COMPUTER ARCHITECTURE

Code: CT173

Part VI: SuperScalar and VLIW

MSc. NGUYEN Huu Van LONG

Department of Computer Networking and Communication,

College of Information & Communication Technology,

CanTho University

Agenda

- **Dynamic Scheduling**
  - General concepts
  - Superscalar architecture
- **Static Scheduling**
  - General concepts
  - VLIW architecture
- **Static Scheduling – Dynamic Scheduling – Pipeline**
- **Explicit Parallel Instruction Computing - EPIC**
- **Itanium Architecture IA – 64**

Recall of ILP techniques

- Micro-architectural techniques that are used to exploit ILP include:
  - Instruction pipelining
  - *Superscalar, VLIW, Explicitly Parallel Instruction Computing EPIC*
  - Out of order execution
  - Register renaming
  - Speculative execution
  - Branch prediction
- We assumed that pipeline/superpipeline technique is implemented on a scalar processor.
- We examined compiler techniques for scheduling the instructions so as to separate dependent instructions and minimize the number of actual hazards and resultant stalls. This approach called ***static scheduling*** **(static parallelism)** became popular with pipelining
- Another approach, that earlier processors used, is called ***dynamic scheduling (dynamic parallelism)***, where the hardware rearranges the instruction execution to reduce the stalls.

Dynamic Scheduling

- A major limitation of the pipelining techniques is that they use in-order instruction issue: *if an instruction is stalled in the pipeline, no later instructions can proceed* → **increase the CPI**
- Example:

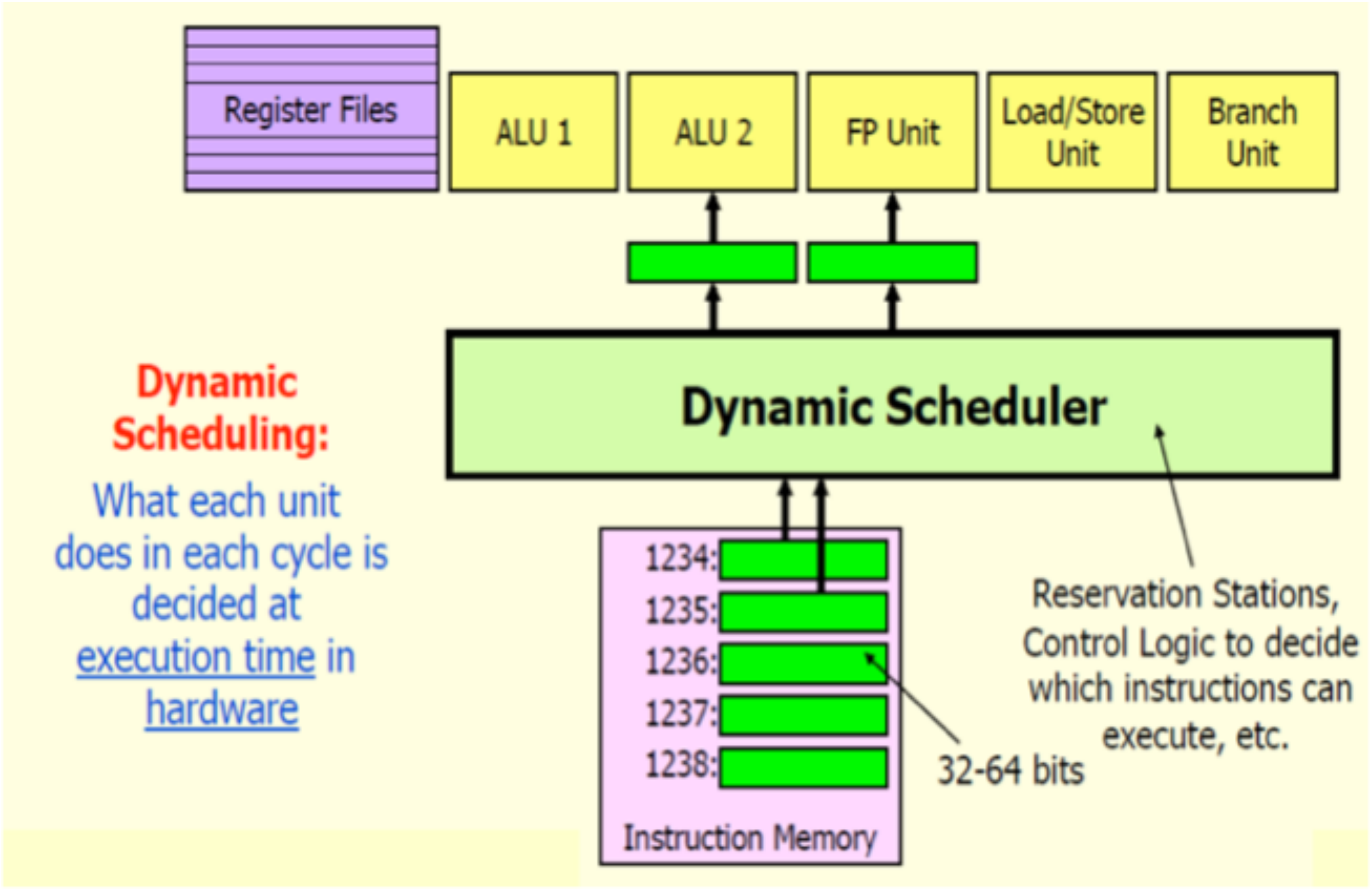
DIV R0, R2, R4	(Long latency operation)
ADD R10, R0, R8	(Depends on DIV)
SUB R12, R8, R14	(Not depends on DIV)

- **Solution**: *out of order execution*
  - Detect dependence of ADD and block it
  - Detect that SUB is not dependent and execute it
  - Not SUB executes before ADD even though it comes after it in program order
  - Hardware must be able to look ahead of blocked instructions
  - Improves utilization of multiple functional units

Dynamic Scheduling

- The hardware *reorder dynamically* the instruction execution to *reduce pipeline stalls* while *maintaining data flow* and *exception behavior*.
- Some combinations are possible
  - **Type 1**: In order issue, execution and completion
  - **Type 2**: In order issue and out of order execution and in order complete
  - **Type 3**: Out of order issue, execution and completion (IMPRACTICAL)
- **Main advantages**
  - Handling unknown dependences at compile time
  - Simplifies the compiler complexity
  - Allows compiled code to run efficiently on a different pipeline
- **Main disadvantages**
  - Significant complexity in hardware
  - Huge power consumption
  - Imprecise exceptions could be appeared
  - Complex design verification
- *Typical example: Scoreboarding, Register Renaming, Tomasulo approach, Out of Order Execution,* ***Superscalar architecture***

Dynamic Scheduling



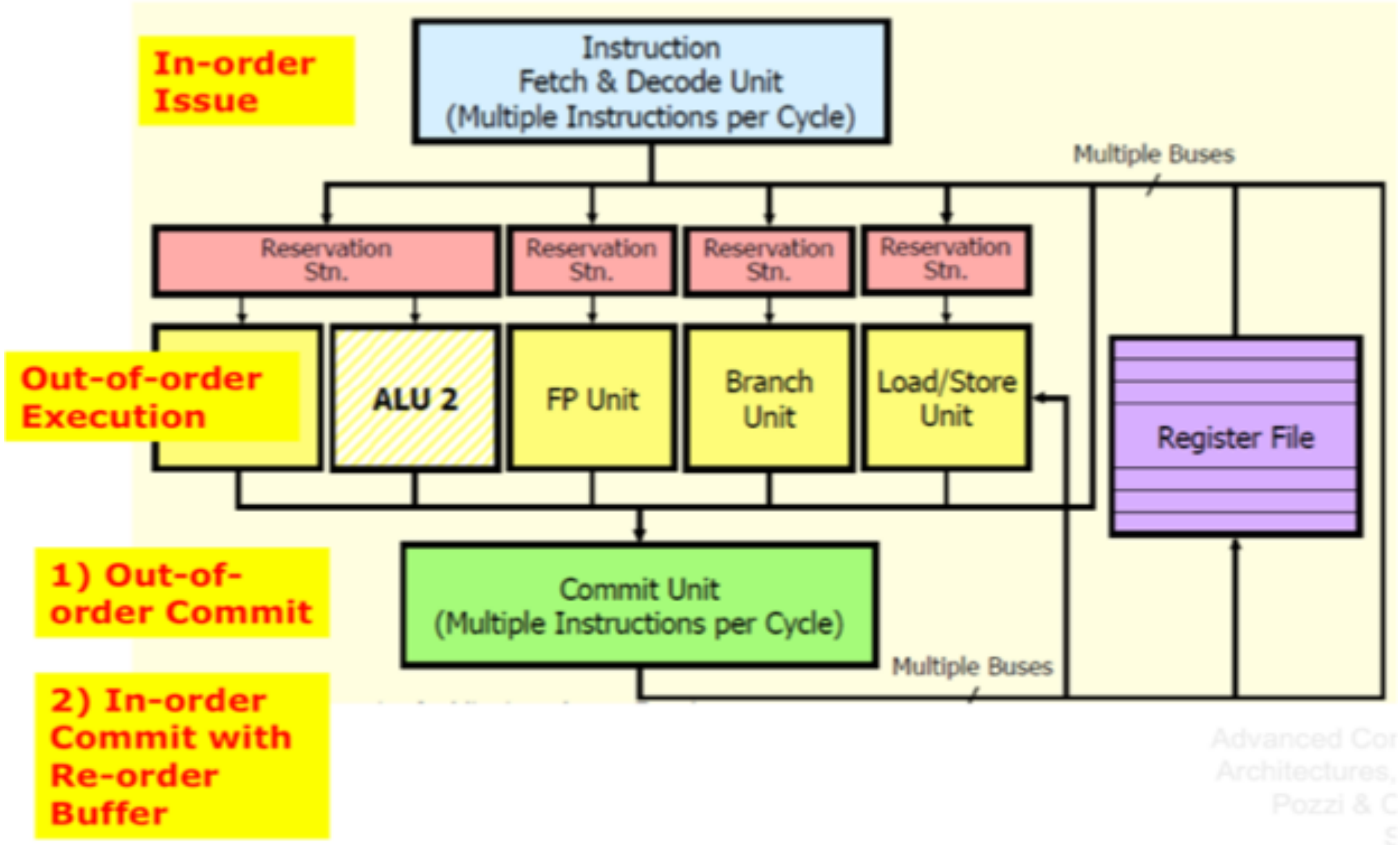
Dynamic Scheduling:

What each unit does in each cycle is decided at execution time in hardware

Superscalar architecture

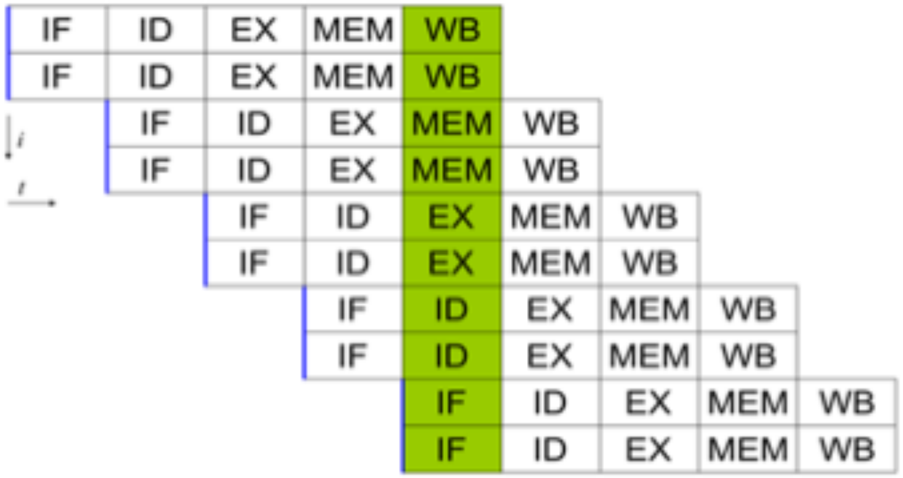
- A scalar processor can execute at most one single instruction per clock cycle
- In contrast, a superscalar processor can *execute more than one instruction during a clock cycle* by *simultaneously dispatching multiple instructions* to *different execution units* on the processor → **increase the Throughput** (Number of Instructions can be executed in a unit of time)
- Superscalar and pipelining execution are considered different performance enhancement techniques
  - **Pipeline**: executes multiple instructions in the same execution unit in parallel by dividing the execution unit into different phases
  - **SuperScalar**: executes multiple instructions in parallel by using multiple execution units
- The Superscalar technique is traditionally associated with several identifying characteristics (within a given CPU):
  - Instructions are issued from a sequential instruction stream → In order issue
  - The CPU dynamically checks for data dependencies between instructions at run time
  - The CPU can execute multiple instructions per clock cycle → Out order execute
  - The CPU commit the results → In or Out order completion

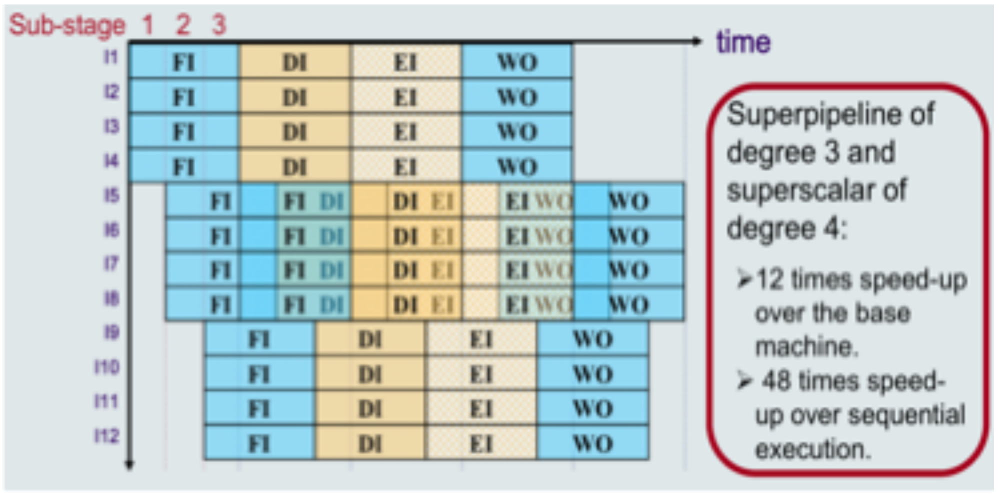
SuperScalar processor: Issue, Execution, Commit



Superscalar architecture

- All high end computer systems now do the combination between Pipeline and SuperScalar
  - Pentium Pro (P6): 3-degree superscalar, 12-stage superpipeline.
  - PowerPC 620: 4-degree superscalar, 4/6-stage pipeline.
  - EV7 Alpha 21364
  - MIPS R10000
  - Intel Core i5, i7
- The hardware determines the maximum number that can be issued from 2 to 6 instructions, normally 4-5 instructions



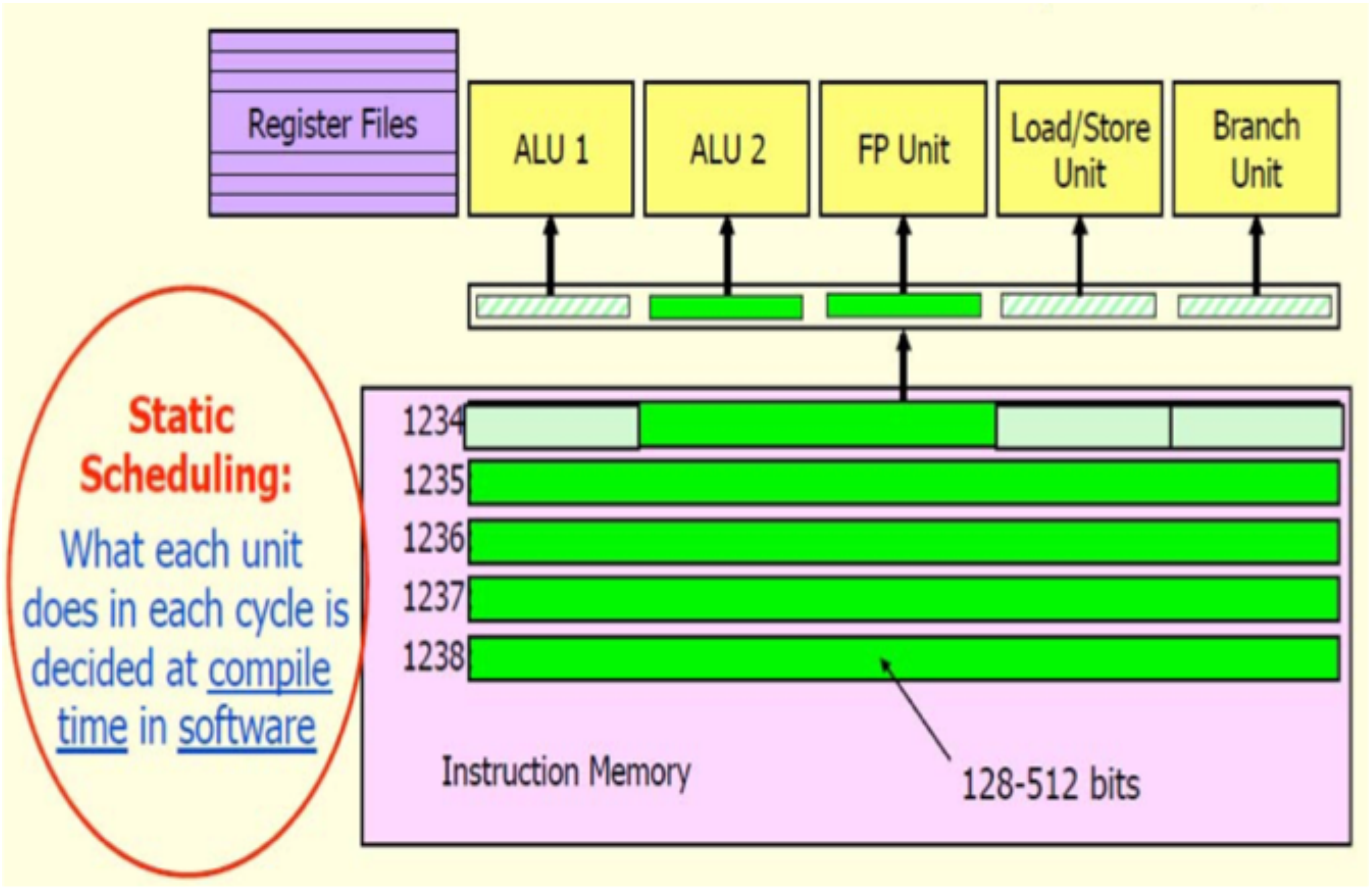


Superpipeline of degree 3 and superscalar of degree 4:
- 12 times speed-up over the base machine.
- 48 times speed-up over sequential execution.

Static Scheduling

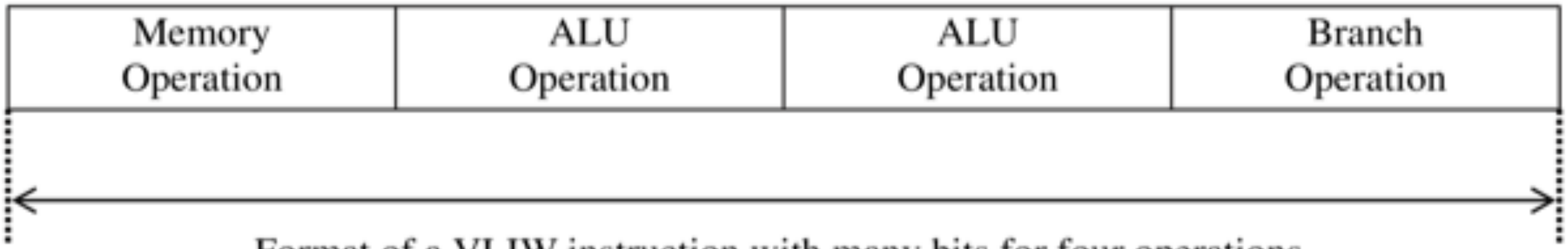
- Static detection and resolution of dependences → accomplished by the sophisticated algorithms of compiler**:** reordered into dependency free code
- Compilers can use sophisticated algorithms for code scheduling to exploit ILP
- The size of a basic block (a straight line code sequence) is usually quite small and the amount of parallelism available within a basic block is quite small
- Main advantages:
  - Simpler HW design with no dynamic scheduler
  - Smaller hardware area
  - Less power consumption
- Main disadvantages:
  - Unpredictable branches
  - Variable memory latency
  - Code size explosion and Code portability
  - Compiler complexity
  - Performance portability
- *Typical example: **VLIW processor***

Static Scheduling



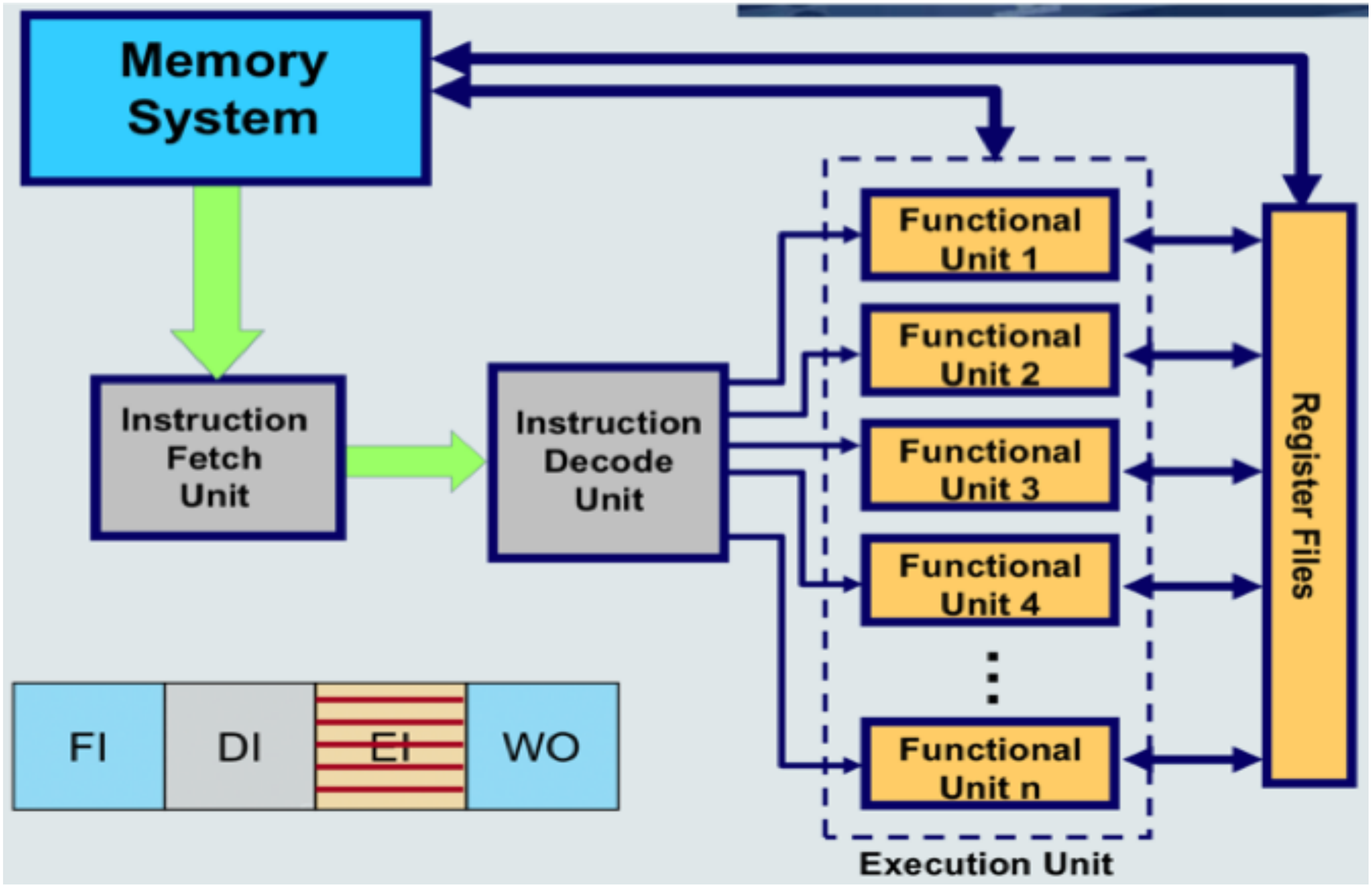
Very Long Instruction Word

- VLIW processor uses a long instruction word which contains multiple fixed number of operations → All operations within a single VLIW instruction must be absolutely independent to process parallel.
- VLIW instruction size: 64 to 128 bits, up to 1024 bits.
- Compiler must perform a detailed analysis on the data flow and control-flow **at compile time**. When the compiler cannot find enough instructions to fill in the width of the long instruction, it simply inserts a NOP.
- An intelligent compiler is the key component which ultimately decides the performance of a VLIW architecture
- Loop unrolling and trace scheduling are the critical VLIW techniques
- VLIW processors are primarily successful as embedded media processors: TriMedia processors, C6000 DSP, STMicroelectronics ST200, SHARC DSP, Itanium 2 – the general purpose VLIW, a 'hybrid' VLIW EPIC



Format of a VLIW instruction with many bits for four operations

Very Long Instruction Word



Superpipeline, Superscalar and VLIW

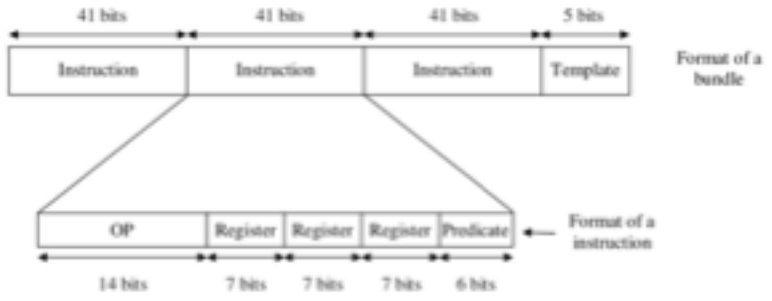
	Superpipelining	Superscalar	VLIW
Approach	Dividing the long latency stages of a pipeline into several shorter stages	Dynamically issuing multiple instructions per cycle	Utilizing the compiler to combine several instructions into one very long instruction and executing it
Effects	Clock cycle time	CPI	IC
Instruction issue rate	1	N	N
Instruction issue style	Dynamic	Dynamic	Dynamic
Difficulty of design	Relatively easy compared to the other two	Complex design issues. Run time tracking	Complex design issues. Compiler support
Keywords	Higher MHz, latch delay, clock skew	Dynamic scheduling, Out of order execution, Register renaming, Score boarding	Compiler support, static scheduling, loop unrolling, trace scheduling

Explicit parallel instruction computing EPIC

- **Explicitly parallel instruction computing (EPIC)** is a term coined in 1997 by the HP-Intel alliance
- VLIW is the basis for the research a new architecture EPIC
- The goal of EPIC is proposed based on the advantages of VLIW:
  - Move the complexity of instruction scheduling from the CPU hardware to the software compiler
  - Eliminates the need for complex scheduling circuitry in the CPU
  - Frees up space and power for other functions, including additional execution resources
- *EPIC architecture evolved from VLIW architecture, but retained many concepts of the superscalar architecture*. To do that, it adds several features to get around the deficiencies of VLIW
  - Each group of multiple software instructions is called a *bundle.* Each of the bundles has a stop bit indicating if this set of operations is depended upon by the subsequent bundle. With this capability, *future implementations can be built to issue multiple bundles in parallel*. The dependency information is calculated by the compiler, so the hardware does not have to perform operand dependency checking.
  - More at: https://en.wikipedia.org/wiki/Explicitly_parallel_instruction_computing

IA-64 architecture

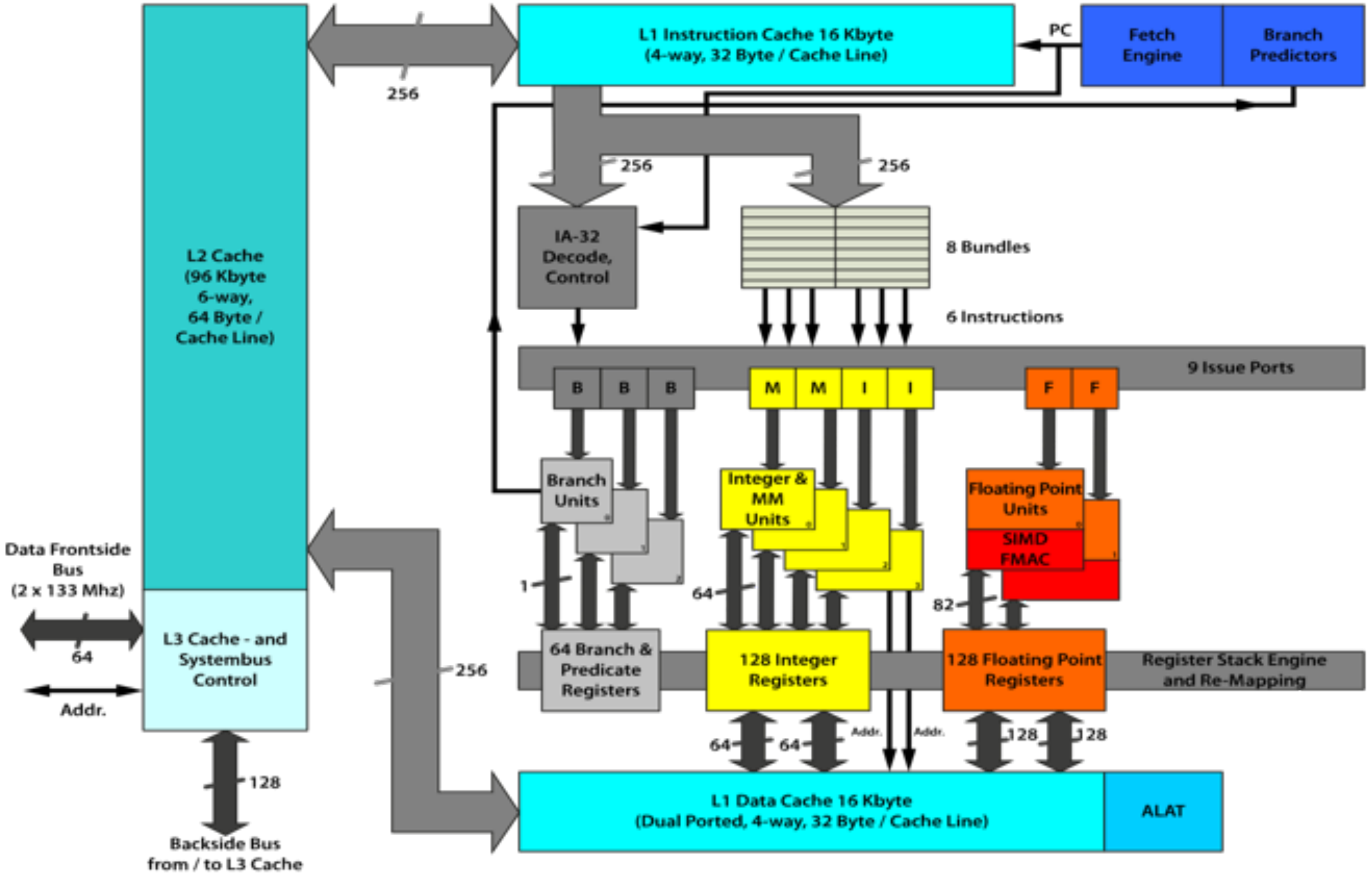
- **IA-64** (also called **Intel Itanium architecture**) is the ISA of the Itanium family of 64-bit Intel microprocessors
- The first Itanium processor, codenamed **Merced** is based on EPIC, was released in 2001. It was proposed as the new generation processor for servers and workstations.
- In all Itanium models, up to and including **Tukwila**, cores execute up to six instructions per clock cycle
- The long instruction word in IA64 has 128-bit wide, could be called a bundle, and holds 3 instructions
  - Each base instruction requires 41-bit wide → three instructions are 123-bit wide
  - The remaining 5 bits can be used as the template to deliver the execution information of the bundle. It could be very useful to avoid empty operations (NOPs) and allow higher flexibility in the exploitation of parallelism.
- Two distinct features of the IA-64 are predication execution and speculative loading



IA-64 architecture

- IA64 provides:
  - 128 65-bit general registers
  - 128 82-bit floating-point registers.
  - 128 64-bit special purpose application registers
  - 64 1-bit register to indicate the result of conditional expression instructions
- The Itanium is implemented with a six-wide and 10 stage pipeline at 800MHz
  - 4 integer units, 4 multimedia units, 2 load/store units, 3 branch units, 2 extended-precision floating-point units, and 2 additional single-precision floating- point units.
  - 2 bundles are fetched in every clock cycle → can fetch 6 instructions at each clock cycle

IA-64 architecture


L1 Instruction Cache 16 Kbyte (4-way, 32 Byte / Cache Line)
PC
Fetch Engine
Branch Predictors
256
L2 Cache (96 Kbyte 6-way, 64 Byte / Cache Line)
256
256
IA-32 Decode, Control
8 Bundles
6 Instructions
9 Issue Ports
B
B
B
M
M
I
I
F
F
Branch Units
Integer & MM Units
Floating Point Units
SIMD FMAC
Data Frontside Bus (2 x 133 Mhz)
64
Addr.
L3 Cache - and Systembus Control
1
64
82
256
64 Branch & Predicate Registers
128 Integer Registers
128 Floating Point Registers
Register Stack Engine and Re-Mapping
64
64
Addr.
Addr.
128
128
128
L1 Data Cache 16 Kbyte (Dual Ported, 4-way, 32 Byte / Cache Line)
ALAT
Backside Bus from / to L3 Cache

Question?