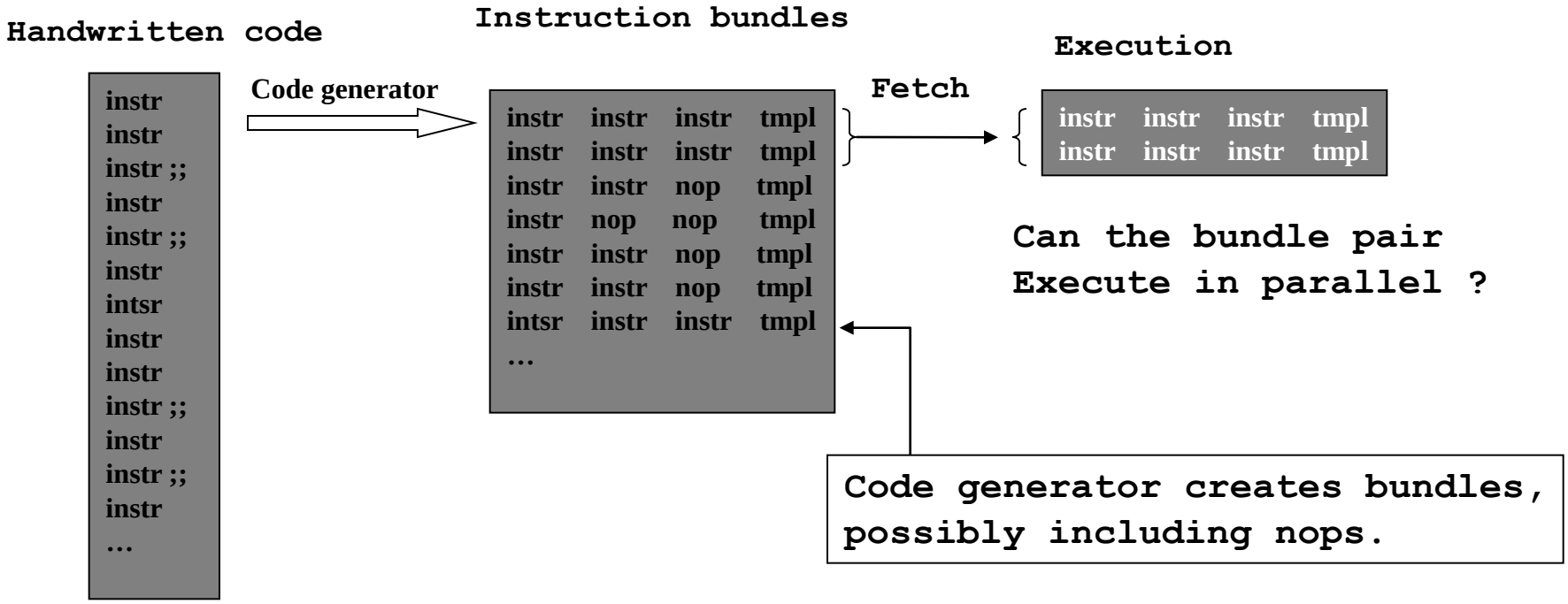
128-bit instruction bundles from I-cache

Fetch one or more bundles for execution (Implementation, Itanium® takes two.)

Try to execute all instructions in parallel, depending on available units.

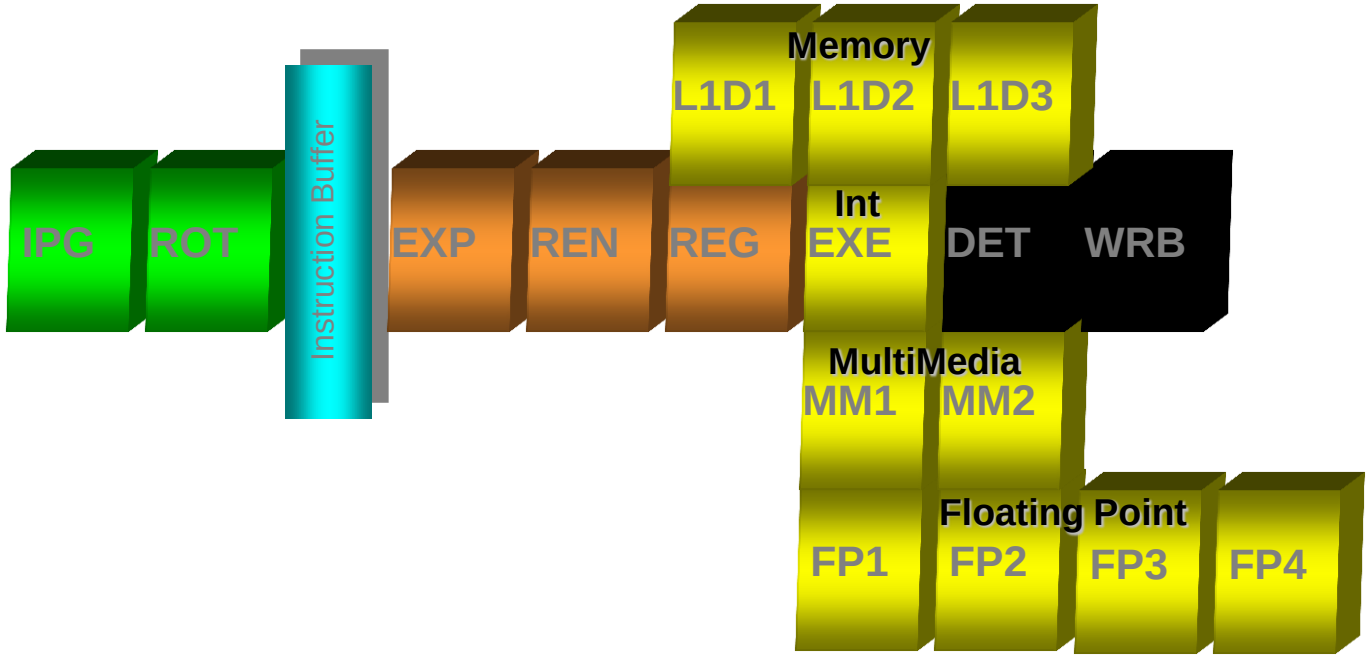
Retired instruction bundles

Itanium® fetches 2 bundles at a time for execution. They may or may not execute in parallel.



There are two difficulties:
1) Finding instruction triplets matching the defined templates.
2) Matching pairs of bundles that can execute in parallel.

Itanium 8-stage Pipelines



- In-order issue, out-of-order completion
  - All functional units are fully pipelined
- Small branch misprediction penalties