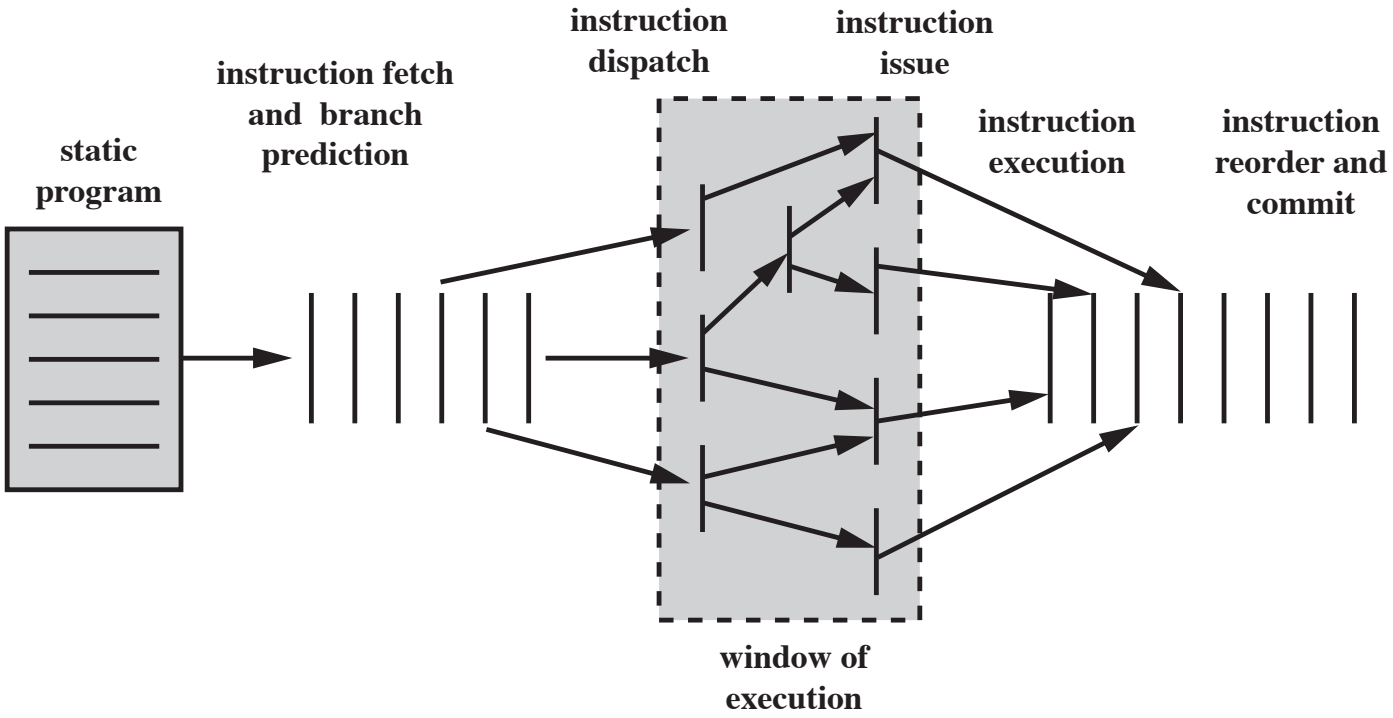
电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC4019: Real-Time Computer Systems and Architecture

Lecture 18
Instruction-Level Parallelism, and Superscalar Processors (Part-2)

Conceptual Depiction of Superscalar Processing



The program to be executed consists of a linear sequence of instructions. This is the static program as written by the programmer or generated by the compiler. The instruction fetch stage, which includes branch prediction, is used to form a dynamic stream of instructions. This stream is examined for dependencies, and the processor may remove artificial dependencies. The processor then dispatches the instructions into a window of execution. In this window, instructions no longer form a sequential stream but are structured according to their true data dependencies. The processor executes each instruction in an order determined by the true data dependencies and hardware resource availability. Finally, instructions are conceptually put back into sequential order and their results are recorded.

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

Superscalar Association

- Although the concept of superscalar design is generally associated with the RISC architecture, the same superscalar principles can be applied to a CISC machine.
- Perhaps the most notable example of this is the Intel x86 architecture. The 386 is a traditional CISC nonpipelined machine.
- The 486 introduced the first pipelined x86 processor, reducing the average latency of integer operations from between two and four cycles to one cycle, but still limited to executing a single instruction each cycle, with nosuperscalar elements.
- The original Pentium had a modest superscalar component, consisting of the use of two separate integer execution units.
- The Pentium Pro introduced a full-blown superscalar design with out-of-order execution. Subsequent x86 models have refined and enhanced the superscalar design.

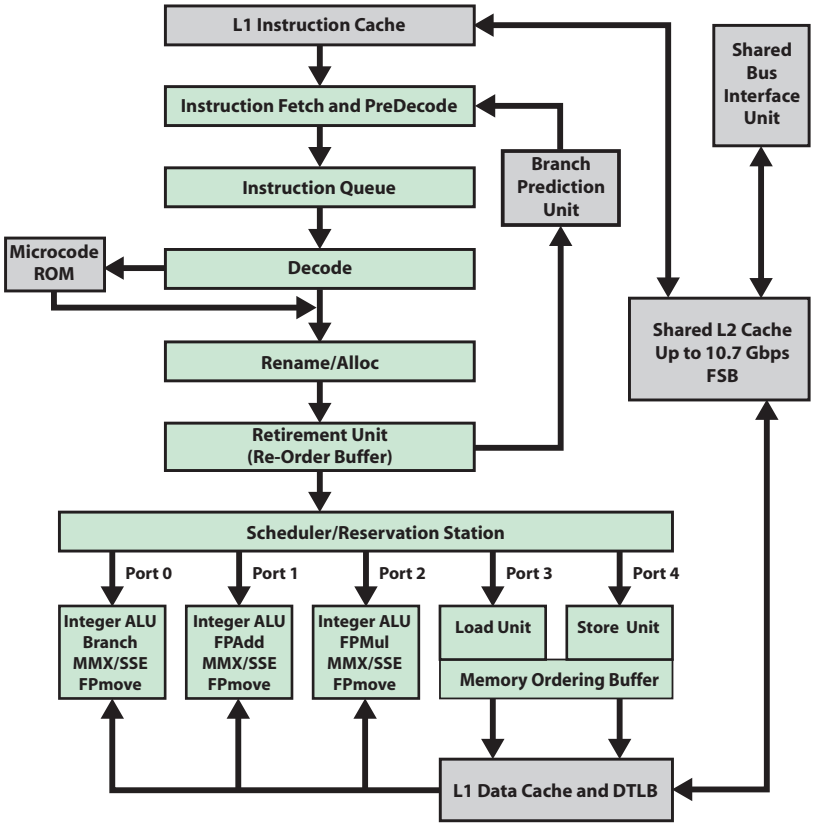
Intel Core Microarchitecture

Figure shows the current version of the x86 pipeline architecture.

Intel refers to a pipeline architecture as a microarchitecture . The microarchitecture underlies and implements the machine's instruction set architecture.

The microarchitecture is referred to as the Intel Core Microarchitecture. It is implemented on each processor core in the Intel Core 2 and Intel Xeon processor families.

There is also an Enhanced Intel Core Microarchitecture. One key difference between the two microarchitectures is that the Enhanced Intel Core Microarchitecture provides a third level of cache.



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 (1 of 2)

- 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

Instruction Fetch and Predecode Unit (2 of 2)

  - 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 into 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

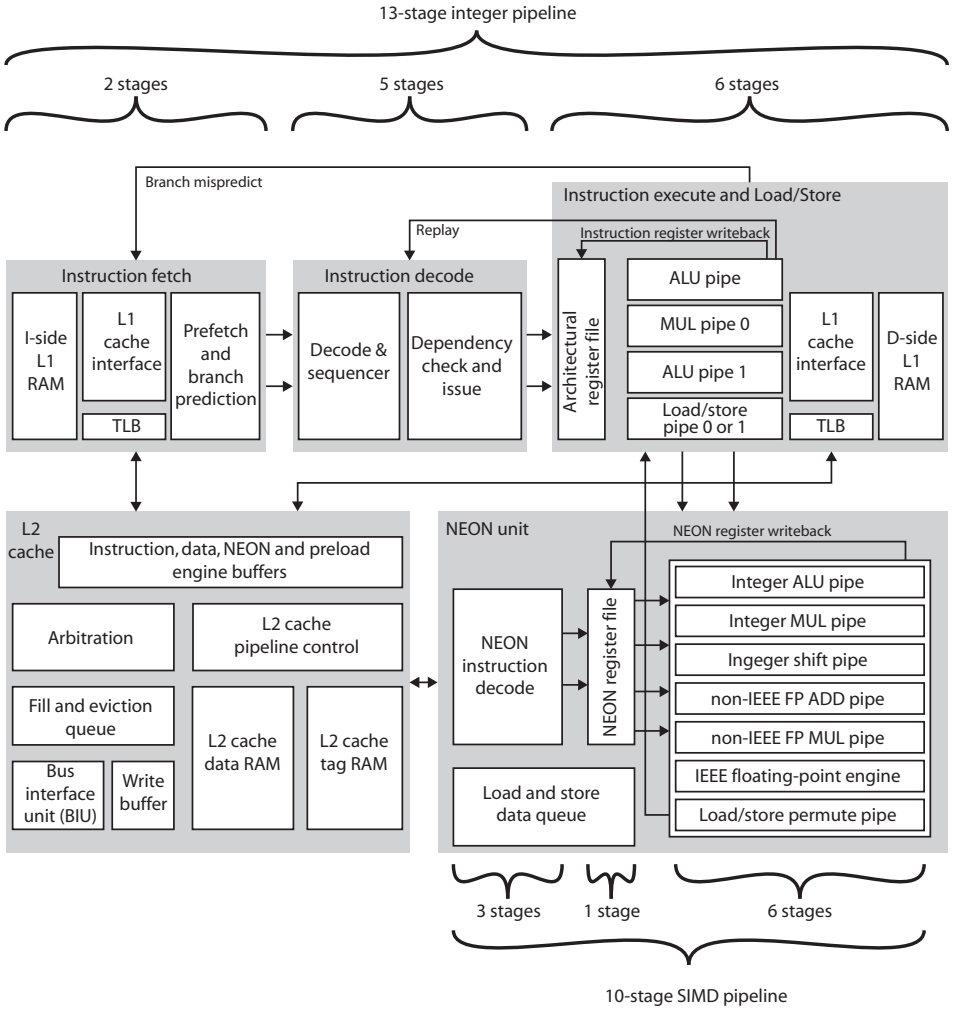
Architectural Block Diagram of ARM Cortex-A8

Figure shows a logical view of the Cortex-A8 architecture, emphasizing the flow of instructions among functional units

The main instruction flow is through three functional units that implement a dual, in-order-issue, 13-stage pipeline

The Cortex designers decided to stay with in-order issue to keep additional power required to a minimum

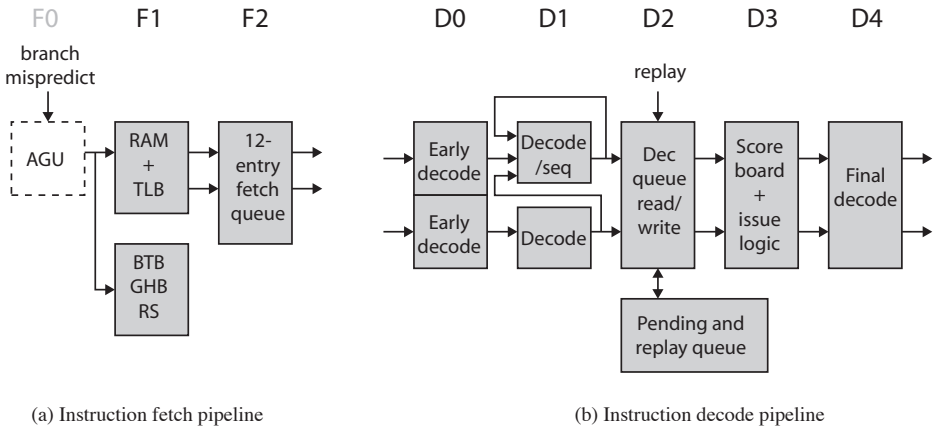
Out-of-order issue and retire can require extensive amounts of logic consuming extra power



ARM Cortex-A8 Integer Pipeline

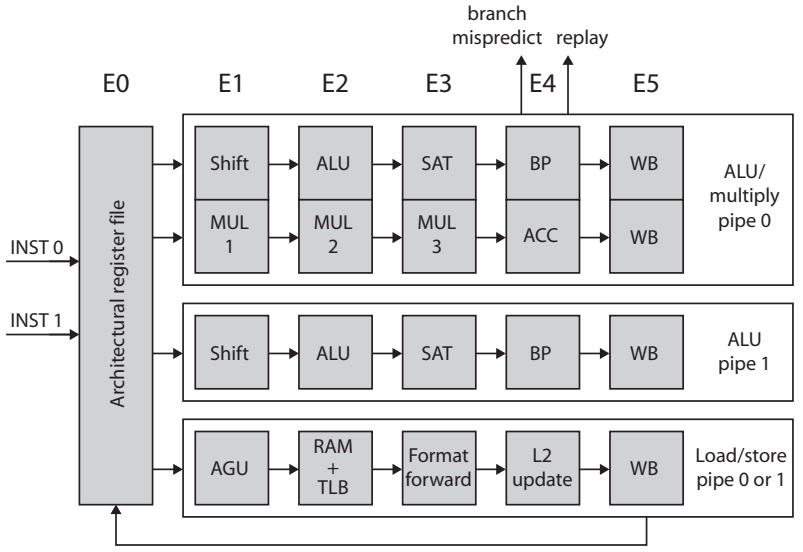
This Figure shows the details of the main Cortex-A8 pipeline

There is a separate unit for SIMD (single-instruction-multiple-data) unit that implements a 10-stage pipeline



(a) Instruction fetch pipeline

(b) Instruction decode pipeline



(c) Instruction execute and load/store pipeline

Instruction Fetch Unit (1 of 2)

- 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

Instruction Fetch Unit (2 of 2)

- 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 (1 of 2)

- 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

Instruction Decode Unit (2 of 2)

- 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 RA W hazards

Instruction Processing Stages (1 of 2)

- 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

Instruction Processing Stages (2 of 2)

- 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

ARM Cortex-M3 Block Diagram

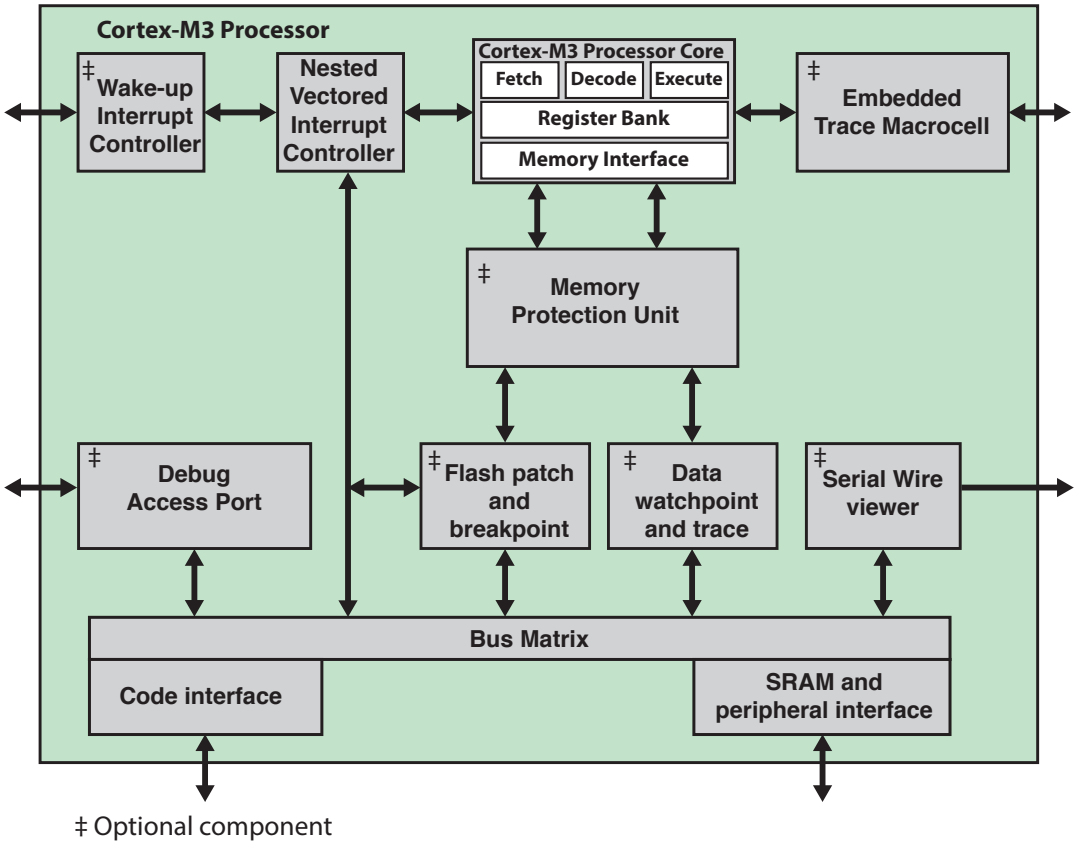


Figure above provides a block diagram overview of the Cortex-M3 processor. This figure provides more detail than that shown previously. Key elements included are given in the following slides.

ARM Cortex-M3 Key Elements (1 of 2)

- Processor core: Includes a three-stage pipeline, a register bank, and a memory interface
- Memory protection unit: Protects critical data used by the operating system from user applications, separating processing tasks by disallowing access to each other's data, disabling access to memory regions, allowing memory regions to be defined as read-only, and detecting unexpected memory accesses that could potentially break the system
- Nested vectored interrupt controller (NVIC): Provides configurable interrupt handling abilities to the processor. It facilitates low-latency exception and interrupt handling, and controls power management
- Wake-up interrupt controller (NVIC): Provides configurable interrupt handling abilities to the processor. It facilitates low-latency exception and interrupt handling, and controls power management

ARM Cortex-M3 Key Elements (2 of 2)

- Flash patch and breakpoint unit: Implements breakpoints and code patches
- Data watchpoint and trace (DWT): Implements watchpoints, data tracing, and system profiling
- Serial wire viewer: Can export a stream of software-generated messages, data trace, and profiling information through a single pin
- Debug access port: Provides an interface for external debug access to the processor
- Embedded trace macrocell: Is an application-driven trace source that supports printf() style debugging to trace operating system and application events, and generates diagnostic system information
- Bus matrix: Connects the core and debug interfaces to external buses on the microcontroller

ARM Cortex-M3 Pipeline

To keep the processor as simple as possible, the Cortex-M3 processor does not use branch prediction, but instead use the simple techniques of branch forwarding and branch speculation