(Advanced) Computer Architechture

Prof. Dr. Hasan Hüseyin BALIK

(10th Week)

Outline

4. The central processing unit
   - Processor Structure and Function
   - Reduced Instruction Set Computers (RISCs)
   - Instruction-Level Parallelism and Superscalar Processors
   - Control Unit Operation and Microprogrammed Control

4.3 Instruction-Level Parallelism and Superscalar Processors

4.3 Outline

- Overview
- Design Issues
- Intel Core Microarchitecture
- ARM Cortex-A8
- ARM Cortex-M3

Superscalar Overview

Term first coined in 1987

Refers to a machine that is designed to improve the performance of the execution of scalar instructions

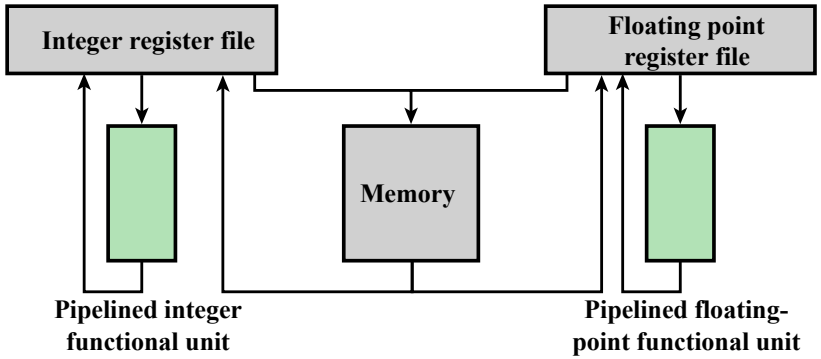
In most applications the bulk of the operations are on scalar quantities

Represents the next step in the evolution of high-performance general-purpose processors

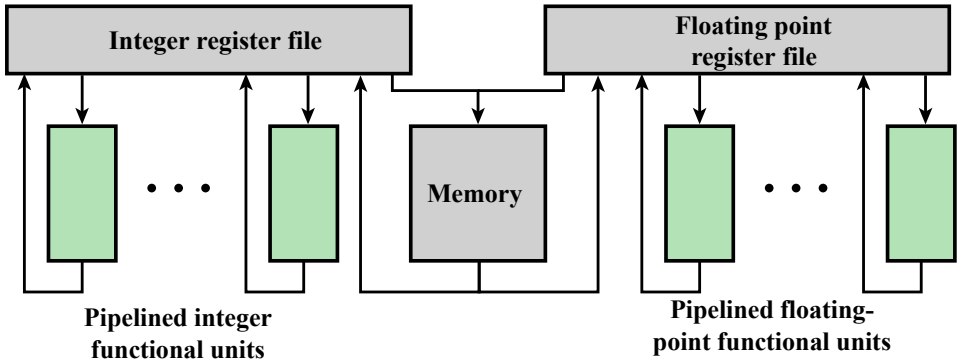
Essence of the approach is the ability to execute instructions independently and concurrently in different pipelines

Concept can be further exploited by allowing instructions to be executed in an order different from the program order

Superscalar Organization Compared to Ordinary Scalar Organization



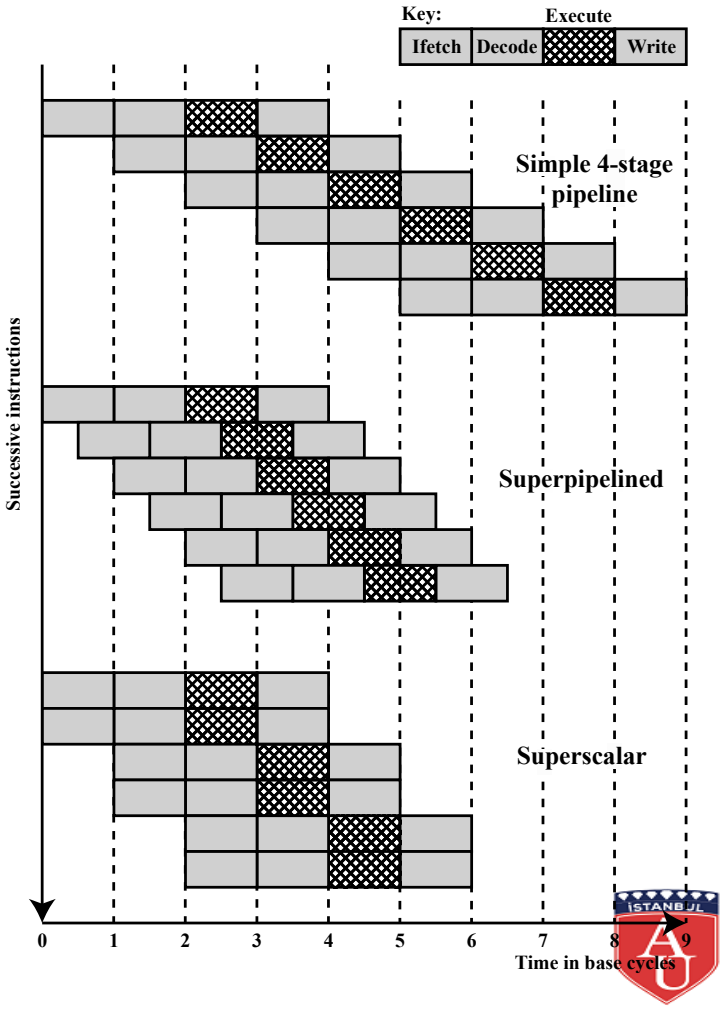
(a) Scalar organization



(b) Superscalar organization

Comparison of Superscalar and Superpipeline Approaches

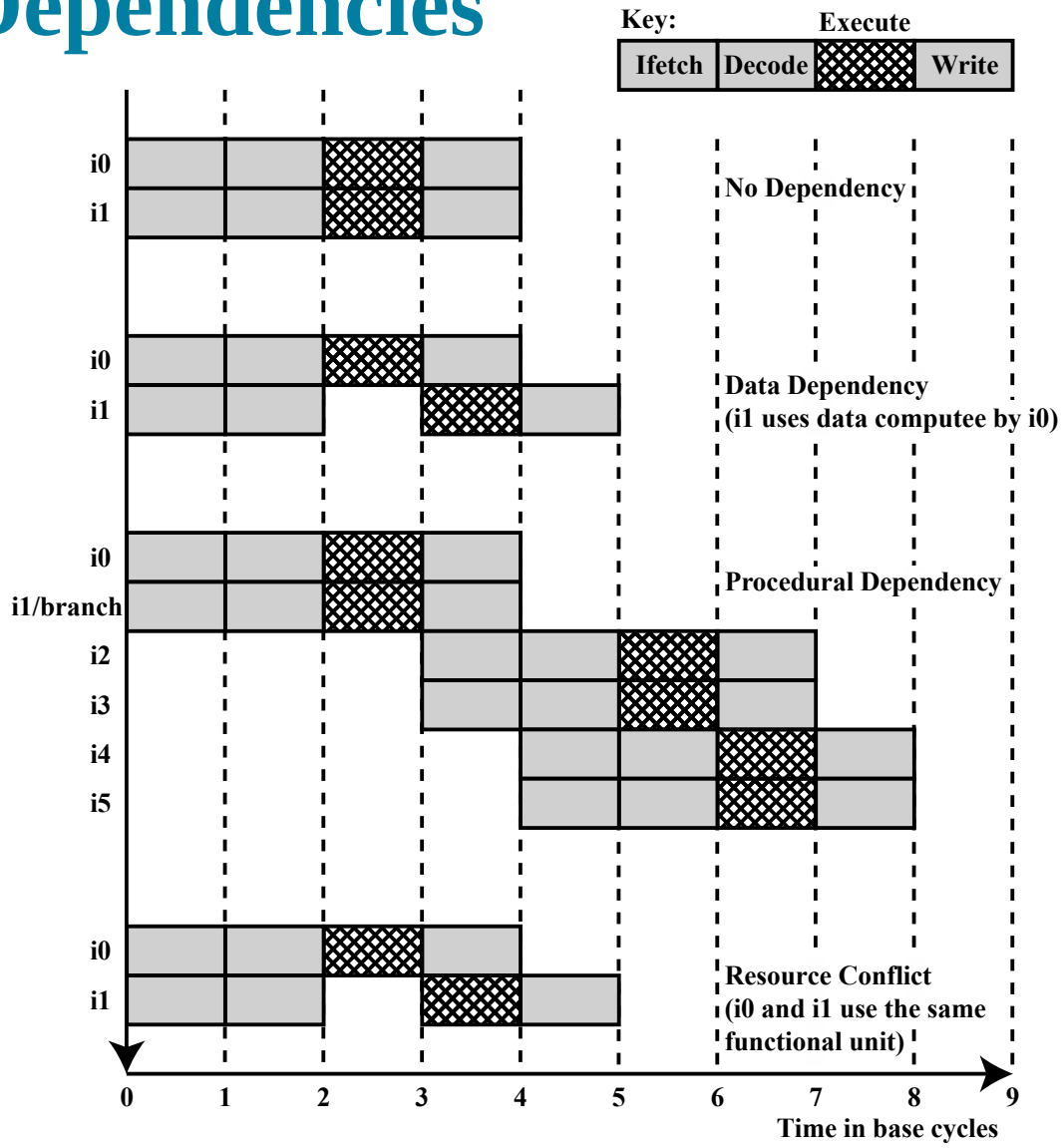
- An alternative approach to achieving greater performance is referred to as superpipelining
- A term first coined in 1988
- Superpipelining divides the pipeline into a greater number of smaller stages in order to clock it at a higher frequency
- There is still only one pipeline, but by increasing the number of stages,we increase its temporal parallelism
- This use of a very deep, very high-speed pipeline for instruction processing is called superpipelining
- MIPS R4000



Constraints

- Instruction level parallelism
  - Refers to the degree to which the instructions of a program can be executed in parallel
  - A combination of compiler based optimization and hardware techniques can be used to maximize instruction level parallelism
- Limitations:
  - True data dependency
  - Procedural dependency
  - Resource conflicts
  - Output dependency
  - Antidependency

Effect of Dependencies

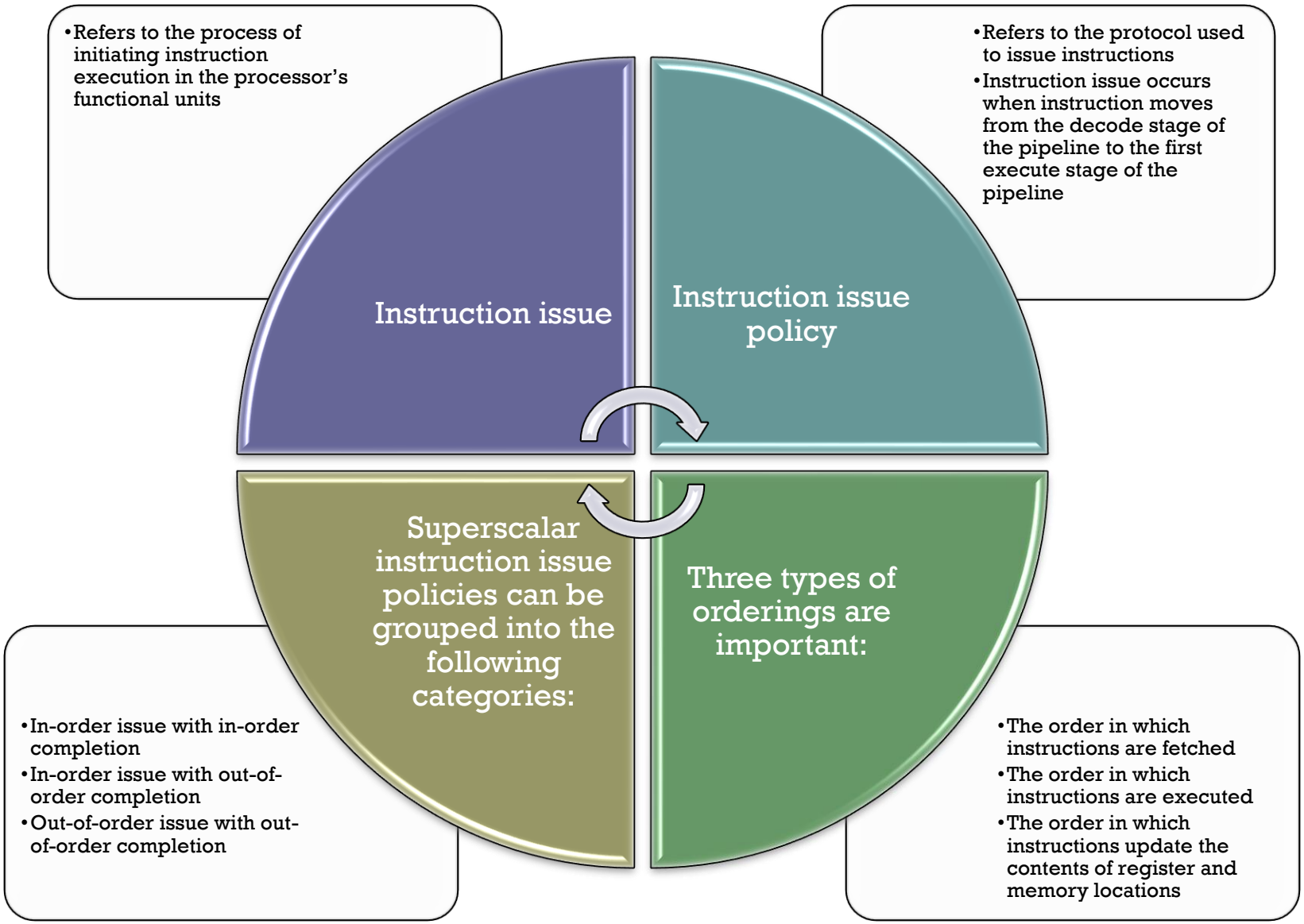


Design Issues

Instruction-Level Parallelism
and Machine Parallelism

- Instruction level parallelism
  - Instructions in a sequence are independent
  - Execution can be overlapped
  - Governed by data and procedural dependency
- Machine Parallelism
  - Ability to take advantage of instruction level parallelism
  - Governed by number of parallel pipelines

Instruction Issue Policy



Superscalar Instruction Issue and Completion Policies

- I1 requires two cycles to execute.
- I3 and I4 conflict for the same functional unit.
- I5 depends on the value produced by I4.
- I5 and I6 conflict for a functional unit.

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

(a) In-order issue and in-order completion

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

(b) In-order issue and out-of-order completion

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1,I2*	I1	I2				2
I5	I6	*I3,I4*	I1		I3	I2		3
		I4,I5,I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

(c) Out-of-order issue and out-of-order completion

Register Renaming (1 of 2)

Output and antidependencies occur because register contents may not reflect the correct ordering from the program

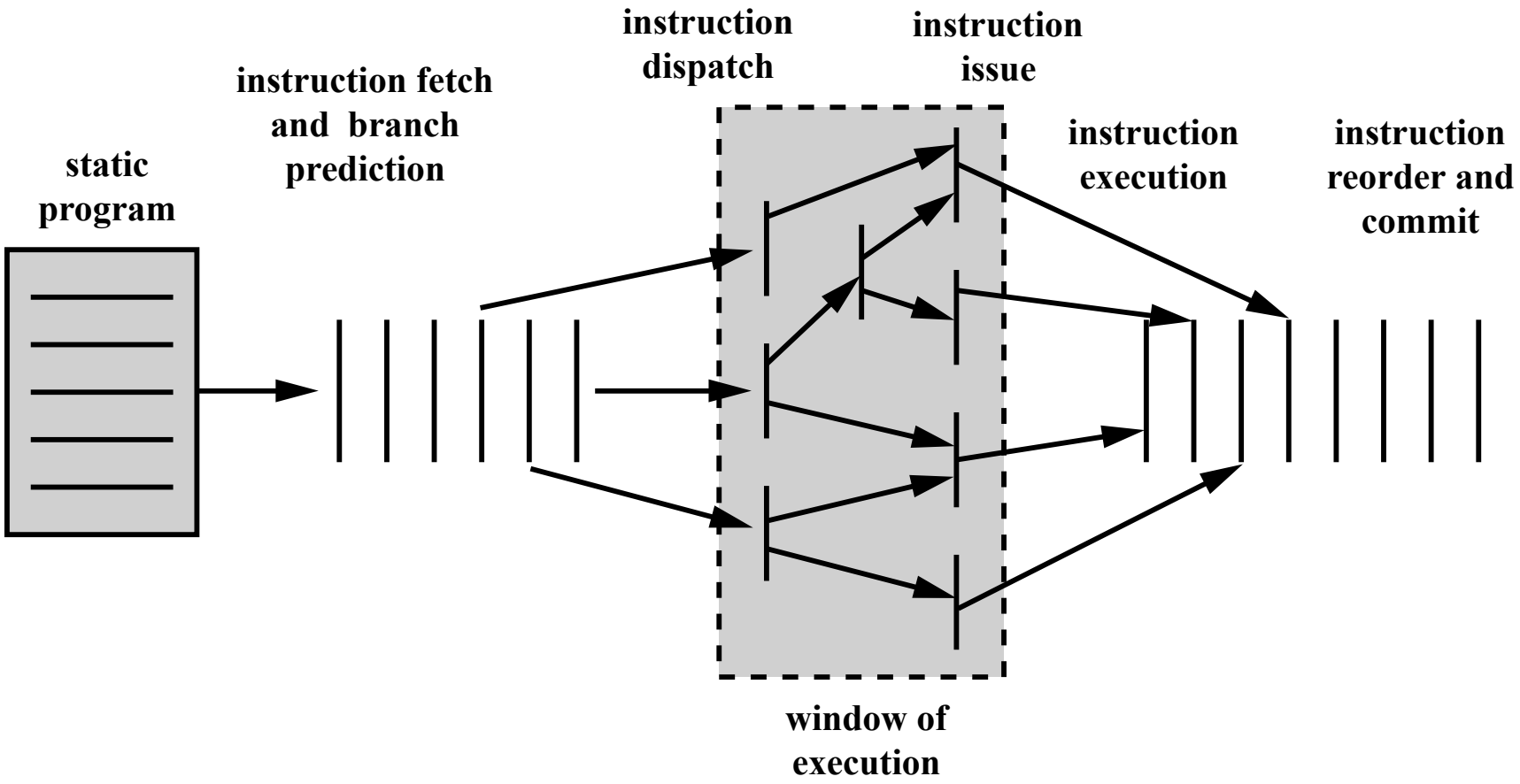
May result in a pipeline stall

Registers allocated dynamically

Branch Prediction

- Any high-performance pipelined machine must address the issue of dealing with branches
- Intel 80486 addressed the problem by fetching both the next sequential instruction after a branch and speculatively fetching the branch target instruction
- RISC machines:
  - Delayed branch strategy was explored
  - Processor always executes the single instruction that immediately follows the branch
  - Keeps the pipeline full while the processor fetches a new instruction stream
- Superscalar machines:
  - Delayed branch strategy has less appeal
  - Have returned to pre-RISC techniques of branch prediction

Conceptual Depiction of Superscalar Processing



Superscalar Implementation

Key elements:

- Instruction fetch strategies that simultaneously fetch multiple instruction
- Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution
- Mechanisms for initiating, or issuing, multiple instructions in parallel
- Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references
- Mechanisms for committing the process state in correct order

Intel Core Microarchitecture

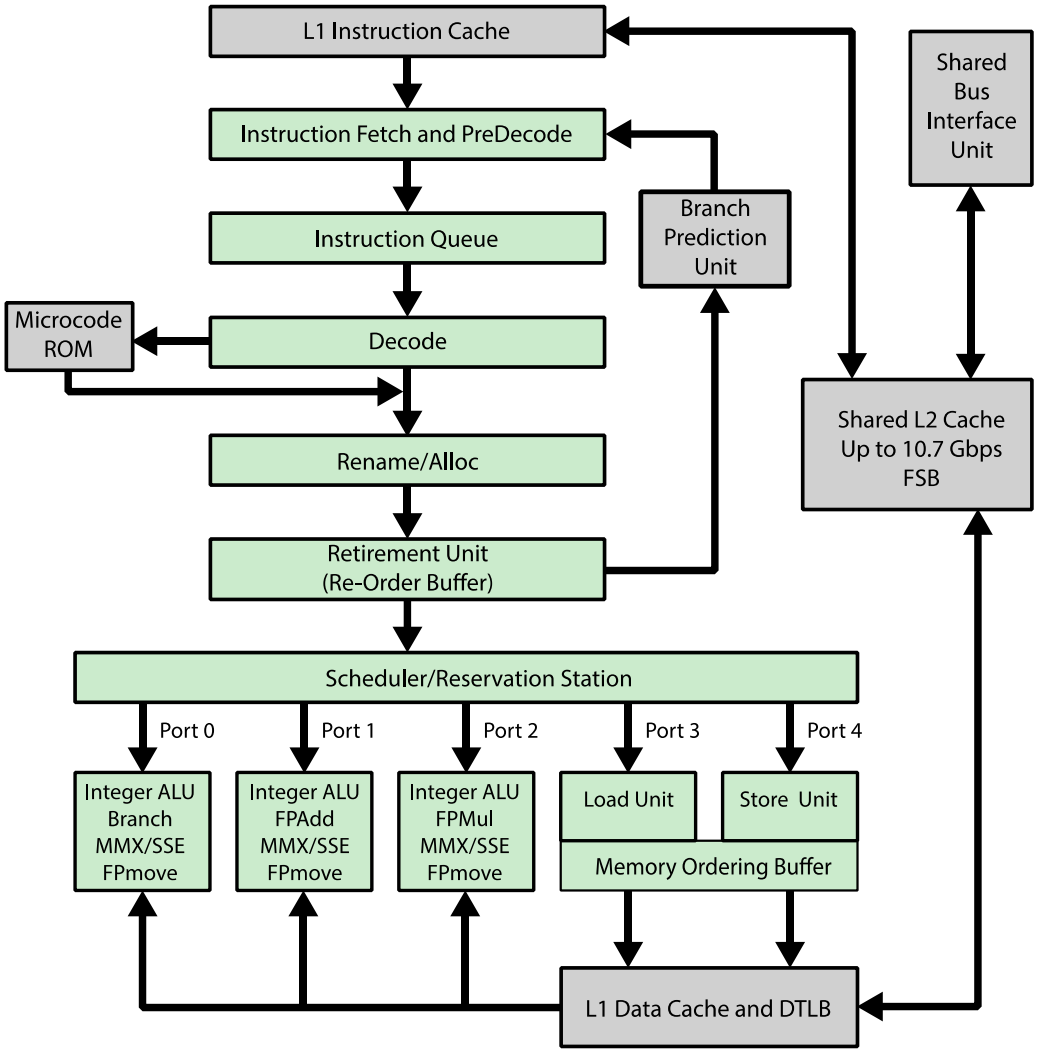


Table:

Cache/Memory Parameters and Performance of Processors Based on Intel Core Microarchitecture

(a) Cache Parameters				
Cache Level	**Capacity**	**Associativity (ways)**	**Line Size (bytes)**	**Writeback Update Policy**
L1 data	32 kB	8	64	Writeback
L1 instruction	32 kB	8	N/A	N/A
L2 (shared)[1]	2, 4 MB	8 or 16	64	Writeback
L2 (shared)[2]	3, 6 MB	12 or 24	64	Writeback
L3 (shared)[2]	8, 12, 16 MB	15	64	Writeback

Notes:
1. Intel Core Microarchitecture
2. Enhanced Intel Core Microarchitecture

(b) Load/Store Performance				
Data Locality	Load		Store	
	Latency	**Throughput**	**Latency**	**Throughput**
L1 data cache	3 clock cycles	1 clock cycle	2 clock cycles	3 clock cycles
L1 data cache of the other core in modified state	14 clock cycles + 5.5 bus cycles	14 clock cycles + 5.5 bus cycles	14 clock cycles + 5.5 bus cycles	N/A
L2 cache	14	3	14	3
Memory	14 clock cycles + 5.5 bus cycles + memory latency	Depends on bus read protocol	14 clock cycles + 5.5 bus cycles + memory latency	Depends on bus read protocol

Front End

Consists of three major components:

- Branch prediction unit (BPU)
- Instruction fetch and predecode unit
- Instruction queue and decode unit

Branch Prediction Unit

- Helps the instruction fetch unit fetch the most likely instruction to be executed by predicting the various branch types:
  - Conditional
  - Indirect
  - Direct
  - Call
  - Return
- Uses dedicated hardware for each branch type
- Enables the processor to begin executing instructions long before the branch outcome is decided
- A branch target buffer (BTB) is maintained that caches information about recently encountered branch instructions

Instruction Fetch and Predecode Unit

Comprises:

- The instruction translation lookaside buffer (ITLB)
- An instruction prefetcher
- The instruction cache
- The predecode logic

The predecode unit accepts the sixteen bytes from the instruction cache or prefetch buffers and carries out the following tasks:

- Determine the length of the instructions
- Decode all prefixes associated with instructions
- Mark various properties of instruction for the decoders

Predecode unit can write up to six instructions per cycle into the instruction queue

- If a fetch contains more than six instructions, the predecoder continues to decode up to six instructions per cycle until all instruction in the fetch are written to the instruction queue
- Subsequent fetches can only enter predecoding after the current fetch completes

Instruction Queue and Decode Unit

- Fetched instructions are placed in an instruction queue
  - From there the decode unit scans the bytes to determine instruction boundaries
  - The decoder translates each machine instruction from one to four micro-ops
    - Each of which is a 118-bit RISC instruction
- A few instructions require more than four micro-ops so they are transferred to microcode ROM, which contains the series of micro-ops (five or more) associated with a complex machine instruction
- The resulting micro-op sequence is delivered to the rename/allocator module

Out-of-Order Execution Logic

- This part of the processor reorders micro-ops to allow them to execute as quickly as their input operands are ready
- Allocate stage
  - Allocates resources required for execution
  - Performs the following functions:
    - If a needed resource is unavailable for one of the three micro-ops arriving at the allocator during a clock cycle, the allocator stalls the pipeline
    - Allocates a reorder buffer (ROB) entry which tracks the completion status of one of the 126 micro-ops that could be in process at any time
    - Allocates one of the 128 integer or floating-point register entries for the result data value of the micro-op, and possibly a load or store buffer used to track one of the 48 loads or 24 stores in the machine pipeline
    - Allocates an entry in one of the two micro-op queues in front of the instruction schedulers

Reorder Buffer (ROB)

Circular buffer that can hold up to 126 micro-ops and also contains the 128 hardware registers

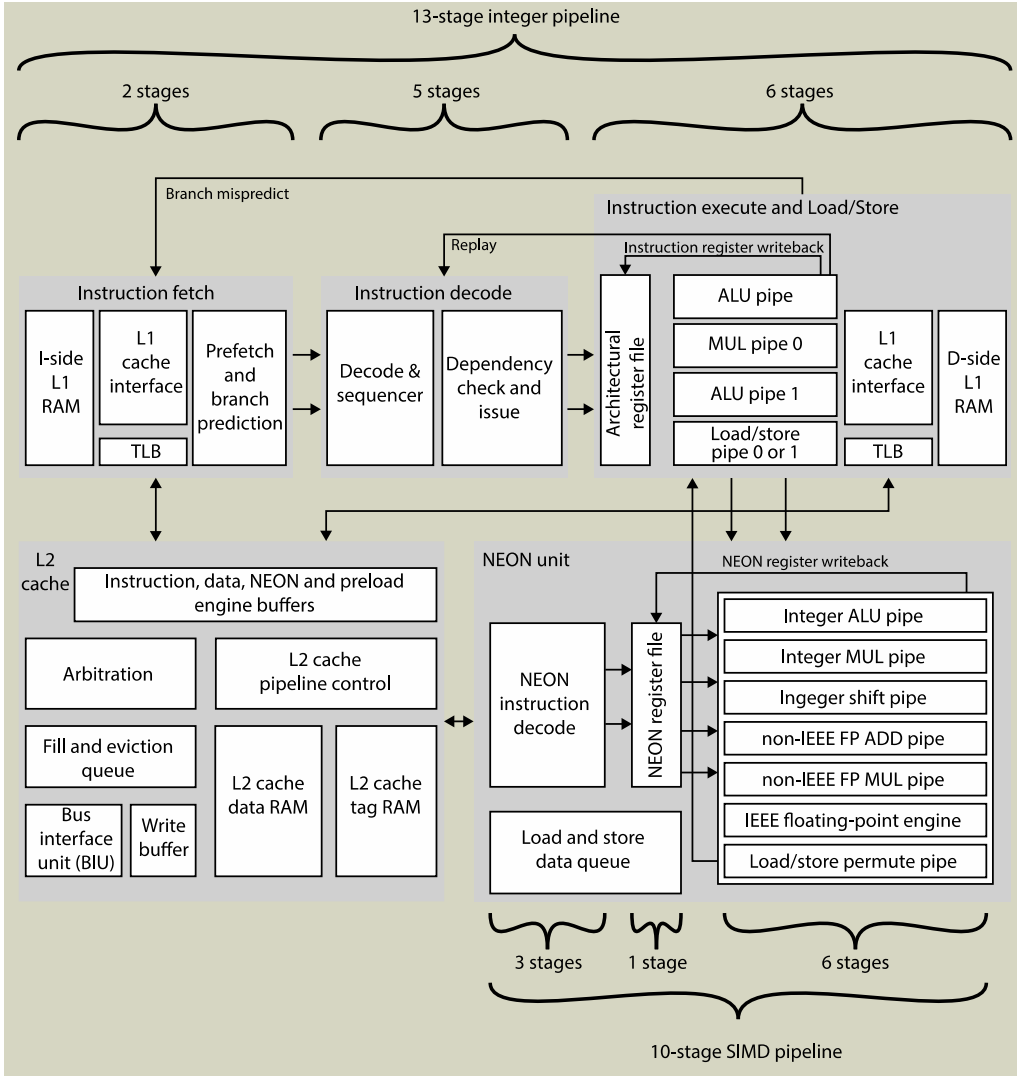
Each buffer entry consists of the following fields:

- **State**
  - Indicates whether this micro-op is scheduled for execution, has been dispatched for execution, or has completed execution and is ready for retirement
- **Memory address**
  - The address of the Pentium instruction that generated the micro-op
- **Micro-op**
  - The actual operation
- **Alias register**
  - If the micro-op references one of the 16 architectural registers, this entry redirects that reference to one of the 128 hardware registers

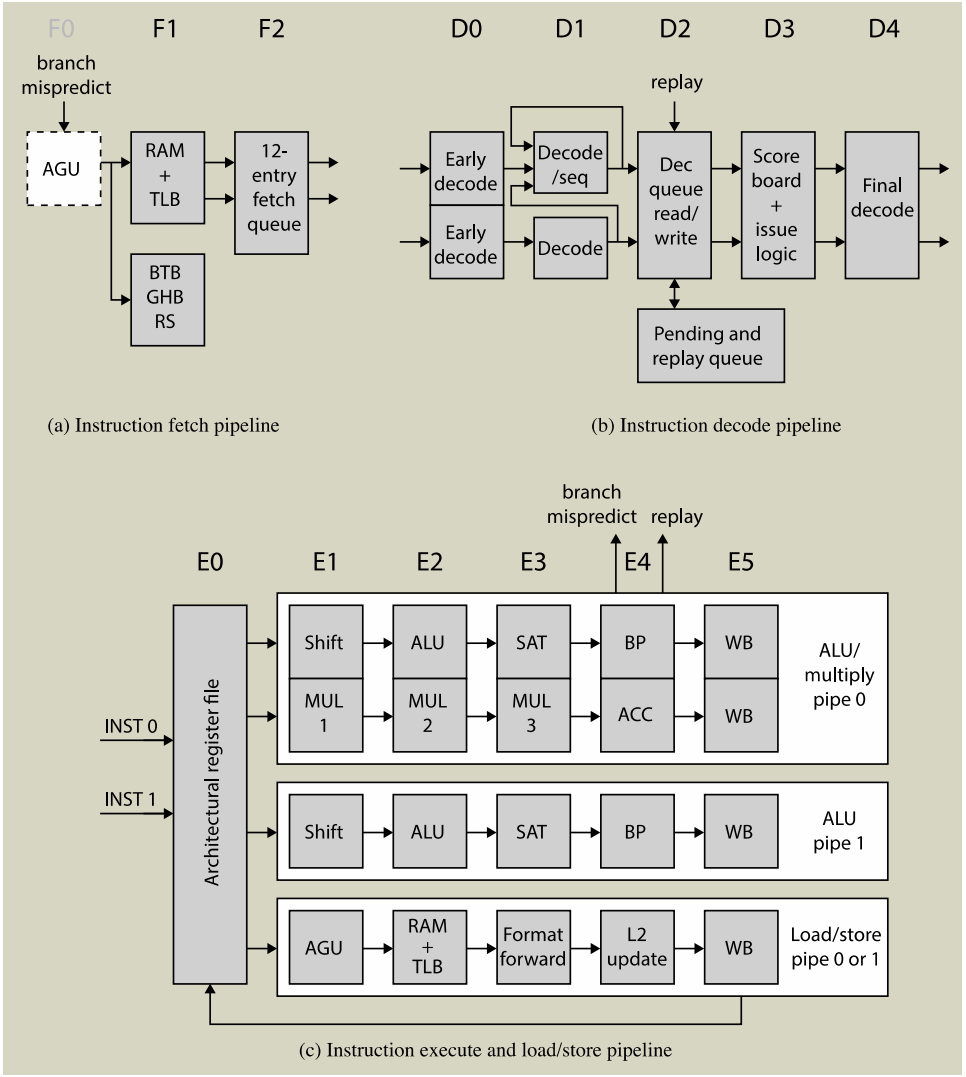
Register Renaming (2 of 2)

- Register renaming
  - The rename stage remaps references to the 16 architectural registers into a set of 128 physical registers
- Micro-op scheduling and dispatching
  - Schedulers are responsible for retrieving micro-ops from the micro-op queues and dispatching these for execution
- Micro-op queuing
  - After resource allocation and register renaming, micro-ops are placed in one of two micro-op queues, where they are held until there is room in the schedulers
- Integer and floating-point execution units
  - The execution units retrieve values from the register files as well as from the L1 data cache

Architectural Block Diagram of ARM Cortex-A8



ARM Cortex-A8 Integer Pipeline



Instruction Fetch Unit

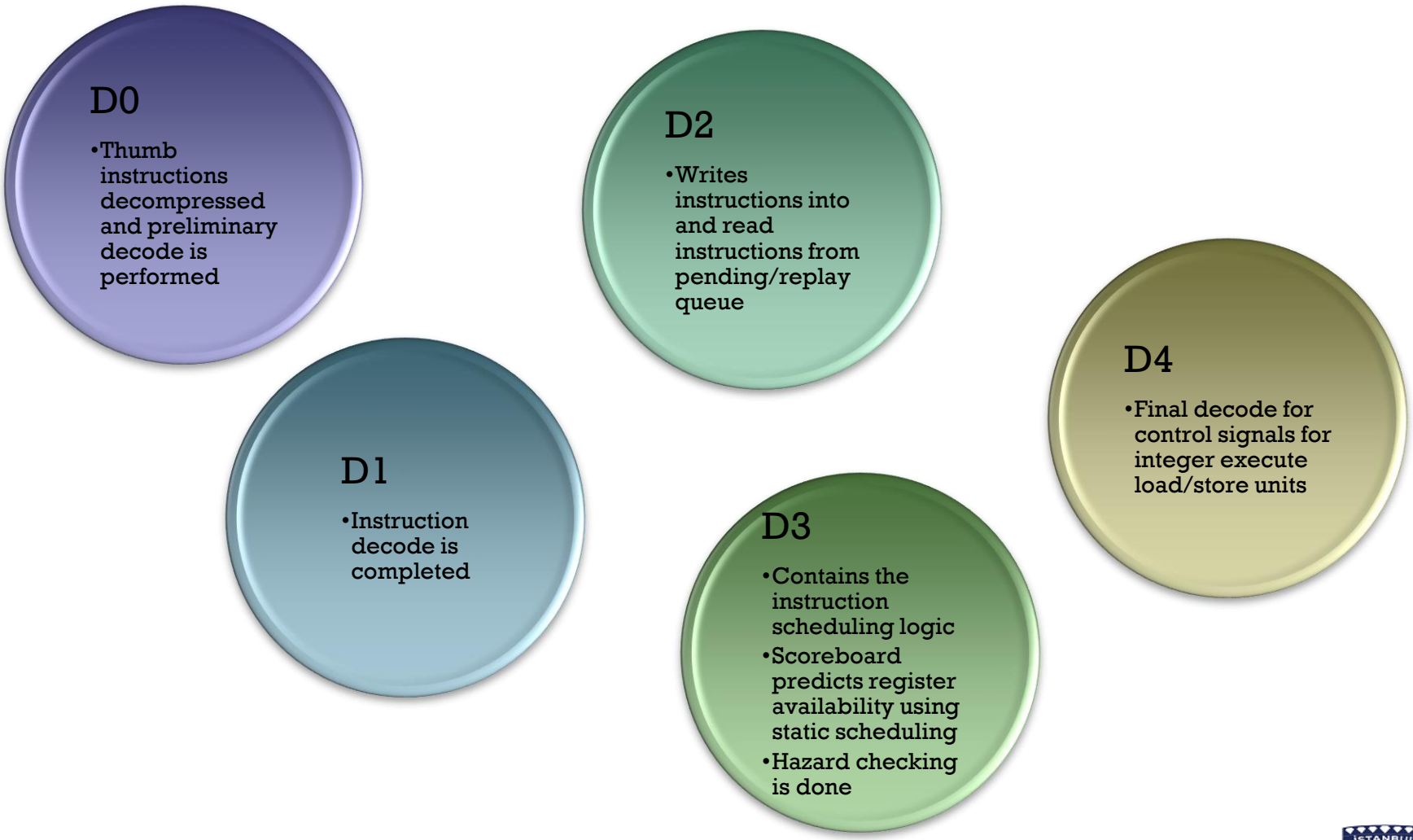
- Predicts instruction stream
- Fetches instructions from the L1 instruction cache
- Places the fetched instructions into a buffer for consumption by the decode pipeline
- Also includes the L1 instruction cache
- Speculative (there is no guarantee that they are executed)
- Branch or exceptional instruction in the code stream can cause a pipeline flush
- Can fetch up to four instructions per cycle

- F0
  - Address generation unit (AGU) generates a new virtual address
  - Not counted as part of the 13-stage pipeline
- F1
  - The calculated address is used to fetch instructions from the L1 instruction cache
  - In parallel, the fetch address is used to access branch prediction arrays
- F3
  - Instruction data are placed in the instruction queue
  - If an instruction results in branch prediction, new target address is sent to the address generation unit

Instruction Decode Unit

- Decodes and sequences all ARM and Thumb instructions
- Dual pipeline structure, *pipe0* and *pipe1*
  - Two instructions can progress at a time
  - Pipe0 contains the older instruction in program order
  - If instruction in pipe0 cannot issue, instruction in pipe1 will not issue
- All issued instructions progress in order
- Results written back to register file at end of execution pipeline
  - Prevents WAR hazards
  - Keeps track of WAW hazards and recovery from flush conditions straightforward
- Main concern of decode pipeline is prevention of RAW hazards

Instruction Processing Stages



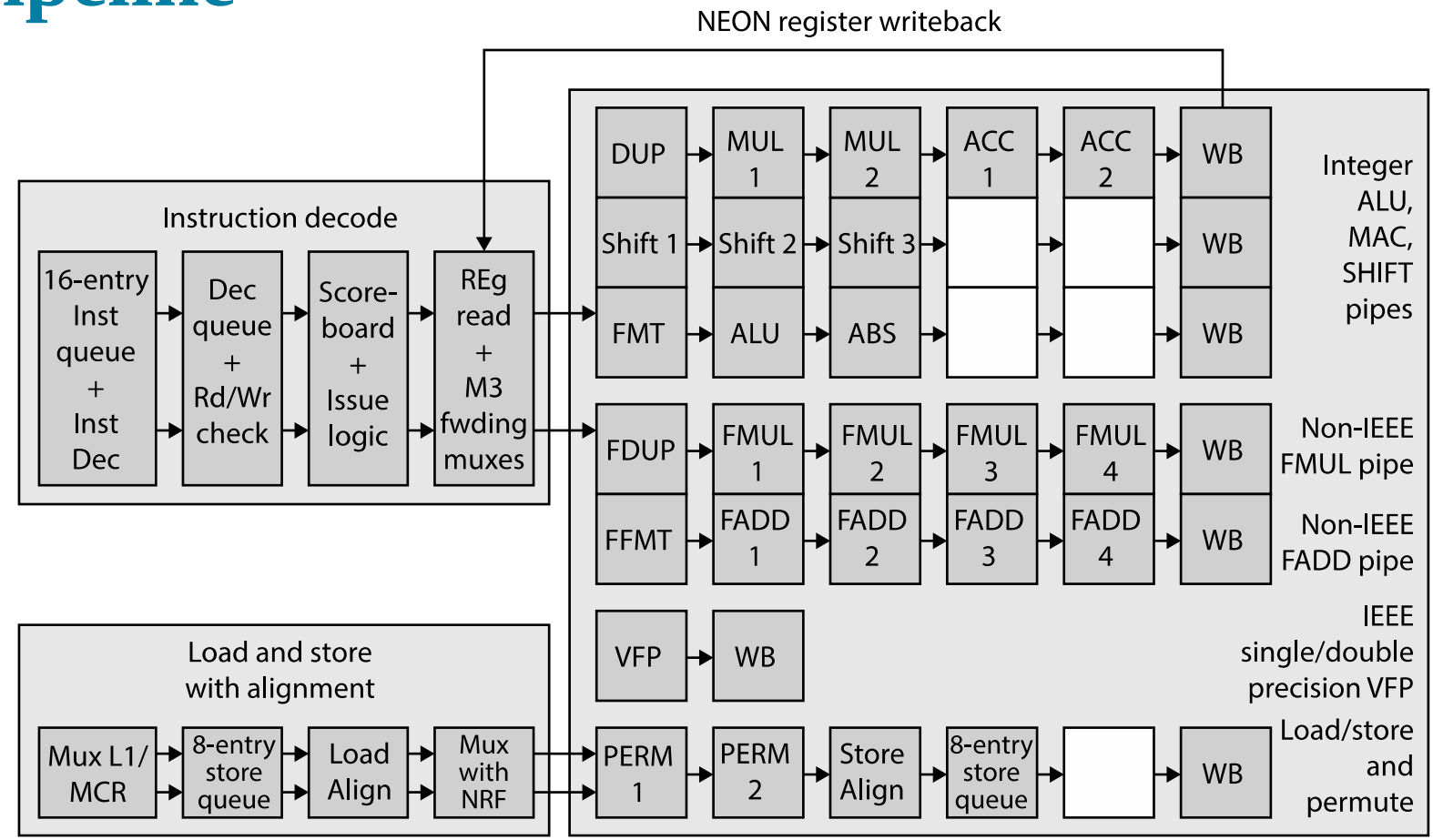
Integer Execute Unit

- Consists of:
  - Two symmetric arithmetic logic unit (ALU) pipelines
  - An address generator for load and store instructions
  - The multiply pipeline
- The instruction execute unit:
  - Executes all integer ALU and multiply operations, including flag generation
  - Generates the virtual addresses for loads and stores and the base write-back value, when required
  - Supplies formatted data for stores and forwards data and flags
  - Processes branches and other changes of instruction stream and evaluates instruction condition codes
- For ALU instructions, either pipeline can be used, consisting of the following stages:
  - E0
    - Access register file
    - Up to six registers for two instructions
  - E1
    - Barrel shifter if needed.
  - E2
    - ALU function
  - E3
    - If needed, completes saturation arithmetic
  - E4
    - Change in control flow prioritized and processed
  - E5
    - Results written back to register file

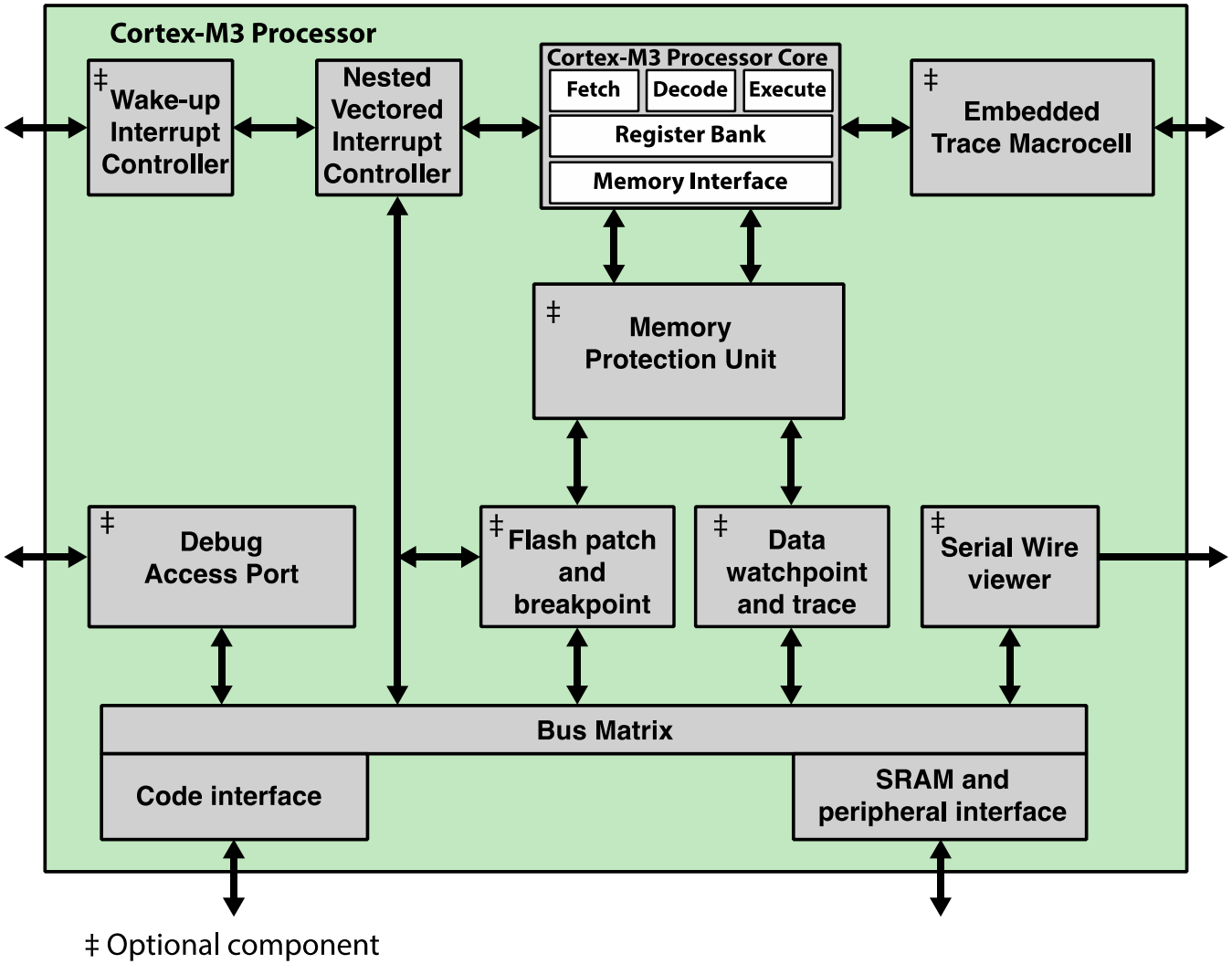
Load/Store Pipeline

- Runs parallel to integer pipeline
- E1
  - Memory address generated from base and index register
- E2
  - Address applied to cache arrays
- E3
  - Load -- data are returned and formatted
  - Store -- data are formatted and ready to be written to cache
- E4
  - Updates L2 cache, if required
- E5
  - Results are written back into the register file

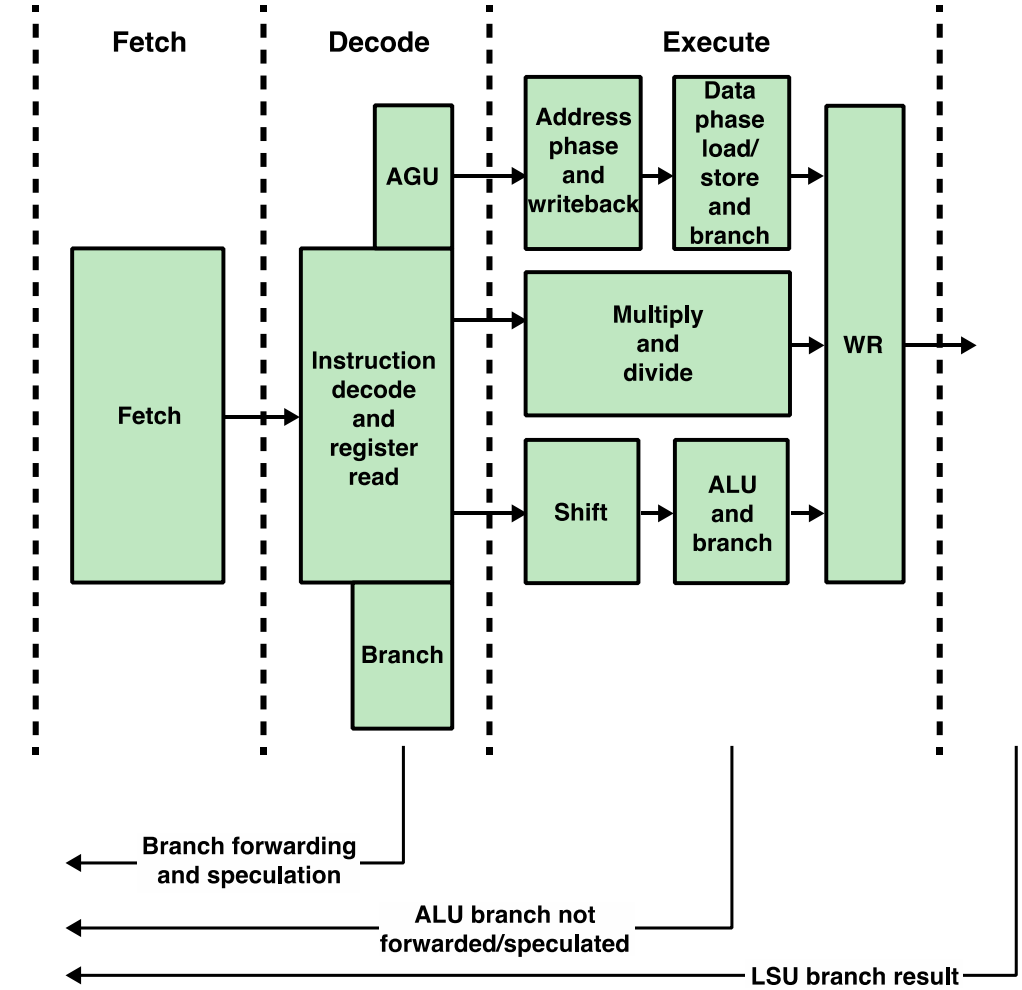
ARM Cortex-A8 NEON and Floating-Point Pipeline



ARM Cortex-M3 Block Diagram



ARM Cortex-M3 Pipeline



AGU = address generation unit