+

SEN361 Computer Organization

Prof. Dr. Hasan Hüseyin BALIK
(10th Week)

+

Outline

3. The Central Processing Unit

+

3.5 Instruction-Level Parallelism and Superscalar Processors

+

3.5 Outline

- Overview
- Design Issues
- Pentium 4
- Arm Cortex-A8

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

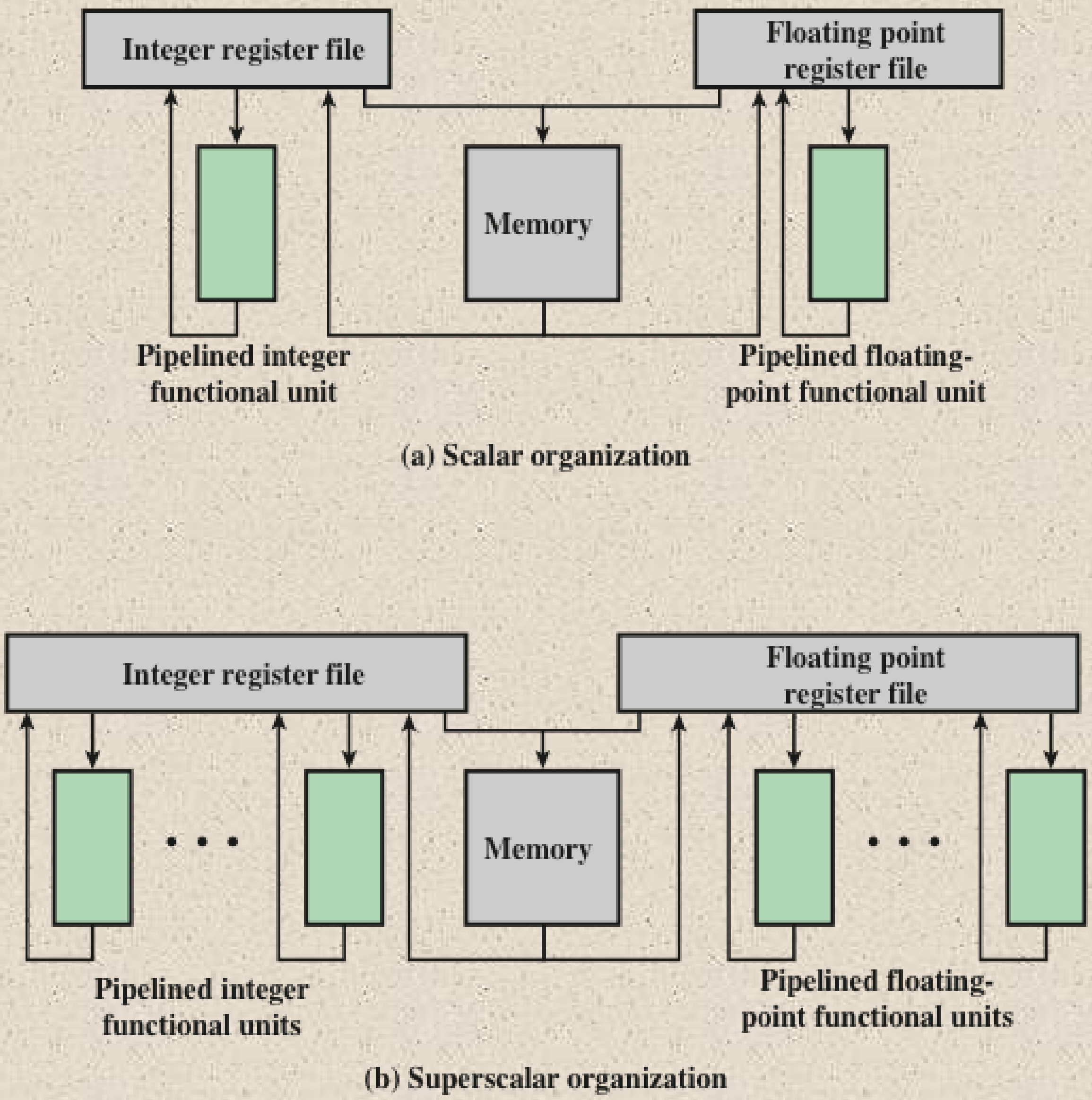

Integer register file
Floating point register file
Memory
Pipelined integer functional unit
Pipelined floating-point functional unit
(a) Scalar organization
Integer register file
Floating point register file
Memory
Pipelined integer functional units
Pipelined floating-point functional units
(b) Superscalar organization

Figure 16.1 Superscalar Organization Compared to Ordinary Scalar Organization

Comparison of Superscalar and Superpipeline Approaches

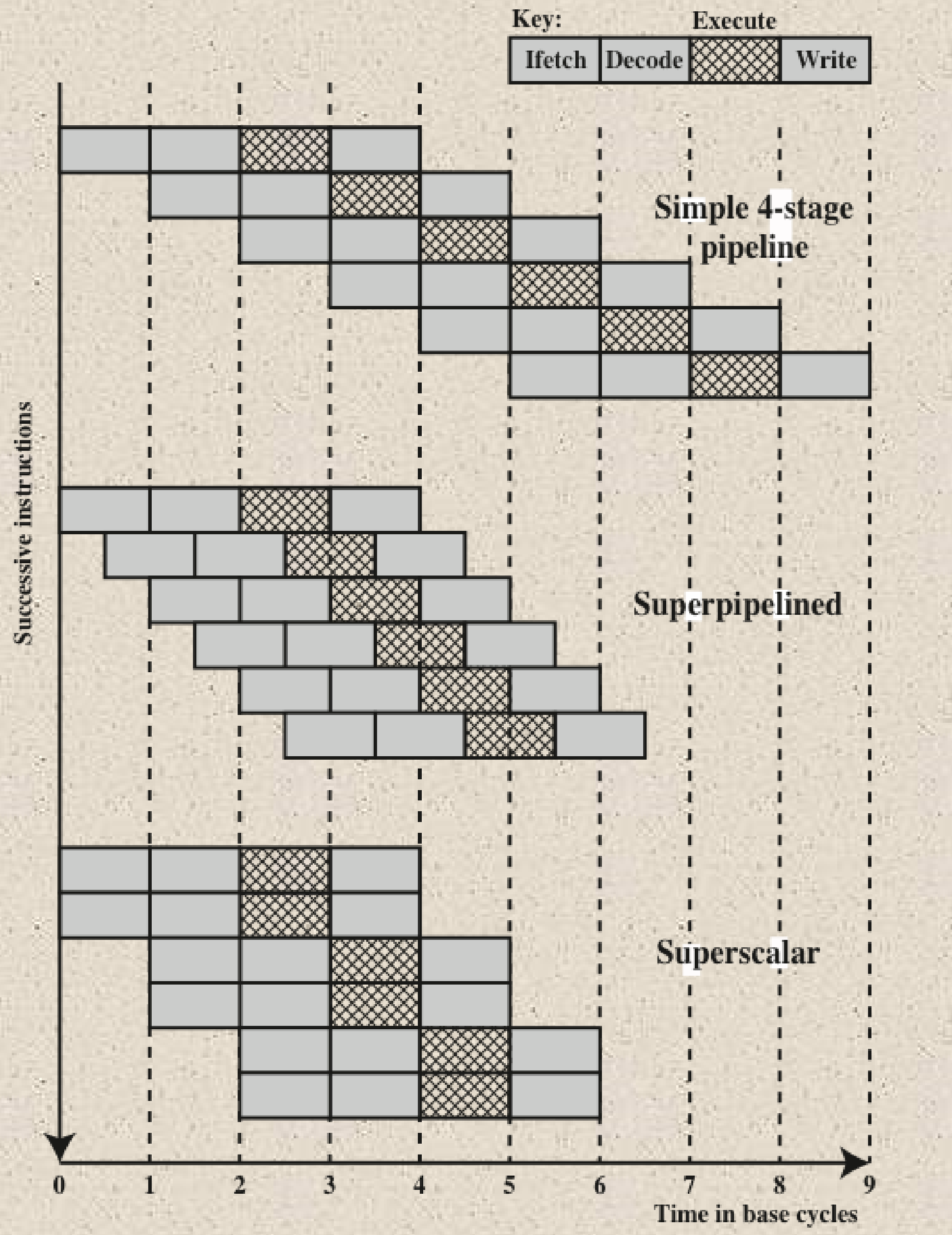


Figure 16.2 Comparison of Superscalar and Superpipeline Approaches

+
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

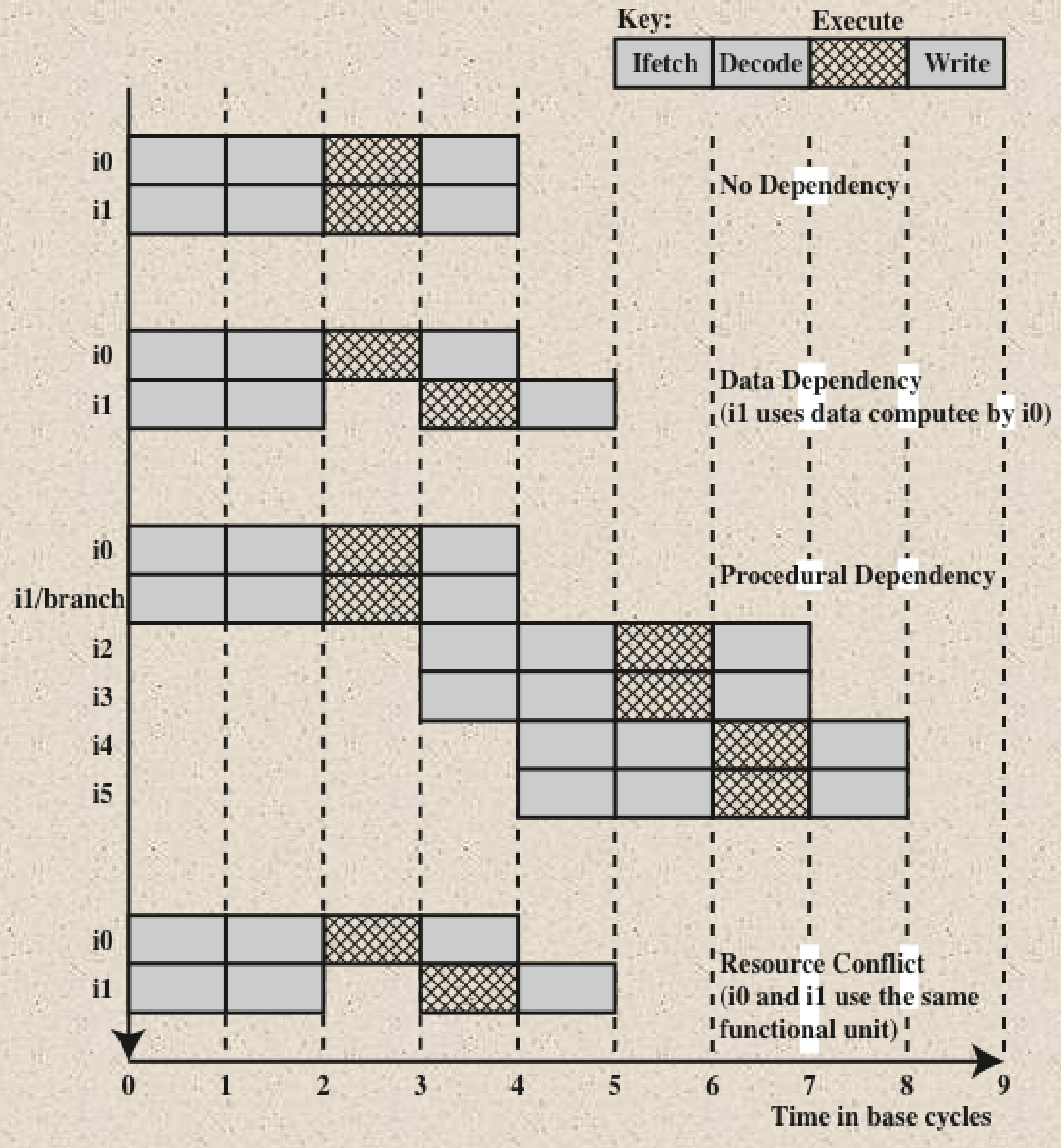


Figure 16.3 Effect of Dependencies

+

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

- Instruction issue
  - Refers to the process of initiating instruction execution in the processor's functional units
- Instruction issue policy
  - Refers to the protocol used to issue instructions
  - Instruction issue occurs when instruction moves from the decode stage of the pipeline to the first execute stage of the pipeline
- Three types of orderings are important:
  - The order in which instructions are fetched
  - The order in which instructions are executed
  - The order in which instructions update the contents of register and memory locations
- Superscalar instruction issue policies can be grouped into the following categories:
  - In-order issue with in-order completion
  - In-order issue with out-of-order completion
  - Out-of-order issue with out-of-order completion

+

Superscalar Instruction Issue and Completion Policies

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

Figure 16.4 Superscalar Instruction Issue and Completion Policies

Organization for Out-of-Order Issue with Out-of-Order Completion

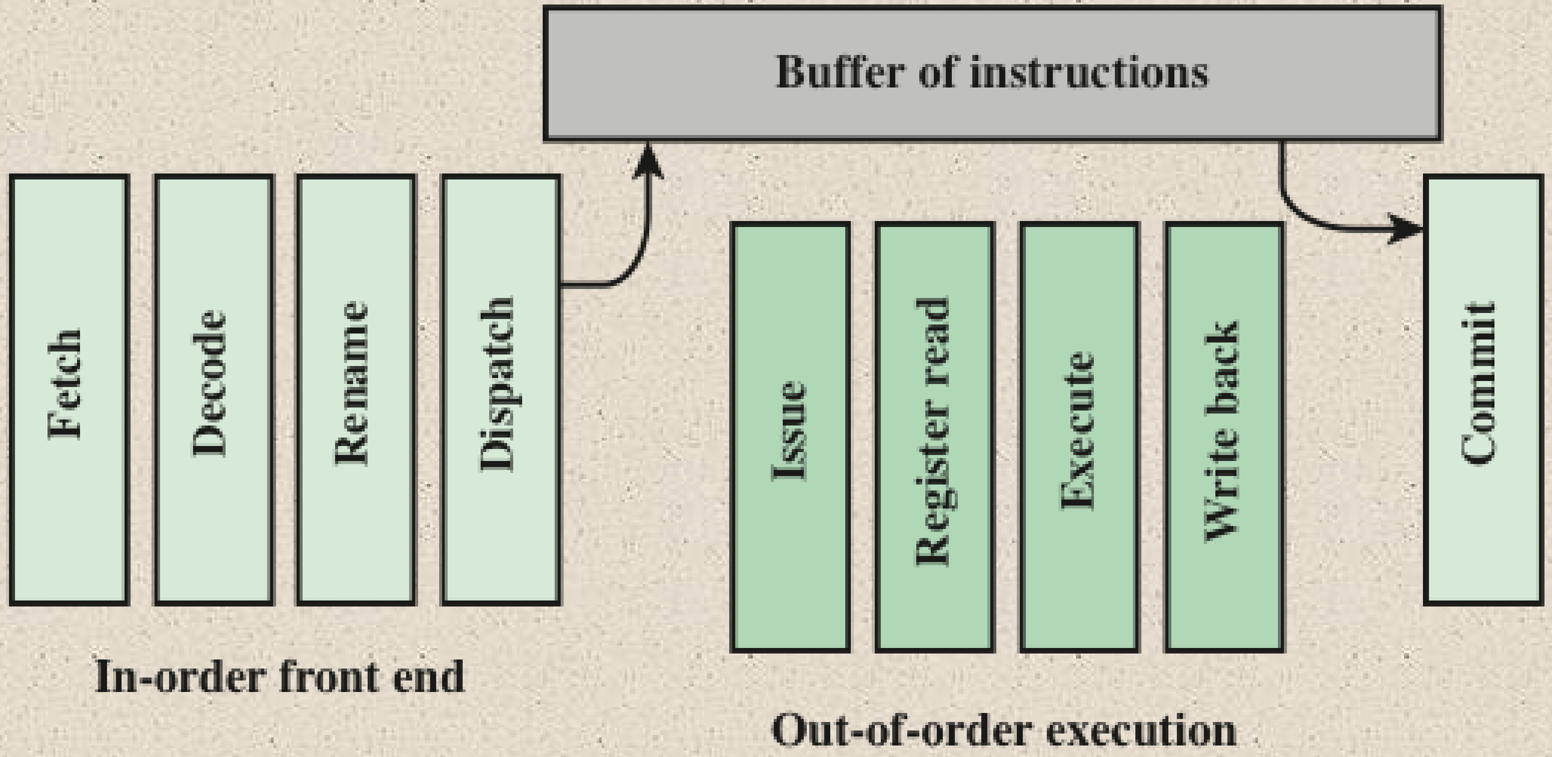


Figure 16.5 Organization for Out-of-Order Issue with Out-of-Order Completion

Register Renaming

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

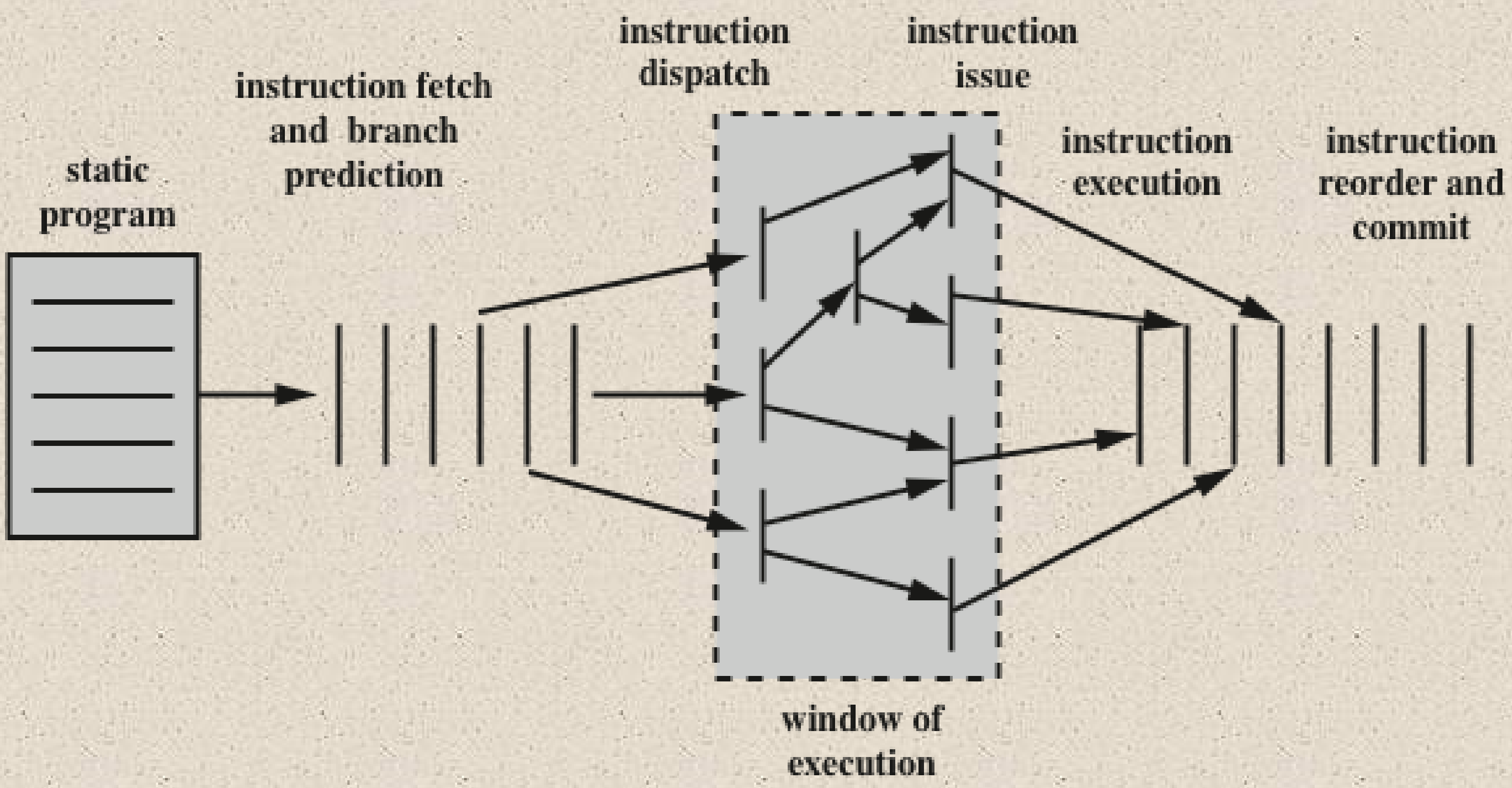
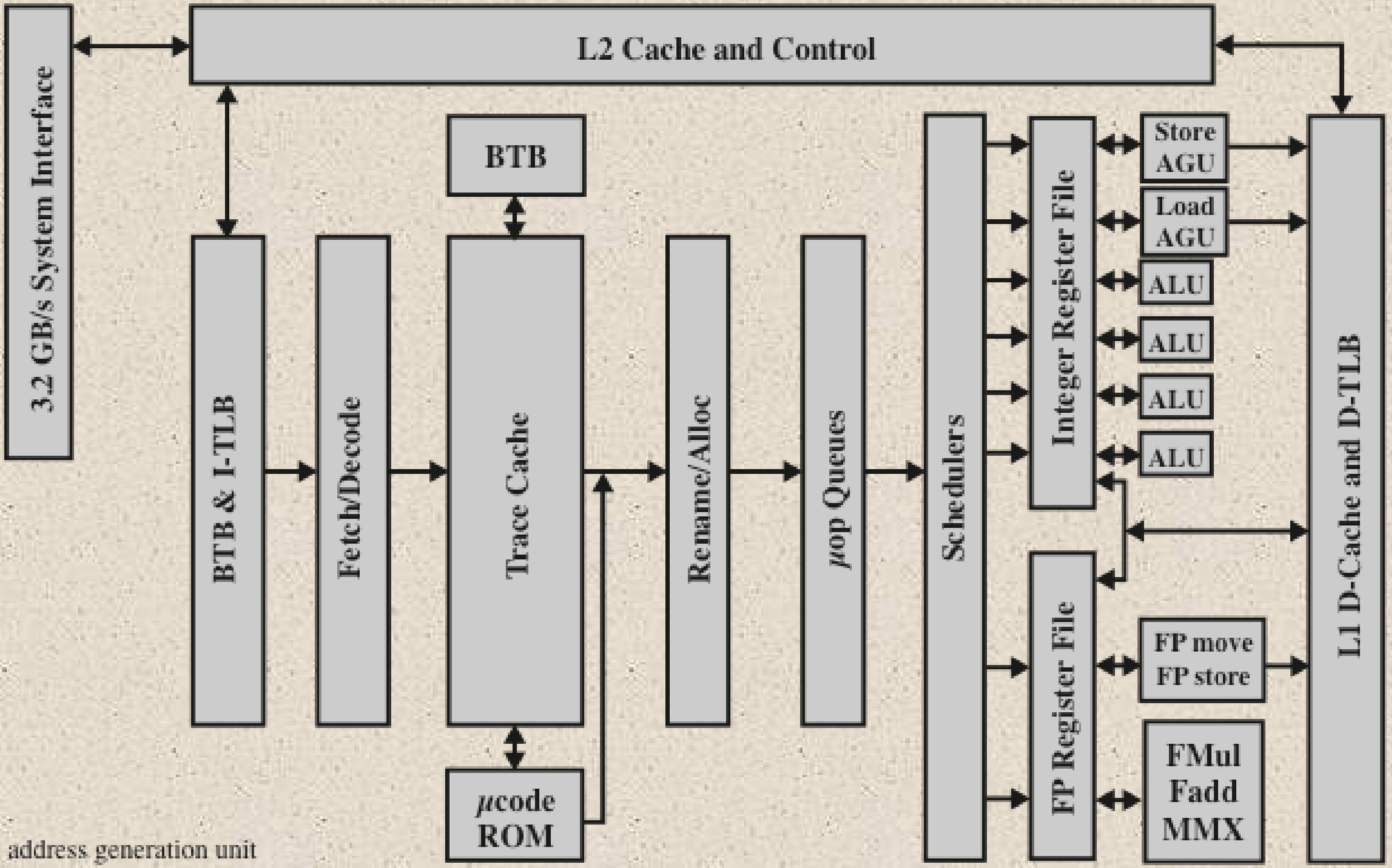


Figure 16.7 Conceptual Depiction of Superscalar Processing

+
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

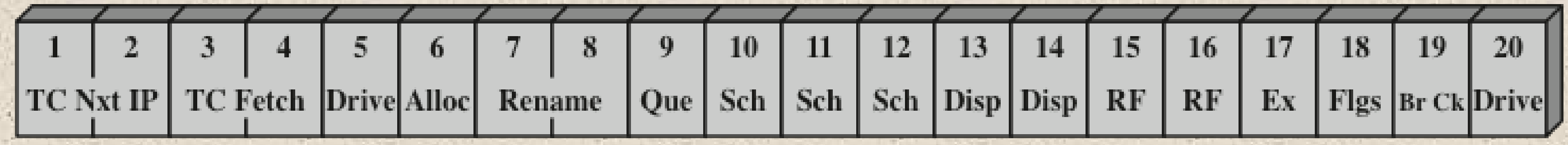
Pentium 4 Block Diagram



AGU	= address generation unit
BTB	= branch target buffer
D-TLB	= data translation lookaside buffer
I-TLB	= instruction translation lookaside buffer

Figure 16.8 Pentium 4 Block Diagram

Pentium 4 Pipeline



TC Next IP = trace cache next instruction pointer	Rename = register renaming	RF = register file
TC Fetch = trace cache fetch	Que = micro-op queuing	Ex = execute
Alloc = allocate	Sch = micro-op scheduling	Flgs = flags
	Disp = Dispatch	Br Ck = branch check

Figure 16.9 Pentium 4 Pipeline

+

Pentium 4 Pipeline Operation

Page 1 of 2

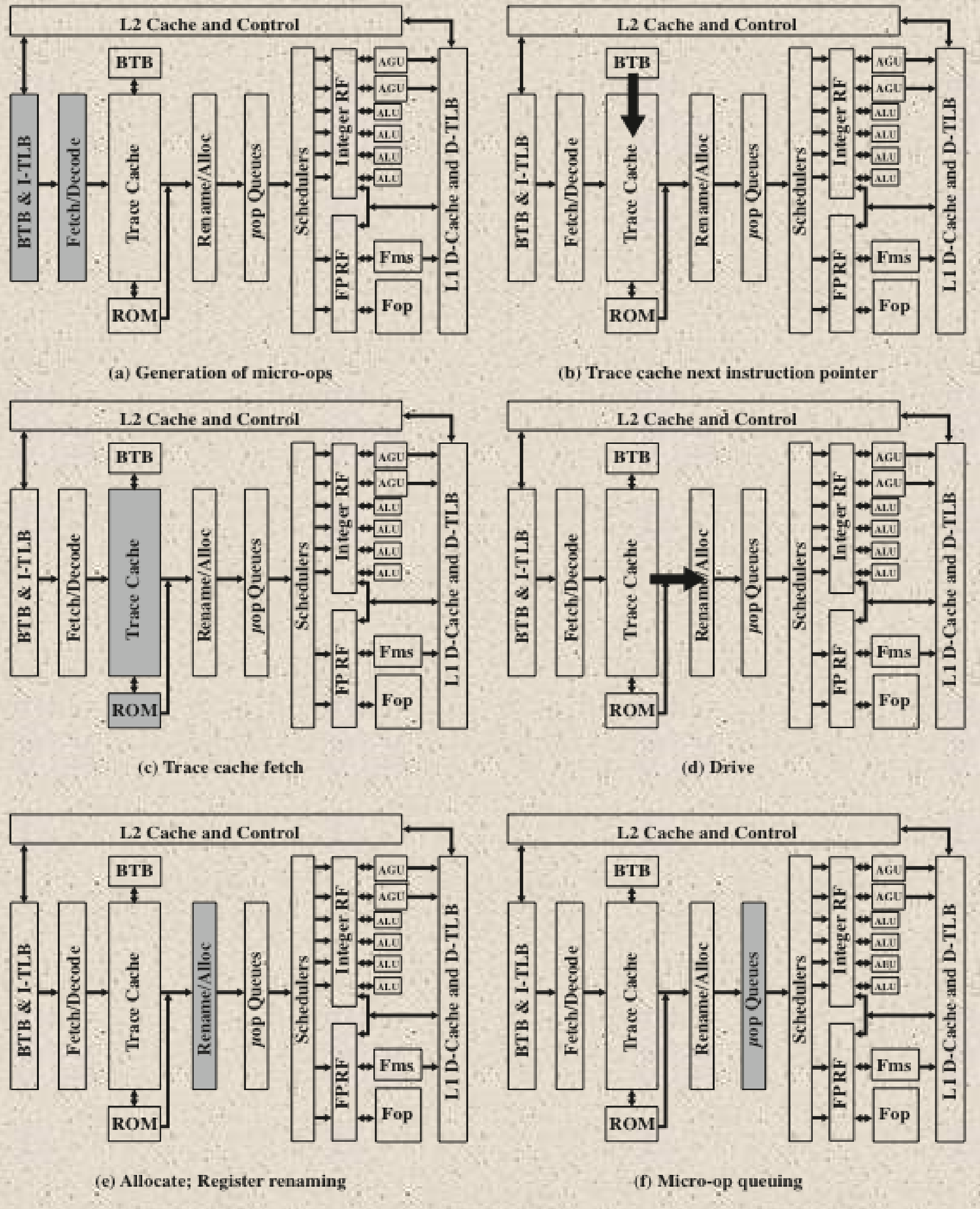


Figure 16.10 Pentium Pipeline Operation (page 1 of 2)

+

Pentium 4 Pipeline Operation

Page 2 of 2

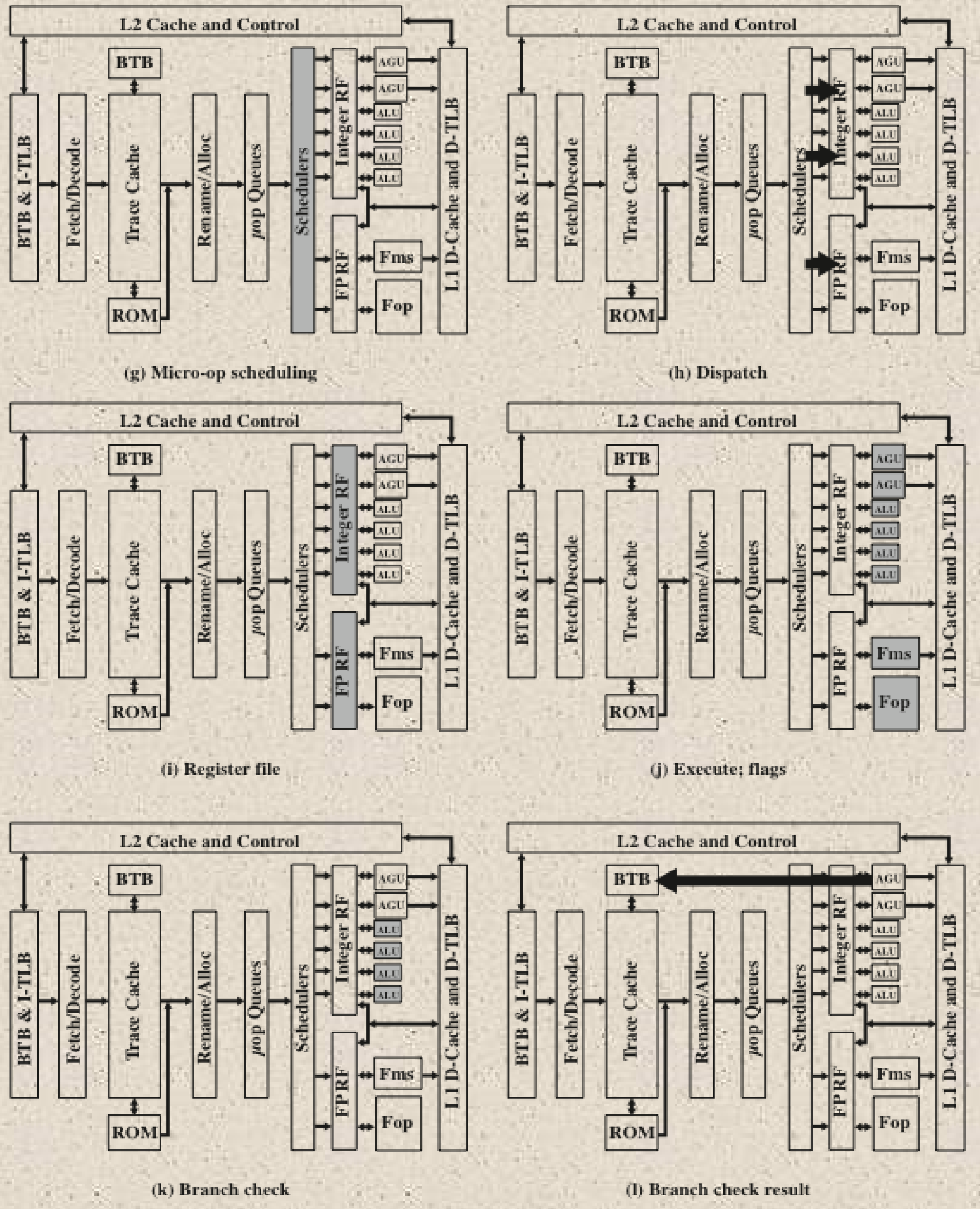


Figure 16.10 Pentium Pipeline Operation (page 2 of 2)

ARM CORTEX-A8

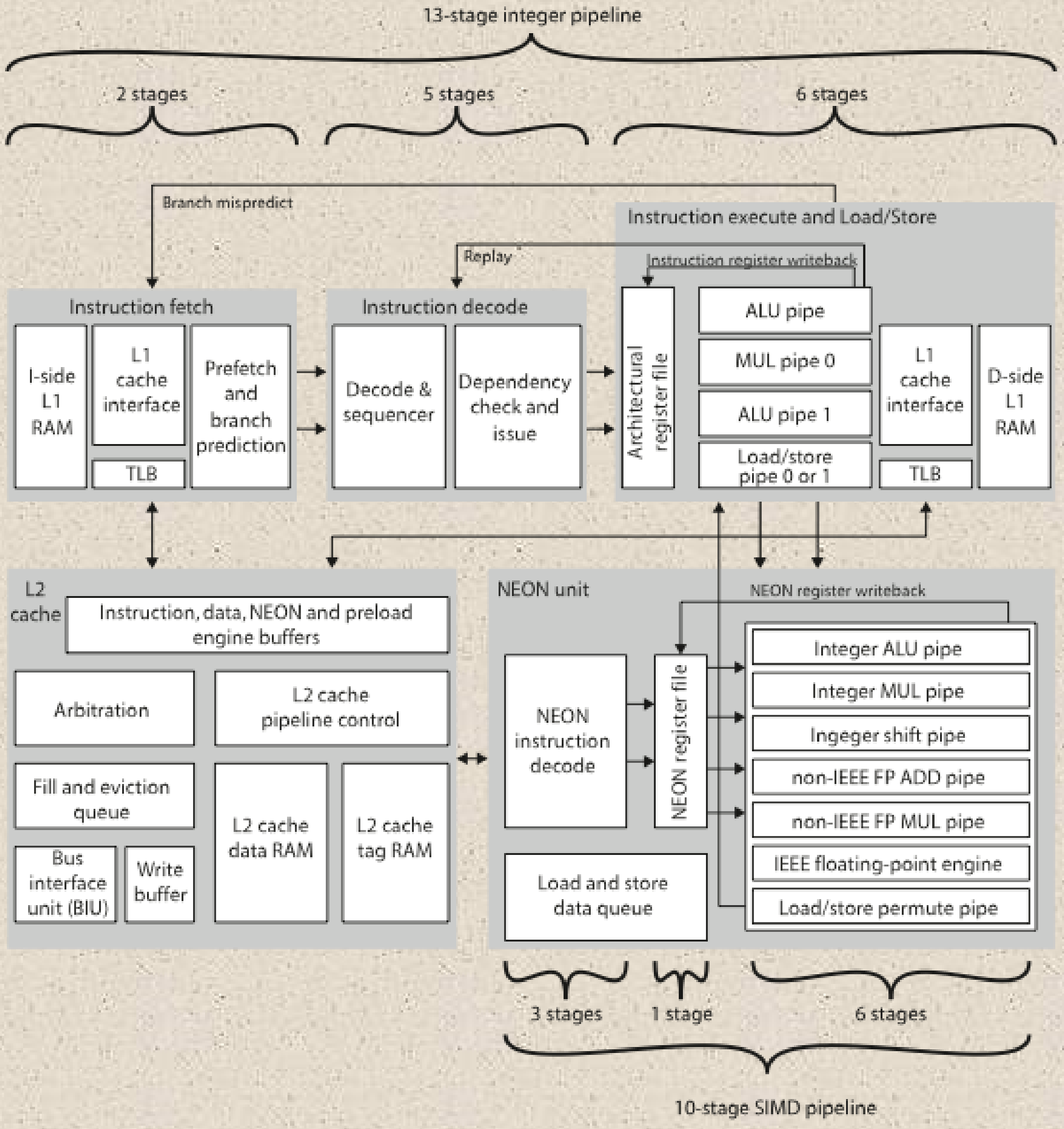


Figure 16.11 Architectural Block Diagram of ARM Cortex-A8

ARM Cortex-A8 Integer Pipeline

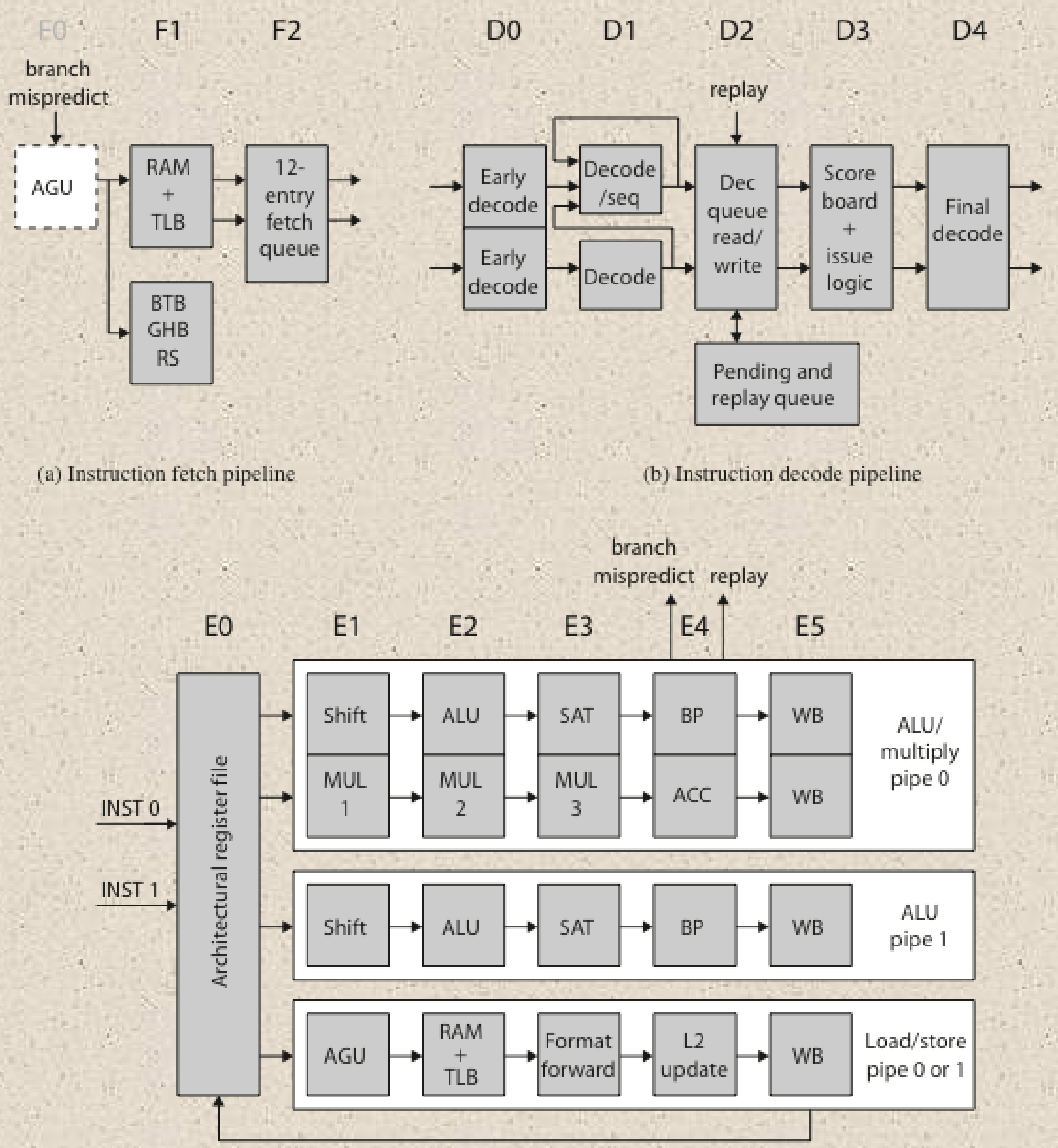


Figure 16.12 ARM Cortex-A8 Integer Pipeline

+
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

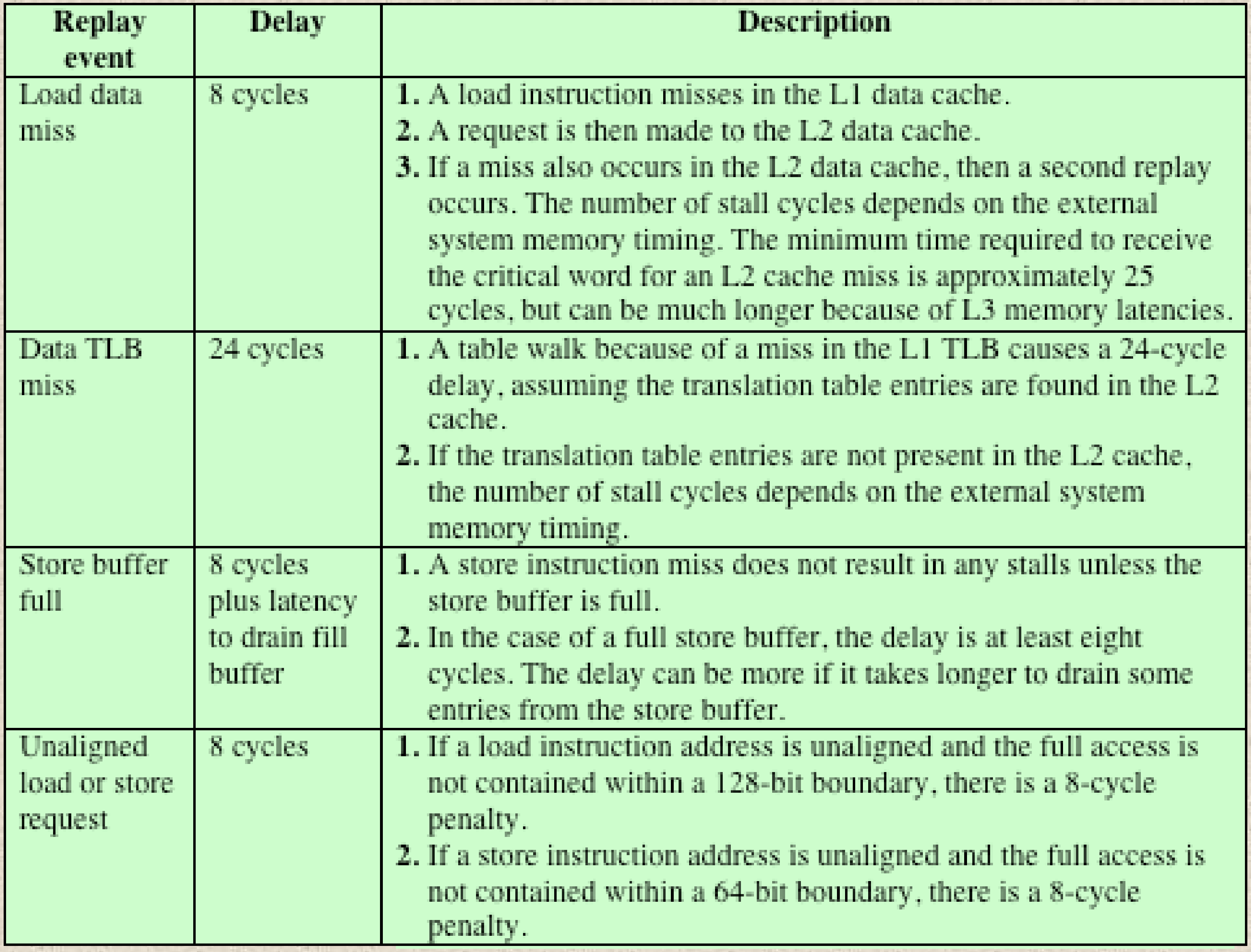
Instruction Decode Unit

- 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

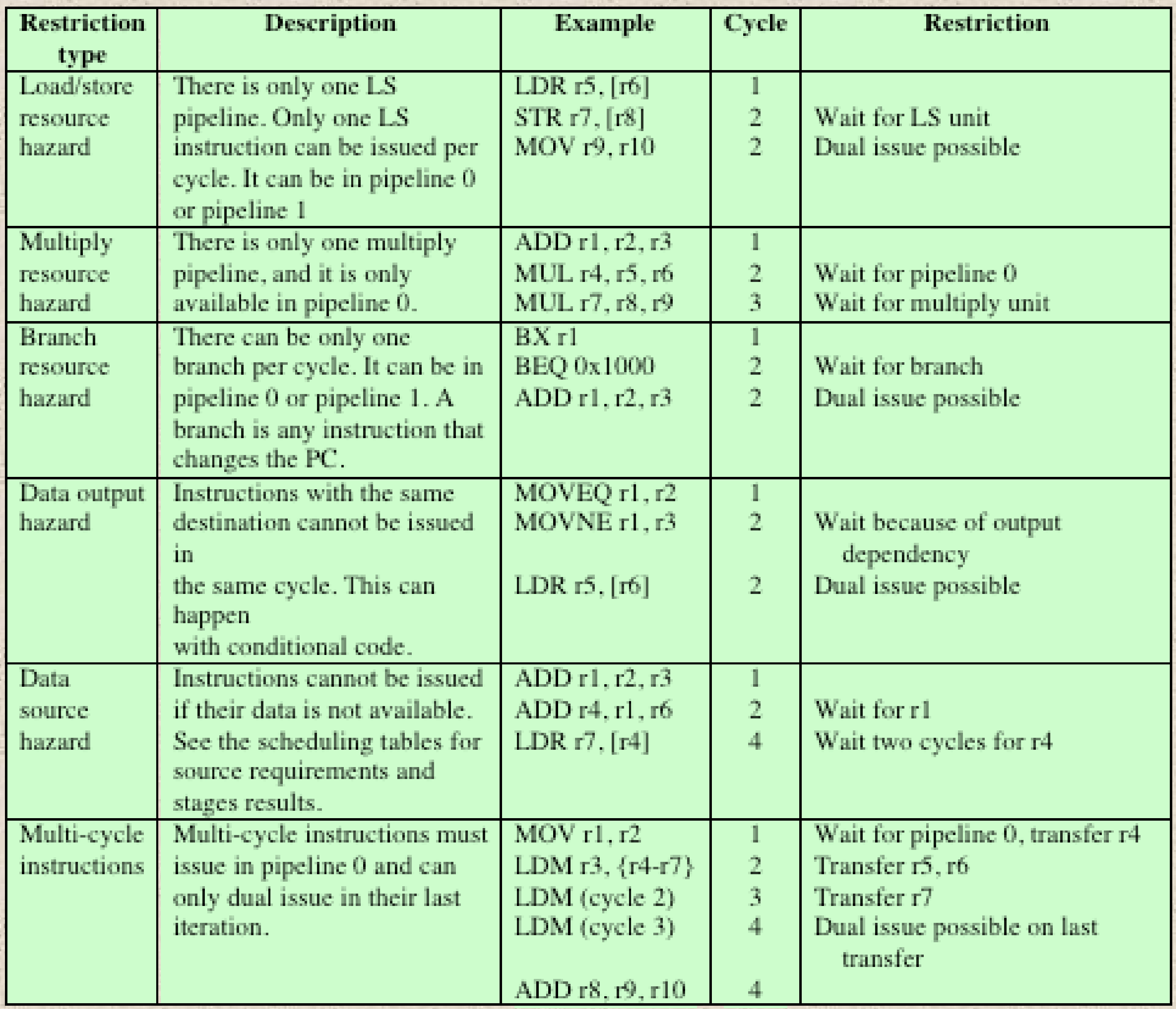
- D0
  - Thumb instructions decompressed and preliminary decode is performed
- D1
  - Instruction decode is completed
- D2
  - Writes instructions into and read instructions from pending/replay queue
- D3
  - Contains the instruction scheduling logic
  - Scoreboard predicts register availability using static scheduling
  - Hazard checking is done
- D4
  - Final decode for control signals for integer execute load/store units

Cortex-A8 Memory System Effects on Instruction Timings



Replay event	Delay	Description
Load data miss	8 cycles	**1.** A load instruction misses in the L1 data cache. **2.** A request is then made to the L2 data cache. **3.** If a miss also occurs in the L2 data cache, then a second replay occurs. The number of stall cycles depends on the external system memory timing. The minimum time required to receive the critical word for an L2 cache miss is approximately 25 cycles, but can be much longer because of L3 memory latencies.
Data TLB miss	24 cycles	**1.** A table walk because of a miss in the L1 TLB causes a 24-cycle delay, assuming the translation table entries are found in the L2 cache. **2.** If the translation table entries are not present in the L2 cache, the number of stall cycles depends on the external system memory timing.
Store buffer full	8 cycles plus latency to drain fill buffer	**1.** A store instruction miss does not result in any stalls unless the store buffer is full. **2.** In the case of a full store buffer, the delay is at least eight cycles. The delay can be more if it takes longer to drain some entries from the store buffer.
Unaligned load or store request	8 cycles	**1.** If a load instruction address is unaligned and the full access is not contained within a 128-bit boundary, there is a 8-cycle penalty. **2.** If a store instruction address is unaligned and the full access is not contained within a 64-bit boundary, there is a 8-cycle penalty.

Cortex-A8 Dual-Issue Restrictions



Restriction type	Description	Example	Cycle	Restriction
Load/store resource hazard	There is only one LS pipeline. Only one LS instruction can be issued per cycle. It can be in pipeline 0 or pipeline 1	LDR r5, [r6] STR r7, [r8] MOV r9, r10	1 2 2	 Wait for LS unit Dual issue possible
Multiply resource hazard	There is only one multiply pipeline, and it is only available in pipeline 0.	ADD r1, r2, r3 MUL r4, r5, r6 MUL r7, r8, r9	1 2 3	 Wait for pipeline 0 Wait for multiply unit
Branch resource hazard	There can be only one branch per cycle. It can be in pipeline 0 or pipeline 1. A branch is any instruction that changes the PC.	BX r1 BEQ 0x1000 ADD r1, r2, r3	1 2 2	 Wait for branch Dual issue possible
Data output hazard	Instructions with the same destination cannot be issued in the same cycle. This can happen with conditional code.	MOVEQ r1, r2 MOVNE r1, r3 LDR r5, [r6]	1 2 2	 Wait because of output dependency Dual issue possible
Data source hazard	Instructions cannot be issued if their data is not available. See the scheduling tables for source requirements and stages results.	ADD r1, r2, r3 ADD r4, r1, r6 LDR r7, [r4]	1 2 4	 Wait for r1 Wait two cycles for r4
Multi-cycle instructions	Multi-cycle instructions must issue in pipeline 0 and can only dual issue in their last iteration.	MOV r1, r2 LDM r3, {r4-r7} LDM (cycle 2) LDM (cycle 3) ADD r8, r9, r10	1 2 3 4 4	Wait for pipeline 0, transfer r4 Transfer r5, r6 Transfer r7 Dual issue possible on last transfer

+

Integer Execute Unit

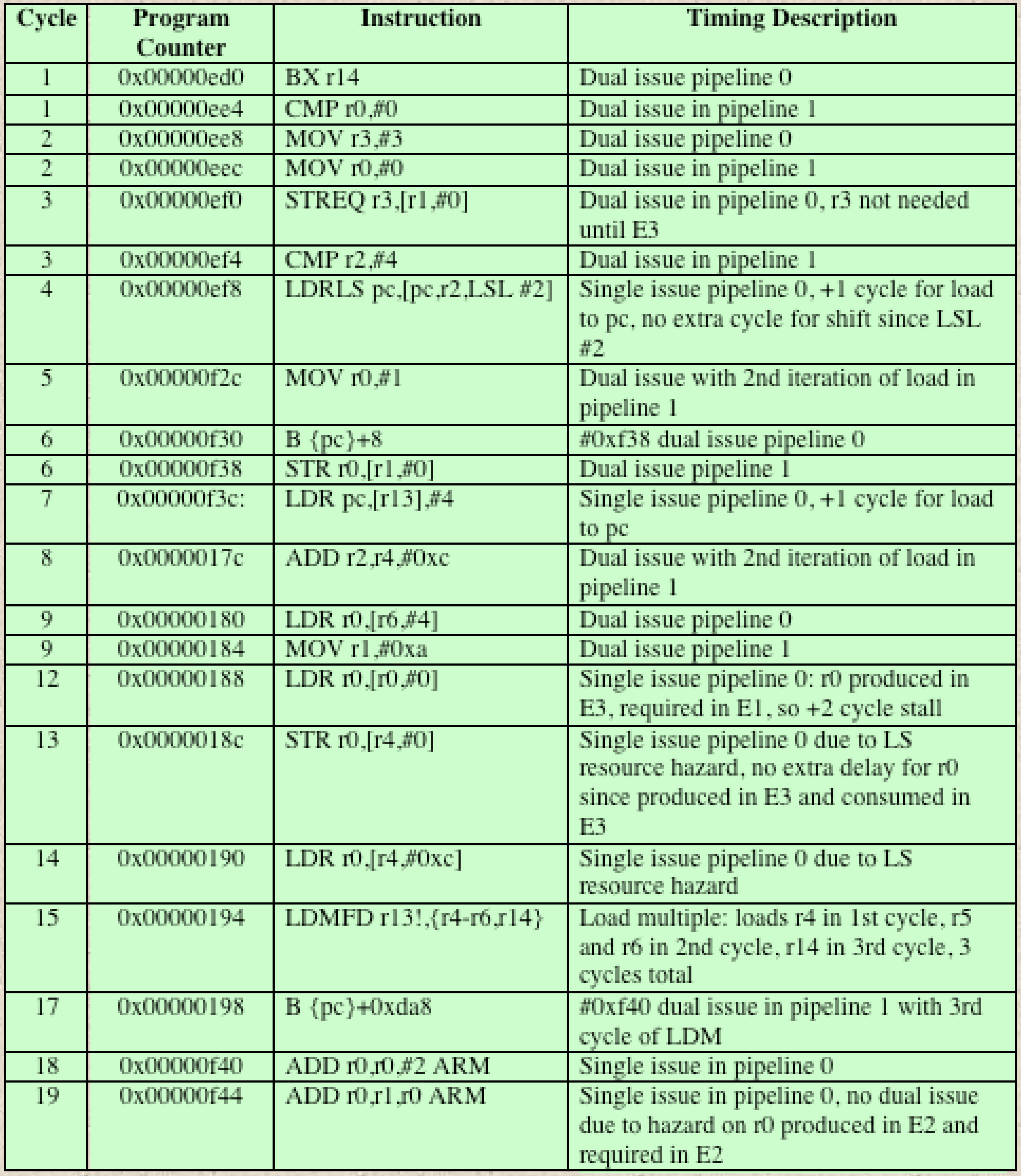
- Consists of:
  - Two symmetric arithmetic logic unit (ALU) pipelines
  - An address generator for load and store instructions
  - The multiply pipeline
- The instruction execute unit:
  - Executes all integer ALU and multiply operations, including flag generation
  - Generates the virtual addresses 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

Cortex-A8 Example Dual Issue Instruction Sequence for Integer Pipeline



Cycle	Program Counter	Instruction	Timing Description
1	0x00000ed0	BX r14	Dual issue pipeline 0
1	0x00000ee4	CMP r0,#0	Dual issue in pipeline 1
2	0x00000ee8	MOV r3,#3	Dual issue pipeline 0
2	0x00000eec	MOV r0,#0	Dual issue in pipeline 1
3	0x00000ef0	STREQ r3,[r1,#0]	Dual issue in pipeline 0, r3 not needed until E3
3	0x00000ef4	CMP r2,#4	Dual issue in pipeline 1
4	0x00000ef8	LDRLS pc,[pc,r2,LSL #2]	Single issue pipeline 0, +1 cycle for load to pc, no extra cycle for shift since LSL #2
5	0x00000f2c	MOV r0,#1	Dual issue with 2nd iteration of load in pipeline 1
6	0x00000f30	B {pc}+8	#0xf38 dual issue pipeline 0
6	0x00000f38	STR r0,[r1,#0]	Dual issue pipeline 1
7	0x00000f3c:	LDR pc,[r13],#4	Single issue pipeline 0, +1 cycle for load to pc
8	0x0000017c	ADD r2,r4,#0xc	Dual issue with 2nd iteration of load in pipeline 1
9	0x00000180	LDR r0,[r6,#4]	Dual issue pipeline 0
9	0x00000184	MOV r1,#0xa	Dual issue pipeline 1
12	0x00000188	LDR r0,[r0,#0]	Single issue pipeline 0: r0 produced in E3, required in E1, so +2 cycle stall
13	0x0000018c	STR r0,[r4,#0]	Single issue pipeline 0 due to LS resource hazard, no extra delay for r0 since produced in E3 and consumed in E3
14	0x00000190	LDR r0,[r4,#0xc]	Single issue pipeline 0 due to LS resource hazard
15	0x00000194	LDMFD r13!,{r4-r6,r14}	Load multiple: loads r4 in 1st cycle, r5 and r6 in 2nd cycle, r14 in 3rd cycle, 3 cycles total
17	0x00000198	B {pc}+0xda8	#0xf40 dual issue in pipeline 1 with 3rd cycle of LDM
18	0x00000f40	ADD r0,r0,#2 ARM	Single issue in pipeline 0
19	0x00000f44	ADD r0,r1,r0 ARM	Single issue in pipeline 0, no dual issue due to hazard on r0 produced in E2 and required in E2

ARM Cortex-A8 NEON & Floating-Point Pipeline

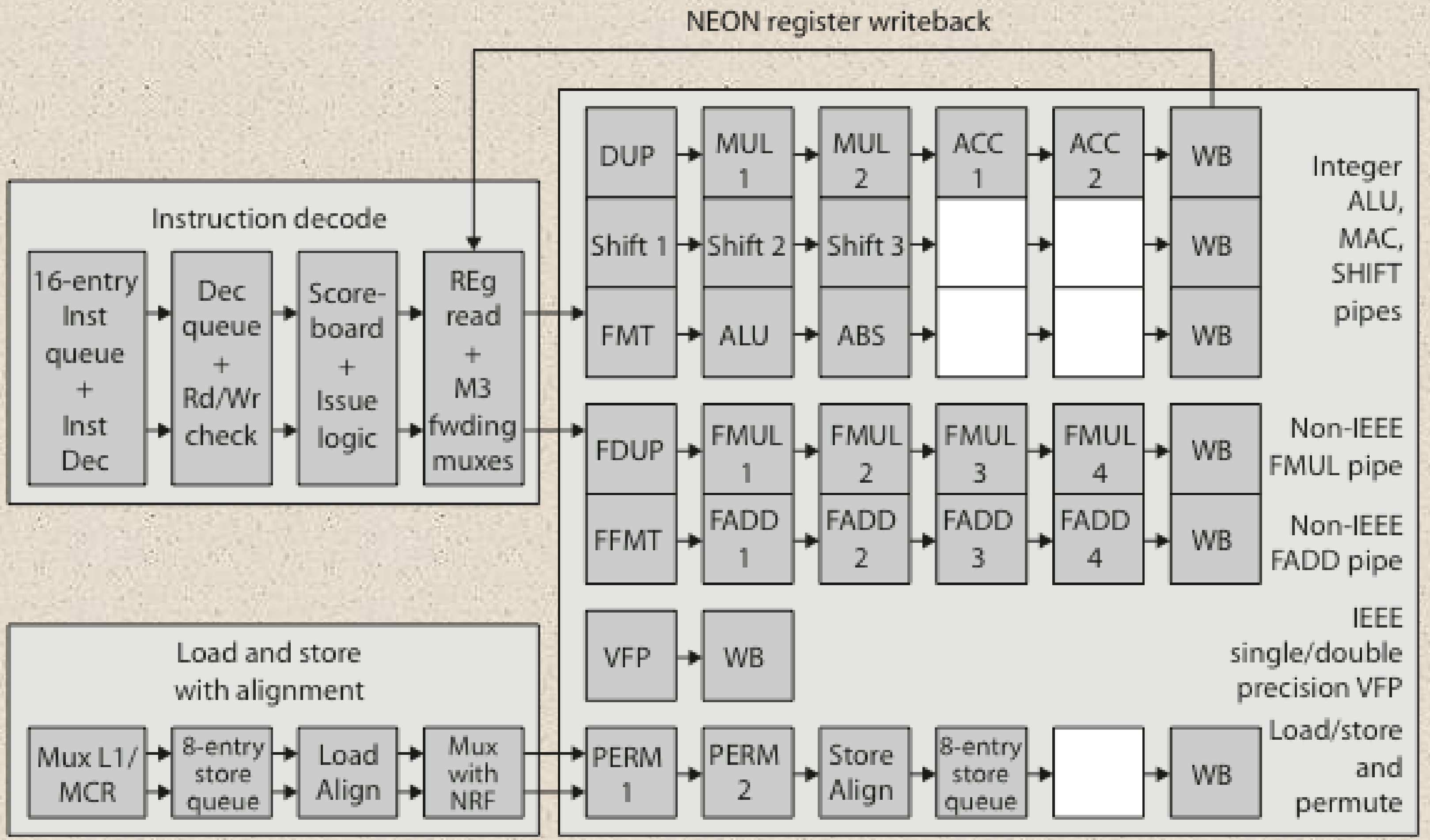


Figure 16.13 ARM Cortex-A8 NEON and Floating-Point Pipeline