A Primer on Processor Architecture

Zhiyang Ong

ongz@acm.org
Department of Electrical and Computer Engineering
Dwight Look College of Engineering
Texas A&M University

December 5, 2014

Abstract

This report provides an introduction into processor architecture (or microarchitecture) design. Four common microarchitecture designs are covered in this report are: single-cycle processors, multi-cycle processors, pipelined processors, and superscalar processors. A description of how to design the datapath and control path of these processor architectures is provided. In addition, a performance comparison of the aforementioned microarchitectures is provided.

Contents

1 Introduction to Processor Architecture

Current and emerging trends in "Big Data" [5, 68, 115], Internet of Things [117], and cyber-physical systems [90] drive a need for energy-efficient computing. From mobile devices to high-performance super computing, the power consumption and cost of cooling these computers have become so dominant that they have become a barrier to improving the performance of computers [35, 44]. Without better computers, it can be harder for companies to sell computers to consumers and enterprises.

Furthermore, computer architecture is a difficult subject to teach, especially at the undergraduate level [122]. Due to the demand for high-performance, energy-efficient processors [30], we need to train more highly-skilled computer engineering students to pursue careers in the field of computer architecture. In addition, before students and professionals can tackle grand challenges in computer architecture [111], they need to master the basics of modern processor design.

To quantify the improvements made to a processor's/computer's performance, we need to determine how much a processor's/computer's performance has improved compared to the best processor/computer for a generic set of software applications. We can also do likewise for a specific category of usage/applications. In order to reduce experimental bias, a standard benchmark suite for a specific category of computers (e.g., embedded processors, general-purpose multi-core processors, or graphics processors) shall be used to evaluate the performance and power consumption of computers [13]. For example, for evaluating the performance of single-threaded, single-core processors (or uniprocessors), the benchmark suite from the *Standard Performance Evaluation Corporation (SPEC)* [107] can be used. For multi-threaded, chip multiprocessors (CMP) [79], benchmark suites from the *Princeton Application Repository for Shared-Memory Computers (PARSEC)* [15, 82] and *SPLASH-2* [113] shall be used.

When the performance of a computer's component is improved, the resultant performance speedup for that computer can be evaluated with Amdahl's law [44] (see Equation 1); Amdahl's law can be extended for improvements made to a diverse a set of components [100].

$$\text{Performance speedup} = \frac{1}{(1 - \text{proportion of speedup}) + \frac{\text{proportion of speedup}}{\text{speedup}}} \tag{1}$$

That is, when the performance of a component is improved by x and this component makes up $y\%$ of the computer system's resources for computation, the overall performance speedup for this computer system is given by: $\frac{1}{(100\% - y\%) + \frac{y\%}{x}}$. In general, performance speedup is considered to be the ratio of the execution times or the performance of the two computers (X and Y); see Equation 2 [44].

$$\text{Performance speedup}, n = \frac{\text{performance of } X}{\text{performance of } Y} = \frac{\text{execution time of } Y}{\text{execution time of } X} \tag{2}$$

From Equation 2, the performance speedup gained by computer X over computer Y is n. That is, X is n times faster than Y.

The "iron law" [100] to determine the execution time of a processor is given as follows in Equation 3 [44].

$$\text{Execution time} = IC \times CPI \times CCT, \tag{3}$$

where IC refers to instruction count (or number of instructions in the benchmark program), CPI refers to cycles per instruction (i.e., average number of clock cycles to execute an instruction), and CCT refers to clock cycle time (or "global" clock period of the processor). The execution time is inversely proportional to the computer's performance; see Equation 4 [85].

$$\text{performance} = \frac{1}{\text{execution time}} \tag{4}$$

To evaluate the performance of two computers X and Y using computer Z as a reference computer for benchmarking, determine the *SPECRatio*s of X and Y for a given benchmark bmk_i by normalizing the performance of X and that of Y with respect to Z. Hence, Equation 2 becomes the ratio of the *SPECRatio*s for X and Y [44]:

$$\text{Performance speedup}, n = \frac{SPECRatio_X}{SPECRatio_Y} = \frac{\left(\frac{\text{execution time of } Z}{\text{execution time of } X}\right)}{\left(\frac{\text{execution time of } Z}{\text{execution time of } Y}\right)} = \frac{\left(\frac{\text{performance of } X}{\text{performance of } Z}\right)}{\left(\frac{\text{performance of } Y}{\text{performance of } Z}\right)} = \frac{\text{performance of } X}{\text{performance of } Y} \tag{5}$$

Consequently, to determine the average performance speedup for X over Y, one needs to determine the geometric mean of the performance speedup of X over Y for each benchmark. For a given set of n benchmarks, this geometric mean is determined as follows [44]:

$$\text{Performance speedup}, n = \sqrt[n]{\prod_{i=1}^{n} \frac{SPECRatio_X}{SPECRatio_Y}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{performance of } X_i}{\text{performance of } Y_i}} \tag{6}$$

Here, "performance of X_i" and "performance of Y_i" refer to the performances of computers X and Y for the benchmark bmk_i, and i denotes the i^{th} benchmark in the set of n benchmarks.

Before proceeding to discuss various techniques for improving the performance of computers, we need to put things in perspective. Computer architecture defines the interface between hardware and software (i.e., the hardware/software interface) with the instruction set architecture (ISA), and a given microarchitecture is an implementation of the ISA. Thus, for a given ISA, many microarchitectures can implement that ISA [44, 100]. In turn, the microarchitecture is implemented as a digital integrated circuit (IC) or very-large-scale integrated (VLSI) system. VLSI system designs begin with developing hardware behavioral models at the register-transfer level (RTL), which require a lot of engineers and time to create; these behavioral RTL models also take a long time to simulate. To speed up the process of designing microarchitectures, processor simulators can be developed and modified to explore different microarchitecture designs (i.e., design space exploration). This is because processor simulators simulate faster than behavioral RTL models [7, 8, 58, 66, 71, 100, 112]. An example of a processor simulator for single-core processors is *SimpleScalar 3.0* [102], and an example of a processor simulator for multi-core processors is *gem5* [16, 37].

This report aims to help students and junior professionals acquire the basics of modern processor design for single-core processors. It will cover different techniques of designing processor architecture (or microarchitecture), and how each processor design can be evaluated for its performance; in this report, the terms processor architecture and microarchitecture will be used interchangeably. The techniques covered include single-cycle processors, multi-cycle processors, pipelined processors, and superscalar processors. However, the following topics are beyond the scope of this report: memory system design [48, 49, 104], such as cache design [9]; multi-core and many-core processor design

[24,29,46,57,73,79]; embedded processors [47,52,63,75,105,118] or low-power and energy-efficient processors [56,78]; data center computers [3,10] and high-performance computing [41,64,67,121]; application-specific processors [14, 21–23, 33, 38, 41, 43, 58, 59, 65]; and network-on-chips (i.e., on-chip interconnect networks) [19,20,25,27,34,36,50,51,72,74,76,77,83,89,101]. It assumes that the reader has significant experience in designing digital ICs or VLSI systems [6,54,92,116,119], is familiar with working with a *UNIX*-like operating system (e.g., *Linux*) [17,88,93], has mastered the basics of computer system organization [1,26,86,106,110], and can develop software [18,91,94,103]. Lastly, the ISA used in this report is the *MIPS* ISA, where *MIPS* refers to the architecture and assembly programming language for *Microprocessor without Interlocked Pipeline Stages (MIPS)*. I have chosen the *MIPS* ISA because of its simplicity; the *MIPS* ISA is a classic specification for a reduced instruction set computer (RISC) [44,86,109].

The rest of this report is organized as follows. In Section 2, I present the fundamental principles of designing a processor that executes each instruction in a single clock cycle. Following that in Section 3, I modify the architecture of the processor to enable instructions to execute in multiple clock cycles, so that the performance of the processor can be improved. Subsequently, in Section 4, I introduce the notion of pipelining and use pipelining to improve execution of computer programs by modifying the microarchitecture. Next, superscalar processors that can issue multiple instructions per clock cycle are briefly described in Section 5. Finally, in Section 6, conclusions are drawn from designing processors using the aforementioned techniques.

2 Single-Cycle Processor Design

A simple microarchitecture of the *MIPS* ISA is the single-cycle processor that executes all instructions in one clock cycle. In this section, the basic concepts and techniques for implementing a single-cycle processor are discussed. These basic concepts can be built upon to implement more complex microarchitectures, such as multi-cycle processors, pipelined processors, and superscalar processors. These concepts and techniques can also be used to design microarchitectures that implement other ISAs, including ISAs of complex instruction set computers (CISC) [85].

This section shall focus on implementing a representative subset of the *MIPS* ISA that spans all the categories of *MIPS* instructions; the remaining instructions of the *MIPS* ISA can be implemented using the aforementioned basic concepts for single-cycle processor design. The categories of *MIPS* instructions are [85]:

1. memory access instructions for read and write access; e.g., `load word (lw)` and `store word (sw)` instructions
2. arithmetic and logical operation instructions; e.g., `add`, `subtract (sub)`, (logical) `or`, and `set on less than (slt)` instructions
3. branch instructions; i.e., `branch equal (beq)` (for conditional branch) and `jump (j)` (for unconditional branch) instructions

The microarchitecture described in this section, and subsequent sections, shall be based on the following design principles [87]:

1. Regularity in the ISA and hardware provides simplicity in the design of the processor architecture.
2. Smaller hardware components are faster than larger hardware components.
3. Hardware resources that are commonly used shall be optimized for performance.
4. Good processor architecture designs involve making good trade-offs between design objectives, while satisfying design constraints.

The remainder of this section is organized as follows: in Subsection 2.1, I will briefly describe how the datapath of a single-cycle processor can be designed; subsequently, in Subsection 2.2, I will describe how the control path of a single-cycle processor can be designed; and, lastly, I will compare the advantages and disadvantages of the single-cycle processor in Subsection 2.3.

2.1 Datapath Design for Single-Cycle Processor

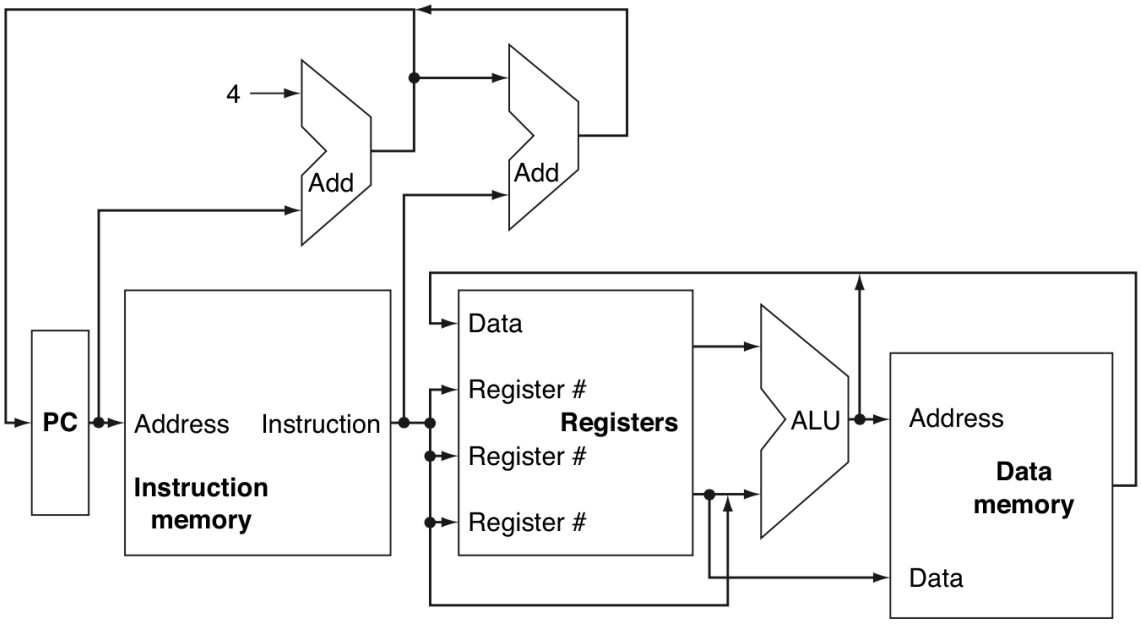

Figure 1: An abstract (or high-level) view of a *MIPS* processor's datapath [85, 86].

Figure 1 shows the datapath of a simple, single-cycle microarchitectural implementation of the *MIPS* ISA. The datapath of a microarchitecture shows the hardware components, or logic circuits, needed to perform computation of arithmetic and logical operations as well as read/write data into memory. Its main components are logic circuits that:

1. Fetch an instruction from memory, and update the program counter (PC).
2. Decode the instruction, and read register(s) (from the register file, `regfile`) used by that instruction. The `regfile` is shown as `Registers` in Figure 1.
3. Perform computation on an arithmetic/logical operation; i.e., the arithmetic-logical unit (ALU).
4. Read data from or write data to a memory device, or memory system (including caches). Separate memory devices can be used to store data and instructions separately. In Figure 1, instruction memory is a memory device that stores instructions and data memory is another memory device that stores data.

These main components for the single-cycle microarchitectural implementation of the *MIPS* ISA are also used in other microarchitectural implementation of the *MIPS* ISA, such as the multi-cycle *MIPS* processor, pipelined *MIPS* processor, and the superscalar *MIPS* processor. The VLSI implementation of the main datapath components are briefly described as follows. The PC can be implemented as a binary counter or linear-feedback shift register that increments a binary number (either a 32- or 64- bit number) by four each clock cycle. A simple memory device for physical memory can be implemented using a dynamic random-access memory (DRAM). Alternatively, a simple memory system can be implemented with a three-level cache system and a DRAM, where a cache in either of the three levels can

be implemented as a static random-access memory (SRAM). The ALU can be implemented with arithmetic circuits, such as adders, multipliers, and dividers. The ALU also includes comparators, shifters, one/zero detectors, encoders, and decoders [116].

Regarding the aforementioned design principles, I will discuss how this basic datapath satisfies the design principles mentioned earlier in Section 2. Firstly, regularity in the *MIPS* ISA is found in majority of instructions that require the ALU to perform arithmetic or logic operations. This implies that a set of datapath components and a subset of the control logic can implement most instructions in the *MIPS* ISA. In addition, the regularity in the design of datapath components, such as adders in the ALU and DRAM devices, results in their simplicity. This allows architects to design simpler and faster datapaths that do not need to interface with complex control logic. Secondly, as for smaller hardware components, components of the ALU (such as adders and shifters) are optimized for performance within area constraints. Also, for memory systems that include a multi-level cache system, the first level cache is designed to be smaller that the lower-level caches and the memory device. This allows the cache to be significantly faster than lower-level caches and the memory device. Thirdly, since most of the instructions use the ALU, a significant amount of work has been done to optimize ALU designs so that computer performance can be improved [31,32,81,108]. Lastly, the design trade-offs made for this simple datapath of a single-cycle *MIPS* processor demonstrates a trade-off between performance and hardware cost.

2.2 Control Path Design for Single-Cycle Processors

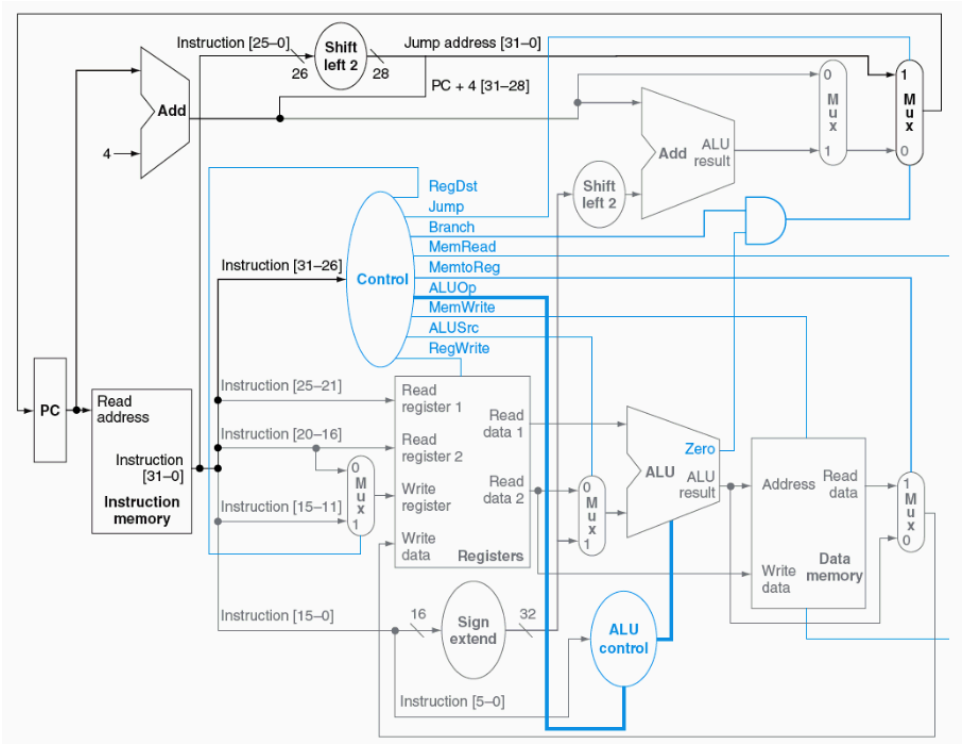

Figure 2: Control path of a single-cycle *MIPS* processor, which is superimposed on the datapath of the *MIPS* processor [85, 86].

The control path of a processor includes a control unit to generate control signals for the processor, and wires to transmit those control signals from the control unit to various datapath components of the

processor. This control unit for a single-cycle *MIPS* processor does not have multiple sequential states. Hence, the control unit can be implemented as a combinational logic circuit [84].

To implement the control unit as a combinational circuit, map the inputs of the control unit to its outputs. The mapping for a set of input signals to an output signal is a boolean function. This mapping can be represented as a truth table [84] or a data structure that is typically used to represent a boolean function. Examples of such data structures include binary decision diagrams (BDDs) [40, 42] and AND-Inverter graphs (AIGs) [12, 114]. Electronic design automation (EDA) software can be used to optimize the size of truth tables, BDDs, or AIGs [114]. By using EDA software to design and optimize the combinational circuit for the control unit, the process of designing the control path can be sped up by automation created by EDA software; also, EDA software can be used to automatically verify the correctness of the resultant circuits [95, 96]. EDA software maps truth tables, BDDs, or AIGs to combinational circuits for the control unit. The combinational circuit for the control unit can be implemented as VLSI circuits using standard cells [54, 92, 116] or a programmable logic array (PLA) [84].

Figure 2 shows the datapath and the control path (in blue). The control path includes the control unit (shown as `Control`) and ALU control unit `ALU control`. The wiring of the buses and wires for the control path are colored in blue. The *PC* provides PC update, and the `Instruction memory` fetches instructions from the memory device for instructions. The `Registers regfile` allows data in registers to be read from the `regfile`, and the lower `ALU` provides computation of arithmetic and logical operations. The data memory allows data to be read from or written to memory [85, 86]. To summarize, sequentially ordered instructions from a software application would be loaded from memory into the processor for execution. After these instructions are executed, the resultant output data from executing those instructions would update the state of the processor. Next, data updated by the processor would be written to memory locations, or data would be read from memory. Finally, registers in the `regfile` may have their data updated.

2.3 Advantages and Disadvantages of Single-Cycle Processor Design

The single-cycle *MIPS* processor has the advantage of a simple datapath, which can be easily designed and verified. However, its major disadvantage is that it is computationally inefficient, since it requires all instructions to be executed in one clock cycle. Therefore, the CPI for the single-cycle *MIPS* processor is one. Since the latency of memory access instructions tends to be much slower than the latency of other instructions, the speed of the single-cycle *MIPS* processor is limited to the memory access speed [85, 86].

3 Multi-Cycle Processor Design

To improve the performance of the single-cycle *MIPS* processor (see §2), a better semiconductor manufacturing process technology can be used to manufacture implementations of the microarchitecture as a VLSI system. Alternatively, the microarchitecture can be modified to significantly improve the performance of the processor. For each instruction in the *MIPS* ISA, decompose the instruction into steps based on the functional units required to complete executing that instruction. That is, if an instruction needs five components, it shall be decomposed/partitioned into five steps. Each step of the decomposed instruction shall take one clock cycle. In general, the datapath in Subsection 2.1 (see Figure 1) can be decomposed into the following five stages [85]:

1. Instruction fetch (IF). Instructions are fetched from memory and the program counter (PC) is updated.

2. Instruction decode (ID). Instructions are decoded and register(s) are read from the register file `regfile`.
3. Execution (EX). Arithmetic/logical operations are performed by the arithmetic-logical unit (ALU).
4. Memory access (MEM). Read data from or write data to the memory device or memory system (including caches).
5. Write-back (WB). Complete memory read operations (i.e., load operations).

Each of the stages listed above can be a step for a decomposed instruction. Since different instructions require different number of stages, the CPI for instructions in the *MIPS* ISA would vary for microarchitectures that allow instructions to complete in a range of clock cycles. These microarchitectures are known as multi-cycle processors. A multi-cycle processor allows a functional unit to be used multiple times per clock cycle. This reduces the amount of required hardware resources/components and encourages microarchitects (or processor architects) to share hardware resources between different steps of an instruction [85].

The rest of this section is organized as follows. In Subsection 3.1, I discuss the datapath design for the multi-cycle *MIPS* processor. Following this, in Subection 3.2, I describe the control path of the multi-cycle *MIPS* processor, including a brief discussion of how to implement the finite state machine representation of the control unit (in the control path) as a sequential circuit (see §3.2.1). Next, I compare the advantages and disadvantages of the multi-cycle *MIPS* processor with the single-cycle *MIPS* processor (see §2) in Subsection 3.3. Lastly, in Subsection 3.4, I discuss how microprogramming can be used to reduce the hardware complexity of a microarchitecture's control path, and reduce the amount of effort needed to design that microarchitecture as a VLSI system.

3.1 Datapath Design for Multi-Cycle Processor

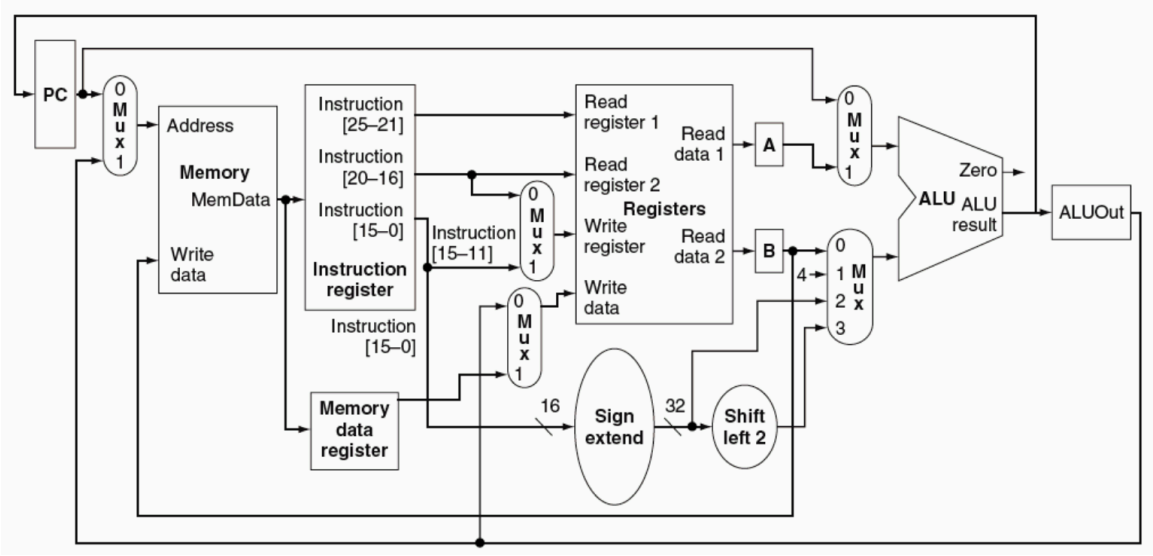

Figure 3: Datapath of a multi-cycle *MIPS* processor [85].

Figure 3 shows the datapath of a multi-cycle *MIPS* processor [85]. The datapath design is obtained by partitioning the datapath of the single-cycle *MIPS* processor into hardware components needed for decomposed steps of instructions. As aforementioned, datapath components (or hardware resources in the datapath) of the single-cycle *MIPS* processor (see Figure 2) are shared to reduce the number of

components in the datapath. For example, the data and instruction memory devices are combined into a memory device for data and instructions. In addition, only one ALU is needed in the datapath, since it can be reused in different clock cycles during the execution of an instruction [85].

For a given instruction, it can use data for multiple clock cycles, since an instruction would require more than two clock cycles to load and complete execution. Therefore, state elements are used to provide this data for multiple clock cycles. State elements are visible to software developers and include the following: PC counter, register file, memory device/system, and registers (for temporary storage of data that will be used in later cycles of a given instruction). The following temporary registers are added to the microarchitecture of the multi-cycle *MIPS* processor [85]:

1. Instruction register (IR). IR is used to save the output of an instruction read operation for use in the same clock cycle.
2. Memory data register (MDR). MDR is used to save the output of an data read operation for use in the same clock cycle.
3. Registers `A` and `B`. They are used to hold the values of register operands of an instruction and serve as a buffer between the register file and the ALU.
4. Register `ALUOut`. It holds the output of the ALU.

Therefore, for the multi-cycle *MIPS* processor to function correctly, the clock cycle has to be determined by the longest latency of the aforementioned five stages. In turn, this determines the performance of the multi-cycle *MIPS* processor.

3.2 Control Path Design for Multi-Cycle Processor

The control unit for a multi-cycle *MIPS* processor has multiple sequential states. Therefore, this control unit needs to be implemented as a sequential logic circuit. The sequential behavior of a sequential logic/digital circuit can be represented as a finite state machine (FSM) [84]. Subsection 3.2.1 provides a description of transforming a FSM representation of the control unit into a sequential digital circuit. Alternatively, microprograms can be used to implement the control path of the processor and simplify the microarchitecture design of the multi-cycle processor (see §3.4) [85].

Figure 4 shows the FSM for a multi-cycle *MIPS* processor [84, 85]. It indicates the states that the processor can be in and how a state can transit to another state. The next state function determines the transition from the current state to the next state. The stages IF, ID, and WB are each represented as a stage in the FSM. Instructions that require the use of the ALU are categorized into memory access operations, execution, branch completion, and jump completion. Each category is represented as a state in the FSM. Similarly, the memory access operations for `read` and `write` are each represented as a state [85]. Subsection 3.2.1 briefly describes how this FSM can be mapped into a sequential circuit.

Figure 5 shows the microarchitecture of the multi-cycle processor that includes the control path (colored in blue) and the datapath [85]. It integrates the datapath (see Figure 3) in Subsection 3.1 with the control logic (see Figure 7 in §3.2.1) and associated wiring. It also has addition control logic for detecting and handling exceptions.

3.2.1 Finite State Control

The process of implementing a FSM as a sequential circuit is described briefly as follows.

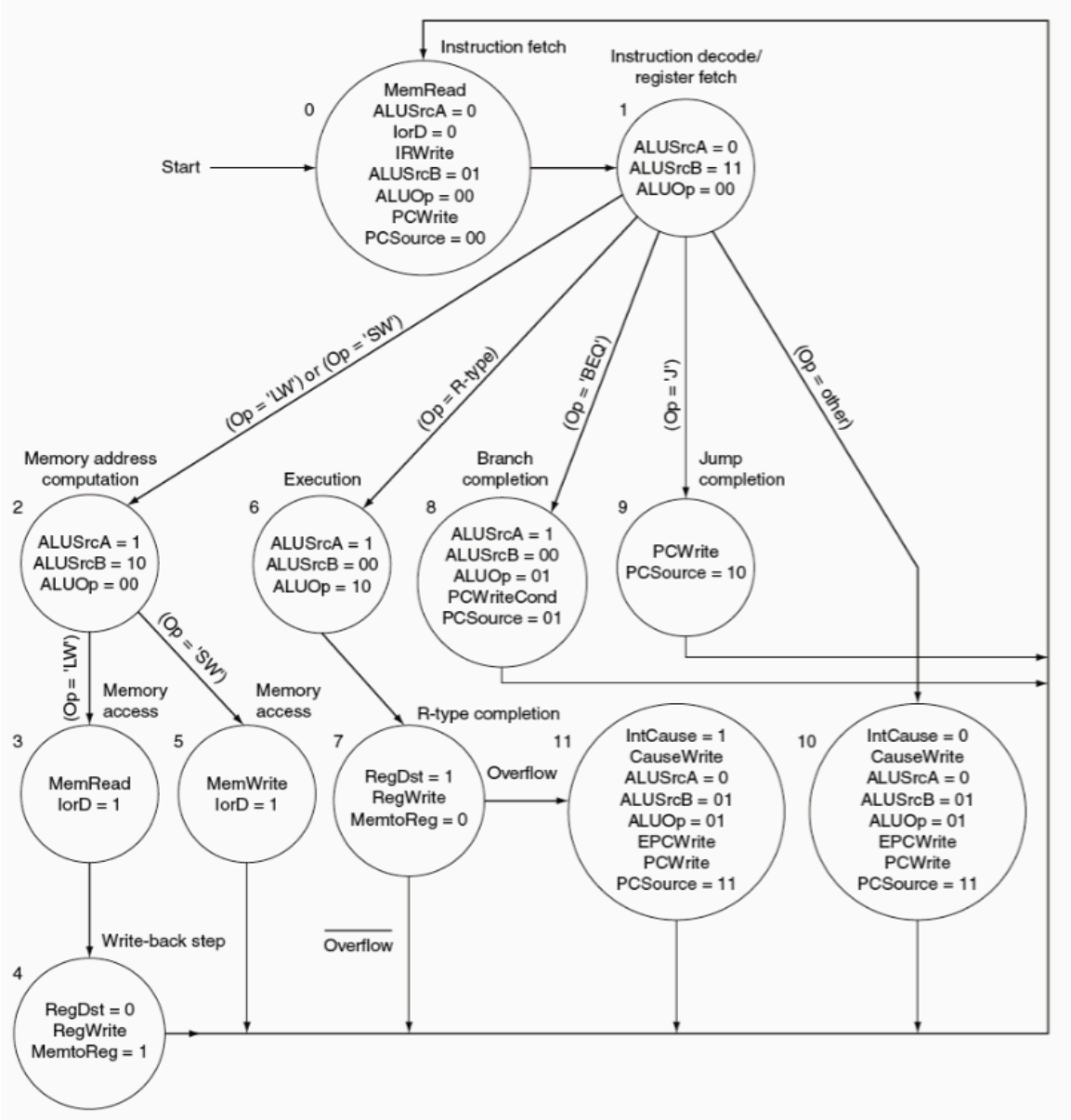

Figure 4: Finite state machine representation of the control unit for a multi-cycle *MIPS* processor [84,85].

Firstly, label each state in the FSM as a binary number, which would be stored in a state register in the sequential circuit [84]; a state register is a flip-flop circuit [54,92,116]. Next, represent each state and output signal in the FSM as a boolean variable. Determine a boolean function of states to obtain the boolean value for each output signal in the FSM. The combinational circuit derived from this boolean function is the output logic circuit for the FSM. Subsequently, design the combinational circuit needed to determine the next state for a given state in the FSM. That is, for each possible next state in the FSM, represent that state as a boolean variable. The value of the boolean variable is determined by a boolean function of the states and output signals of the FSM. Here, each state or output signal is represented as a boolean variable. The second boolean function is mapped to a combinational circuit. This combinational circuit is called the next state logic circuit for the FSM.

Figure 6 shows the canonical circuit topology for a synchronous sequential circuit, which encapsulates the next state logic and the output logic circuits as the combinational circuit C and represents

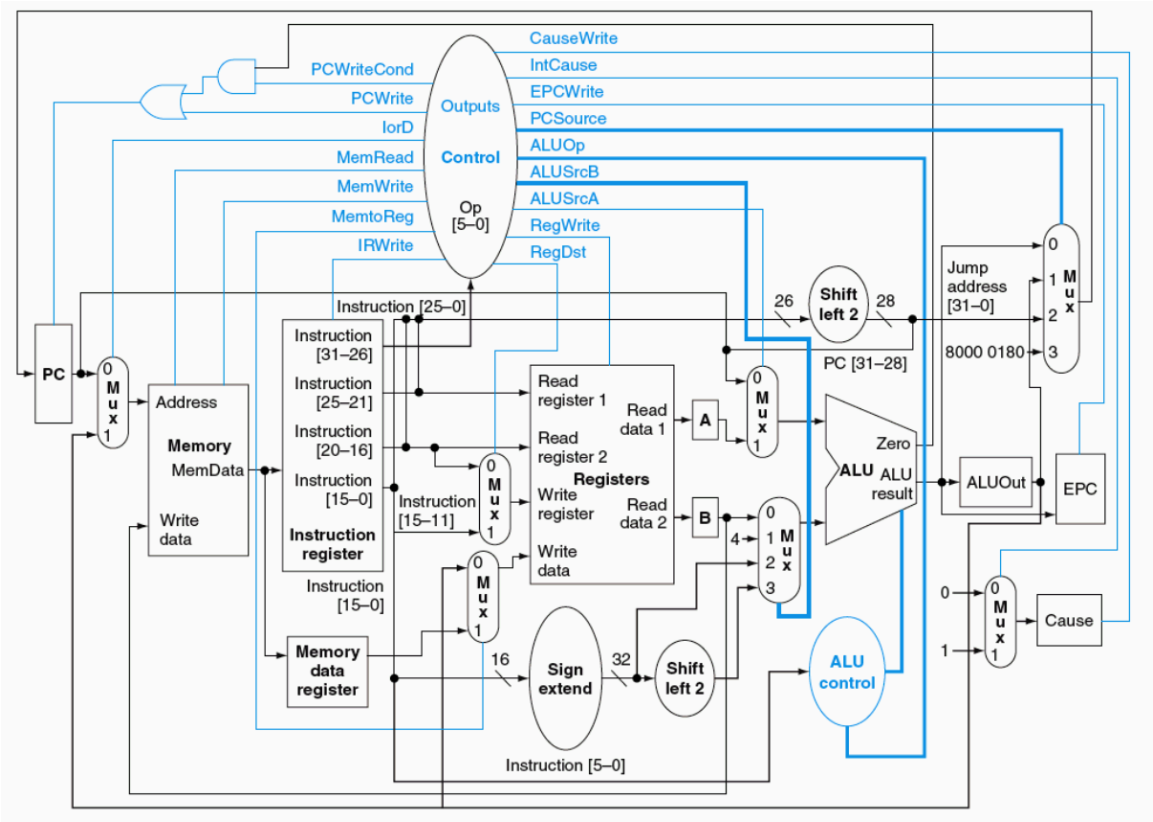

Figure 5: Microarchitecture for a multi-cycle *MIPS* processor [85].

state storage as a pair of flip-flops (F/F). The clock signal `CLOCK` indicates that this sequential circuit is synchronous. Figure 7 shows a schematic for a sequential circuit implementing the FSM (see Figure 4) of the multi-cycle *MIPS* processor's control unit in this canonical structure.

The combinational circuit C in Figure 6 or the control logic in Figure 7 can be implemented using standard cells [54, 92, 116], a read-only memory (ROM) device, or a programmable logic array (PLA). The ROM device or PLA encodes the truth tables for the boolean functions in the combinational circuit C.

3.3 Advantages and Disadvantages of Multi-Cycle Processor Design

The advantage of the multi-cycle processor design over the microarchitecture for the single-cycle microarchitecture is the lower amount of hardware resources that are required in the datapath. From Equation 3, we can relatively compare the performance of the multi-cycle processor with the single-cycle processor. The CPI for the multi-cycle processor is higher than that of the single-cycle processor, since the multi-cycle processor takes multiple cycles to complete executing each instruction. However, the multi-cycle processor has a lower clock cycle period. If the normalized decrease in the clock period of the multi-cycle processor is relatively greater than the normalized increase in the CPI of the multi-cycle processor, the multi-cycle processor would have better performance than the single-cycle processor.

3.4 Microprogramming

Contemporary processors implement ISAs that include more than 100 instructions, and these instructions can have clock cycle latencies ranging from 1 to 20 CPI. Therefore, manually detailing the control

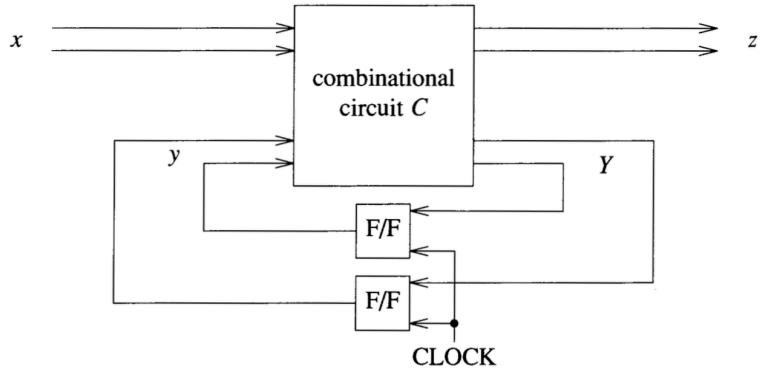

Figure 6: A schematic for the canonical circuit topology for a synchronous sequential circuit. The clock signal `CLOCK` provides synchrony for the sequential circuit. The combinational circuit C represents the next state logic and output logic circuits of the sequential circuit's finite state machine (FSM) representation, and the pair of flip-flops (F/F) provide the state storage for the FSM [2].

signals in a schematic of the VLSI circuit implementation of a microarchitecture is error prone. Likewise, a FSM representation of the control unit of a processor would be cumbersome to manually describe too. There would be too many states (i.e., more than 1000 states) in the FSM representation. This means that it would be very difficult to verify such large FSMs [84].

Therefore, microprogramming is used to address this processor design challenge by treating each set of control signals as a microinstruction (or low-level control instruction) that can be executed by the datapath. This instruction would represent a state in the FSM for the control unit. That is, for each state in the FSM of the control unit, represent the output control signals from that state as a microinstruction to be executed in the datapath. It allows a set of (datapath) control signals to be symbolically represented as a microprogram, which consists of multiple microinstructions. Since many instructions in an ISA, such as the *MIPS* ISA, would share common sequences of microinstructions, a lot of code reuse of these common sequences (or microcode) occurs. Representing control signals as microcode/microprograms instead of wires, buses, and digital circuits allow the design of a microarchitecture to have a simpler control path [84].

4 Pipelined Processor Design

Pipelining is a microarchitecture design technique that overlaps instructions, so that more hardware resources (or datapath components) in the datapath pipeline can be utilized. This allows computer programs to execute faster, and improve the performance of the pipelined processor. Figure 8 shows how the datapath of a pipelined *MIPS* processor can be pipelined to utilize more datapath components in each clock cycle. This pipelined *MIPS* processor fetches a new instruction every clock cycle, and each pipelined stage of the processor is processed in a clock cycle. At the next clock cycle, a given stage in the pipelined datapath would process the next instruction. An example of maximizing the usage of datapath components is shown in the fifth clock cycle, when all pipeline stages of the processor are being utilized. Therefore, pipelined processors are faster than multi-cycle processors, since the throughput of executing instructions is increased [87] to reduced the average execution time per instruction [44].

The performance speedup for a pipelined *MIPS* processor over an unpipelined *MIPS* processor is

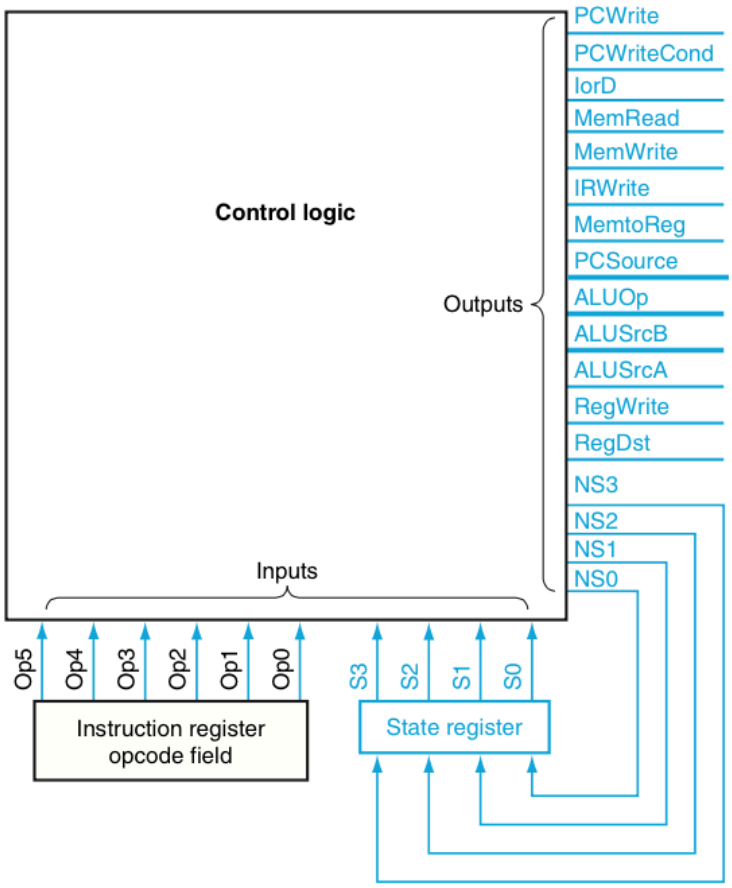

Figure 7: A sequential circuit representing the FSM for the control unit of the multi-cycle *MIPS* processor [84]. Its circuit topology has the canonical structure of sequential circuits shown in Figure 6. The state register provide the state storage, and the control logic includes the next state logic and the output logic circuits needed to implement the FSM for the control unit of the processor.

given as follows [44]:

$$\text{performance speedup from pipelining} = \frac{\text{average execution time per unpipelined instruction}}{\text{average execution time per pipelined instruction}} \tag{7}$$

Figure 9 shows the microarchitecture of a pipelined *MIPS* processor that supports detection and handling of exceptions [87]. The pipelined stages of the pipelined *MIPS* processor's datapath are the same stages of the multi-cycle *MIPS* processor; see Section 3. That is, the pipelined stages of the pipelined *MIPS* processor are: IF, ID, EX, MEM, and WB. In order to execute each pipelined stage in a clock cycle, pipelined registers are placed between pipelined stages.

Pipeline hazards prevent the next instruction in the instruction sequence to execute during the clock cycle that it is supposed to. These pipeline hazards prevent the computer from achieving the ideal performance speedup. These hazards are categorized as follows [44]:

1. Structural hazards occur when multiple instructions need to use a datapath component in the same clock cycle. They prevent instructions from being overlapped in sequences, such that multiple instructions need to use a datapath component in the same clock cycle.
2. Data hazards occur when the computational results of an instruction are used or overwritten in subsequent instructions.

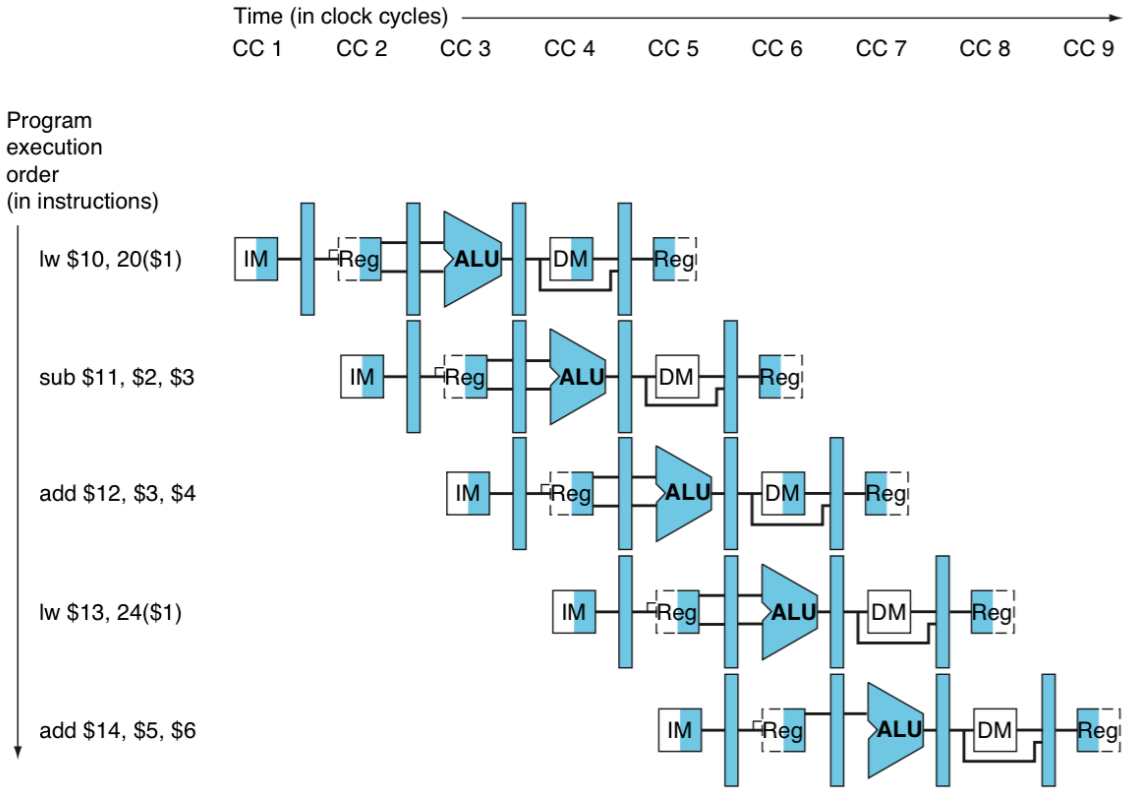

Figure 8: A diagram of the datapath pipeline of the pipelined *MIPS* processor that shows the order of instruction execution for a sequence of five *MIPS* instructions. This datapath pipeline shows that each instruction would take at at least 5 clock cycles to execute [87].

3. Control hazards occur when pipelined `branch` and `jump` instructions change the PC value.

An extensive coverage of pipeline hazards is beyond the scope of this report. However, I will briefly address data dependencies and data hazards. Figure 10 shows an example of Read-After-Write (RAW) data dependencies [44, 100] between instructions in a snippet of a computer program [87]. There exist Read-After-Write (RAW) data dependencies [44, 100] regarding register `$2` between the instruction `sub $2, $1, $3` and the following instructions: `and $12, $2, $5`; `or $13, $6, $2`; `add $14, $2, $2`; and `sw $15, 100($2)`. Figure 11 shows how these data dependencies can be resolved by forwarding dependent data from a datapath component to another datapath component in the same or next clock cycle [87]. For example, there exists RAW data dependencies [44, 100] regarding register `$2` between the instruction `sub $2, $1, $3` and the following instructions: `and $12, $2, $5`; `or $13, $6, $2`; and `add $14, $2, $2`. Therefore, forwarding is used to transfer the updated value of register `$2` to: the input of the ALU for the instruction `and $12, $2, $5`; the input of the ALU for the instruction `or $13, $6, $2`; and the register file at `add $14, $2, $2`. The RAW data dependency for register `$2` in last instruction `sw $15, 100($2)` will not cause a data hazard, since the value at register `$2` will be ready to be read from the register file at cycle six.

Figure 12 shows an example of how some data dependencies cannot be resolved by forwarding of data values to another datapath component and these data dependencies will cause data hazards. The register `$2` is written by the `lw` instruction and is ready at the end of the fifth cycle. However, it is also required in the subsequent two instructions `and` and `or` in previous cycles. Hence, the data dependencies between these instructions have caused data hazards that will affect the correct functionality of the processor. To address the problem of data hazards, bubbles are used to delay the execution of

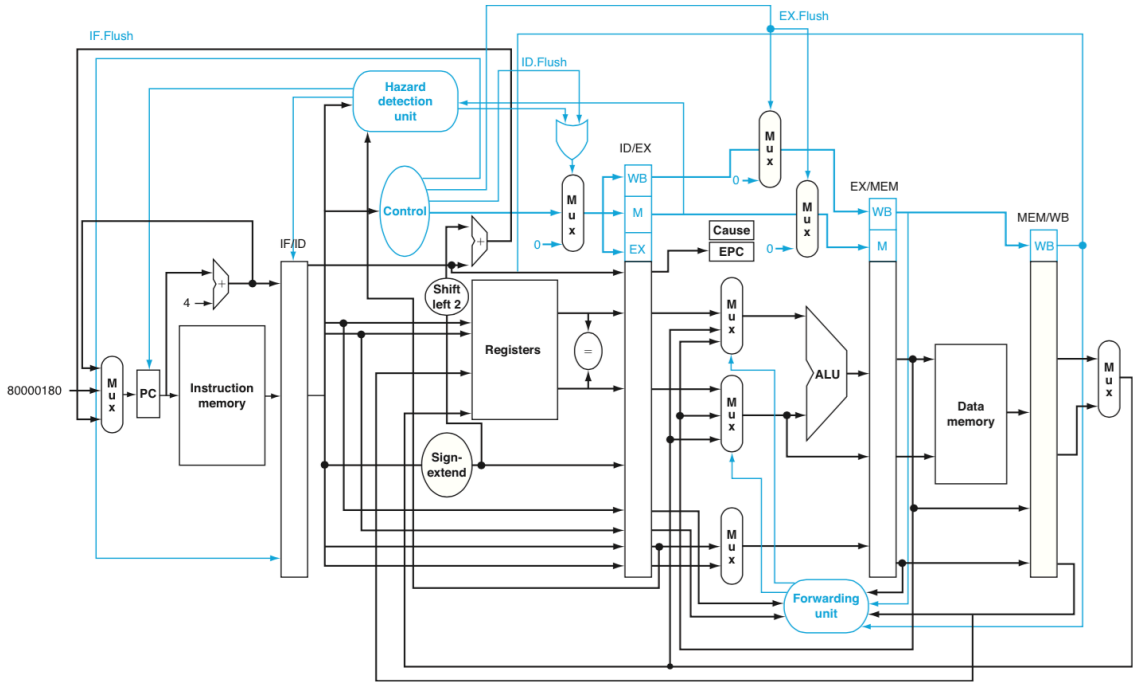

Figure 9: Microarchitecture of a pipelined *MIPS* processor that includes the datapath and the control path. It supports detection and handling of exceptions [87].

instructions by flushing contents of the datapath pipeline that are associated with instructions causing these data hazards. Figure 13 shows how bubbles/stalls can be inserted into the datapath pipeline of a pipelined *MIPS* processor to avoid data hazards [87]. Bubbles (or `nop` instructions) are inserted to flush contents of the ALU, memory device, and associated `$2` register in the register file.

Lastly, to improve the performance of pipelined processors regarding control hazards, techniques based on static branch prediction and dynamic branch prediction can be used. The former category of branch prediction techniques are carried out by compilers during compilation time, while the latter category of branch prediction techniques are carried out at run-time by the branch prediction unit in the pipelined microarchitecture [44, 100].

5 Superscalar Processor Design

To improve the performance of pipelined processors, multiple instructions can be issued at the same time (i.e., superscalar instruction issue) in the pipelined microarchitecture; scalar processors can only issue one instruction per clock cycle. Therefore, superscalar processors allow the throughput of instruction execution to increase by exploiting instruction-level parallelism via instruction scheduling, loop unrolling, and branch prediction to reduce unnecessary stalls in the pipelined datapath [44, 100]. However, the scope of literature on techniques to improve the performance of superscalar processors, including superpipelined and superscalar processors, is vast. Hence, I would only briefly mention the important concepts behind superscalar processors.

A brief description of static and dynamic pipelined, superscalar processors is provided as follows. Figure 14 shows the static datapath pipeline of a superscalar processor [87]. It issues multiple instructions per clock cycle, and has two separate datapath pipelines of functional units to support superscalar

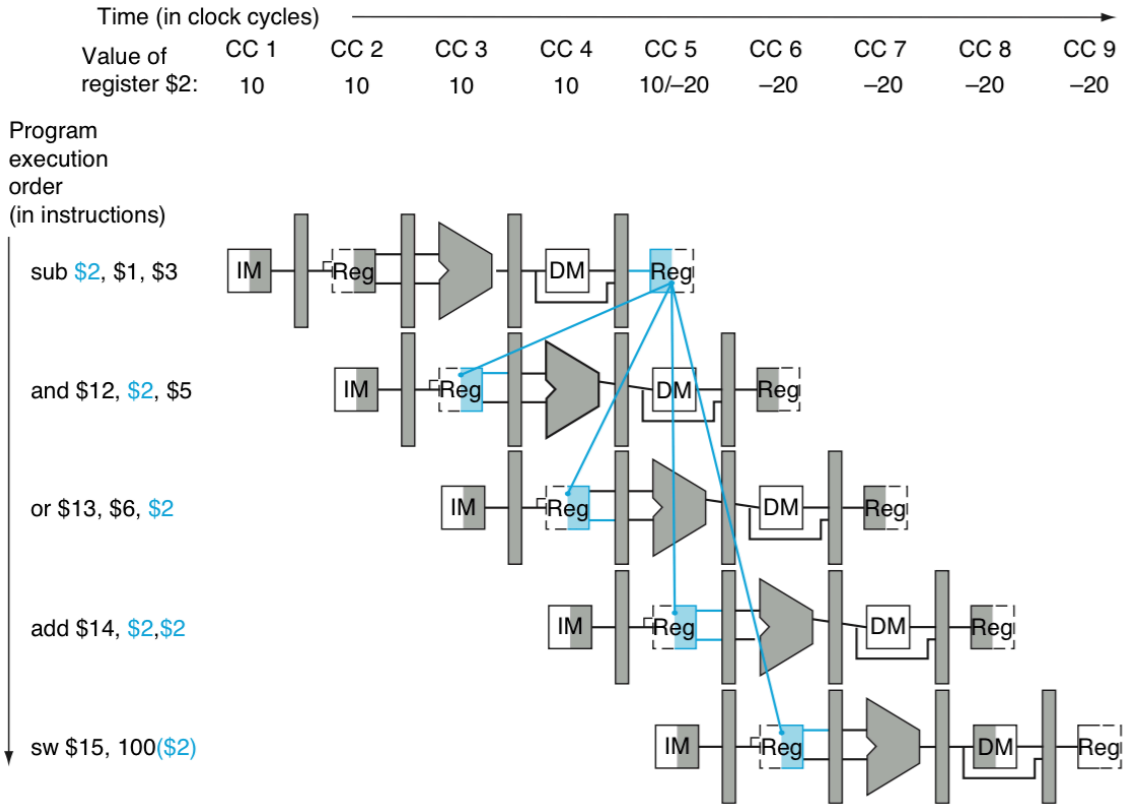

Figure 10: An example of Read-After-Write (RAW) data dependencies [44, 100] between instructions in a snippet of a computer program [87]. There exists RAW data dependencies regarding register `$2` between the instruction `sub $2, $1, $3` and the following instructions: `and $12, $2, $5`; `or $13, $6, $2`; `add $14, $2, $2`; and `sw $15, 100($2)`.

execution of instructions. In Figure 15, a dynamic datapath pipeline of a superscalar processor is shown [87]; such superscalar processors are also known as out-of-order, superscalar processors. Instruction scheduling and loop unrolling are exploited to allow instructions to be issued in order, executed out of order (due to instruction scheduling), and completed in order so that less bubbles have to be issued in the datapath pipeline. Here, loop unrolling increases the window size of instructions that the compiler or corresponding hardware can use to carry out instruction scheduling. In order for dynamic pipelined, superscalar processors to function correctly, reservation stations (or a dispatch buffer) is placed between the front-end of the processor (for instruction fetch and decode) and the execution stage. They ensure that instructions are issued in order as they appear in the instruction stream. In addition, reorder buffers in the commit unit must be used to complete and retire the executed instructions in order. This in-order retirement of instructions guarantees the functional correctness of running computer programs on that out-of-order superscalar processor [44, 100]. Figure 16 shows the dynamic pipelined datapath of the AMD Opteron X4 processor's microarchitecture [87]. It reflects multi-instruction issue in contemporary microarchitectures and the use of parallel functional units to support superscalar instruction issue, such as three integer ALUs in this example.

6 Conclusion

By examining the different microarchitectural implementations of the *MIPS* ISA, we can conclude that the ISA determines various aspects of a microarchitecture for that ISA. For example, the microarchi-

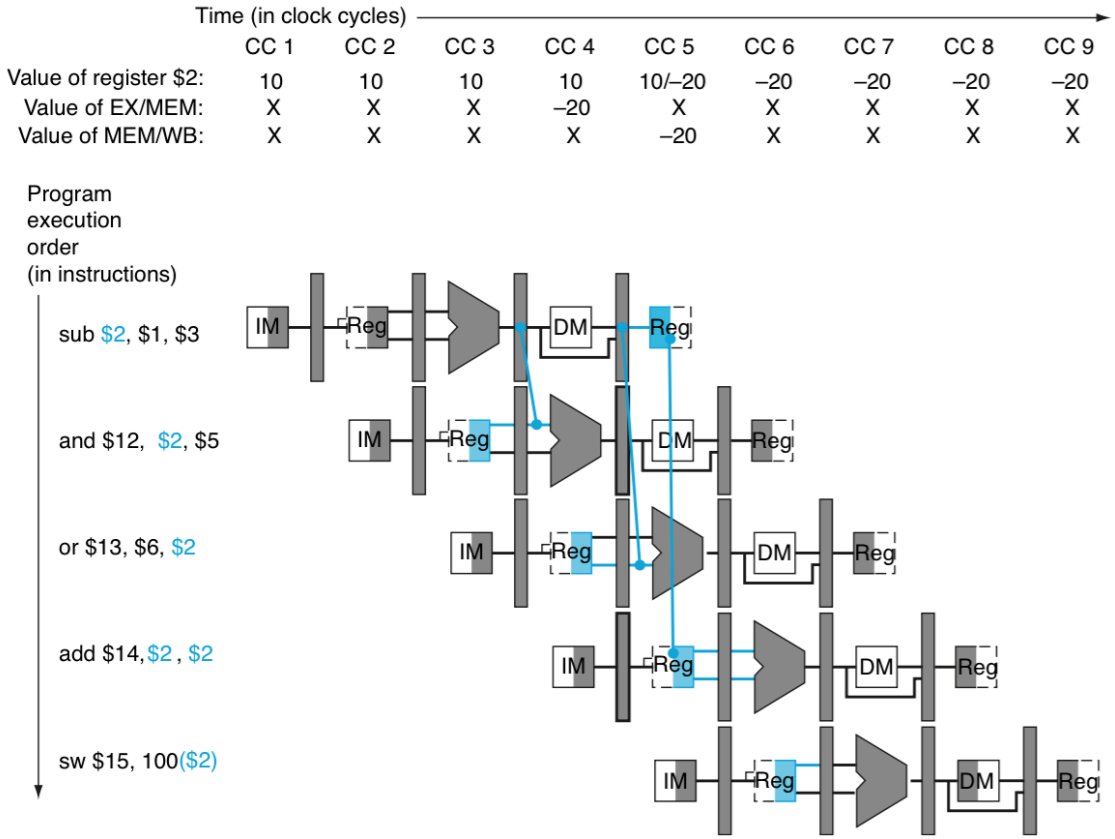

Figure 11: An example of data dependencies that can be resolved by forwarding. This avoids data hazards [87].

tecture for a multi-cycle processor (see Figure 5) would have a simpler control unit/path than the microarchitecture for a pipelined processor (see Figure 9). Also, we can observe that the microarchitecture design of a processor affects the performance of the computer system, since the clock rate (CCT) and the CPI for the processor is determined by its microarchitecture [85].

Furthermore, there is a trade-off between hardware complexity (and hence, design effort and power consumption) and performance for the three scalar processor architectures (single-cycle, multi-cycle, and pipelined processors) that are considered. Microarchitectures that have better performance have more hardware complexity, in terms of the datapath and the control path. For example, the aforementioned multi-cycle processor performs better than the single-cycle processor at the cost of more hardware complexity (in the datapath and the control path). Also, the pipelined processor trades off more hardware complexity than the multi-cycle processor for better performance. However, as we add move to superscalar and superpipelined microarchitectures [53], we have to consider a more nuanced trade-off between hardware complexity for the issue logic, instructions per cycle (IPC) (see Equation 8 for its definition), and computer performance [45, 80]. Comparing a heavily pipelined processor architecture to a not-so-heavily pipelined processor, the former may have worse performance than the latter, since the former would have a greater proportion of pipeline overhead per clock cycle. It means that the heavily pipelined processor wastes more computational resources for inefficient work than the not-so-heavily pipelined processor. Similarly, when comparing a superscalar processor with a wider instruction issue width to another superscalar processor of narrower instruction issue width, the former would have greater inefficiency than the latter. This is because the former would have greater hardware complexity (in terms of logic/circuit and wiring complexities) than the latter. Also, after issuing instructions in

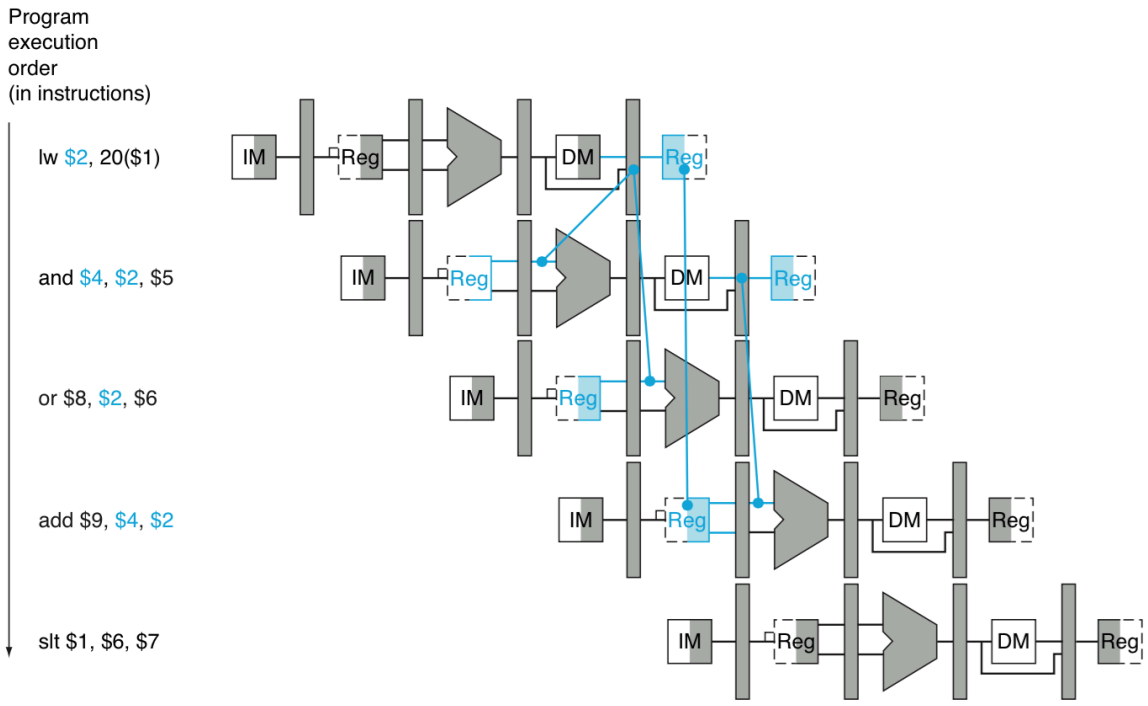

Figure 12: An example of data dependencies that cannot be resolved by forwarding and will cause data hazards [87].

order, the former would spend more computational resources than the latter in trying to execute instructions out of order and retiring them in order [44, 100].

In Figure 17, a plot of clock frequency (i.e., clock rate) versus instructions per clock cycle (IPC) is shown [85]. *IPC* is determined as follows [100]:

$$IPC = \frac{1}{CPI}, \tag{8}$$

where *CPI* refers to clock cycles per instruction (see Equation 3). The performance of a processor architecture design improves as design choices are enumerated from the bottom-left corner of Figure 17 towards its top-right corner. Data and control dependences of instructions in any given computer program limit how much instruction-level parallelism (ILP) can be exploited with pipelining and multiple instruction issue (i.e., superscalar processor design) for out-of-order execution. Therefore, in modern superscalar processor designs, instruction scheduling, dynamic branch prediction, and hardware speculation are used to improve processor performance [44, 85, 100].

A plot of the amount of hardware resource sharing versus instruction latency is shown in Figure 18. Like in Figure 17, the performance of a processor architecture design improves as design choices are enumerated from the bottom-left corner towards the top-right corner. As aforementioned, pipelining and the width of instruction issue cannot be increased indefinitely without incurring performance penalty; this is the "ILP wall". Recent history indicates that the "memory wall" and the "power wall" have become bottlenecks in microarchitecture design; the "memory wall" refers to the diverging rates of performance improvement between the processor and physical memory (i.e., main memory), and the "power wall" refers to the increasing power consumption of advanced processor designs from the late 1980s till the early 2000s. Hence, the processor design industry has shifted towards designing chip mul-

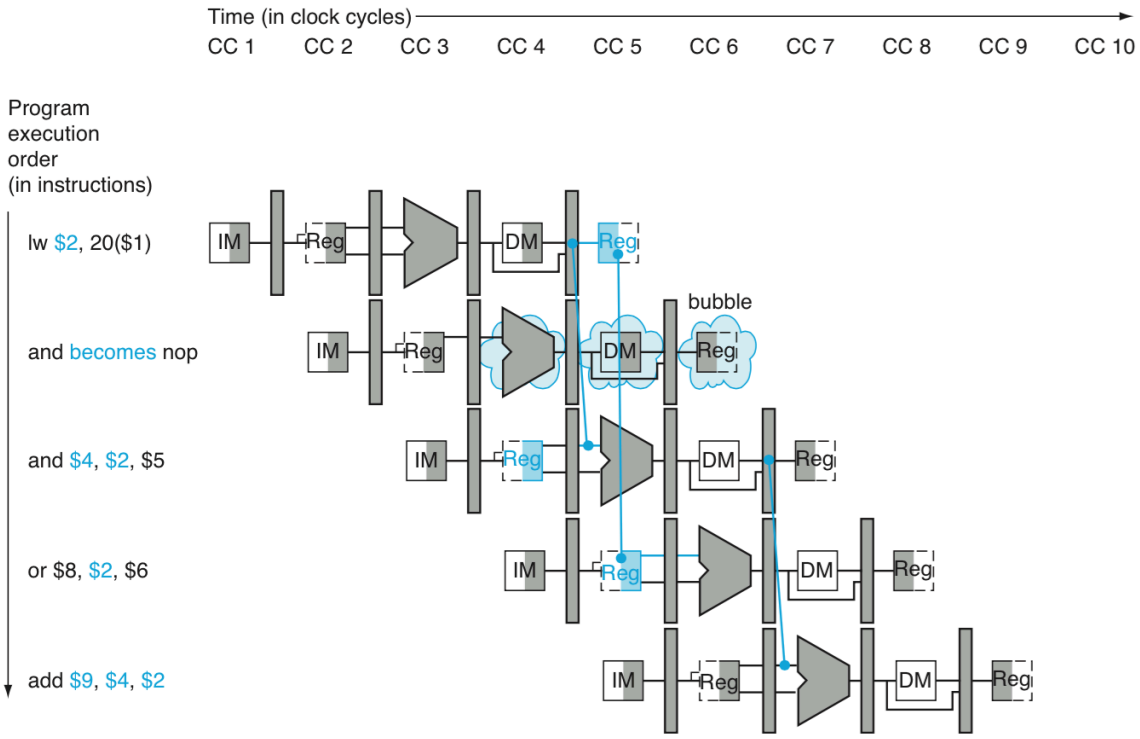

Figure 13: An example of how bubbles/stalls can be inserted into a pipelined *MIPS* processor to avoid data hazards [87].

tiprocessors to improve computer performance [44, 120].

Hence, wisdom and insight about the applications that a processor would run can be used while considering trade-offs between hardware complexity (as well as design effort and power consumption) with performance. However, performance speedup need not come with only microarchitectural improvements. Performance speedup can also come from designing computer architectures for other computing paradigms (such as re-revisiting dataflow processor architectures [28]). It can also come using alternative computer system design paradigms, such as application-specific instruction set processors [39, 55, 97], neuromorphic processors [11,60,61,69,70,99] and other cognitive computers, and general-purpose graphics processing units [4, 59, 62].

Lastly, to paraphrase Prof. Roberto Sebastiani [98], neither the problem of designing an efficient processor architecture, nor that of acquiring a comprehensive background of microarchitecture is easy. The challenge of designing high-performance, energy-efficient processor architectures is not just a daunting engineering task, but also an art in itself.

Acknowledgements

I would like to thank Prof. Richard McGuire and my classmates of my "Scientific Writing" class for helping me improve my technical writing skills. In addition, I would like to thank Ms. Kadie Henderson and Ms. Laura from the University Writing Center at Texas A&M University for reviewing my work and for helping me to find mistakes and errors in my report.

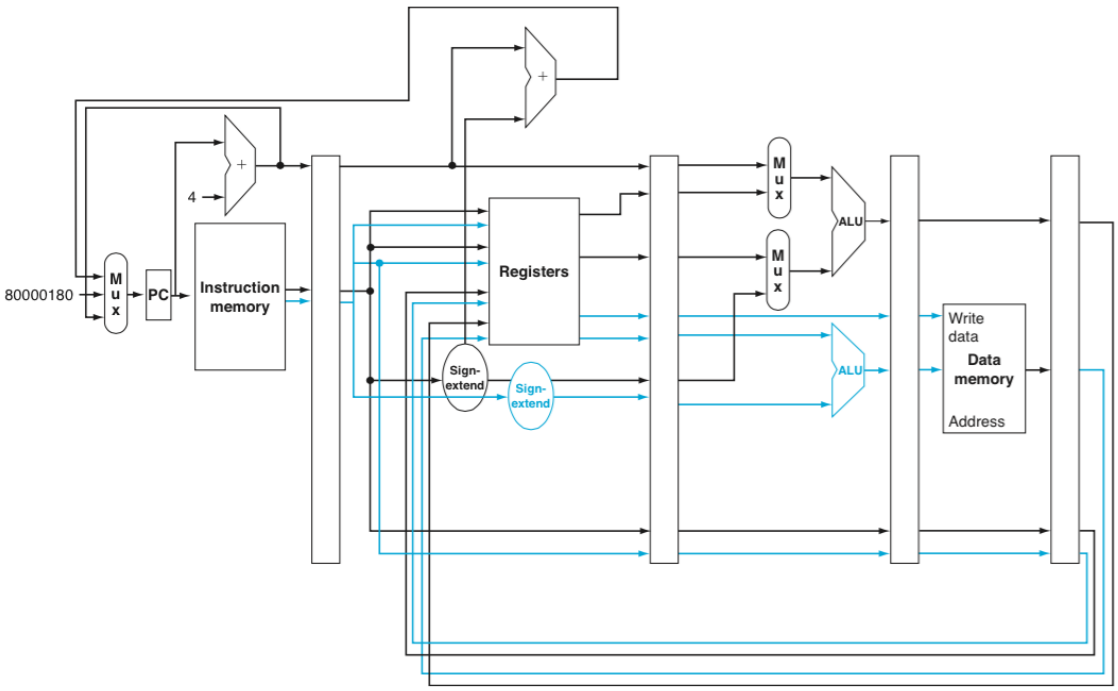

Figure 14: The static datapath pipeline of a superscalar processor [87].

References

[1] Mostafa Abd-El-Barr and Hesham El-Rewini. Fundamentals of Computer Organization and Architecture. Wiley Series in Parallel and Distributed Computing. John Wiley & Sons, Hoboken, NJ, 2005.

[2] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. Digital Systems Testing and Testable Design. IEEE Press, New York, NY, 1990. Revised Printing.

[3] Dennis Abts and John Kim. High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, March 2011.

[4] Erik R. Altman. CPUs and GPUs: Who owns the future? IEEE Micro, 31(5):2–3, September–October 2011.

[5] Pedro Less Andrade, Scott C. Beardsley, Beñat Bilbao-Osorio, Roberto Crotti, Bahjat El-Darwiche, Soumitra Dutta, Luís Enríquez, Peter Haynes, John Garrity, Ferry C. Grijpink, Anant Gupta, Jess Hemerly, Volkmar Koch, Bruno Lanvin, David Meer, M-H. Carolyn Nguyen, Alex "Sandy" Pentland, Robert Pepper, Matt Quinn, Gabriel Recalde, Patrick Ryan, Sergio Sandoval, Ramez T. Shehadi, Steven Spittaels, Malin Strandell-Jansson, Chris Taylor, and Walid Tohme. The global information technology report 2014: Rewards and risks of big data. Technical report, World Economic Forum Switzerland, Cologny, Geneva, Switzerland, 2014.

[6] Mohit Arora. The Art of Hardware Architecture: Design Methods and Techniques for Digital Circuits. Springer Science+Business Media, LCC, New York, NY, 2012.

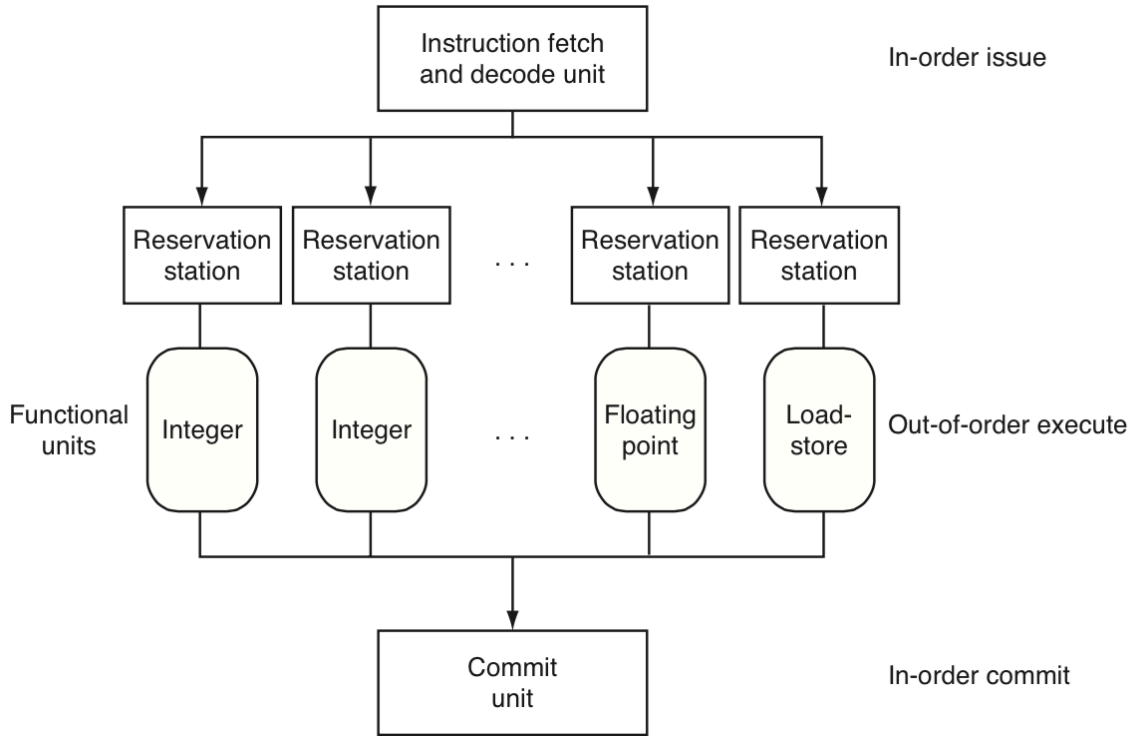

Figure 15: The dynamic datapath pipeline of a superscalar processor [87].

[7] Brian Bailey and Grant Martin. ESL Models and their Application: Electronic System Level Design and Verification in Practice. Embedded Systems. Springer Science+Business Media, LCC, New York, NY, 2010.

[8] Brian Bailey, Grant Martin, and Andrew Piziali. ESL Design and Verification: A Prescription for Electronic System-Level Methodology. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, San Francisco, CA, 2007.

[9] Rajeev Balasubramonian, Norman P. Jouppi, and Naveen Muralimanohar. Multi-Core Cache Hierarchies. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, November 2011.

[10] Luiz André Barroso, Jimmy Clidaras, and Urs Hōlzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, second edition, July 2013.

[11] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R. Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V. Arthur, Paul A. Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. Proceedings of the IEEE, 102(5):699–716, May 2014.

[12] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification, release 10508. Available online at Berkeley Logic Synthesis and Verification Group, Department of Electrical Engineering and Computer Sciences, College of Engineering, University of California, Berkeley: http://www.eecs.berkeley.edu/~alanmi/abc/; last accessed on May 10, 2011; Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, February 28 2011.

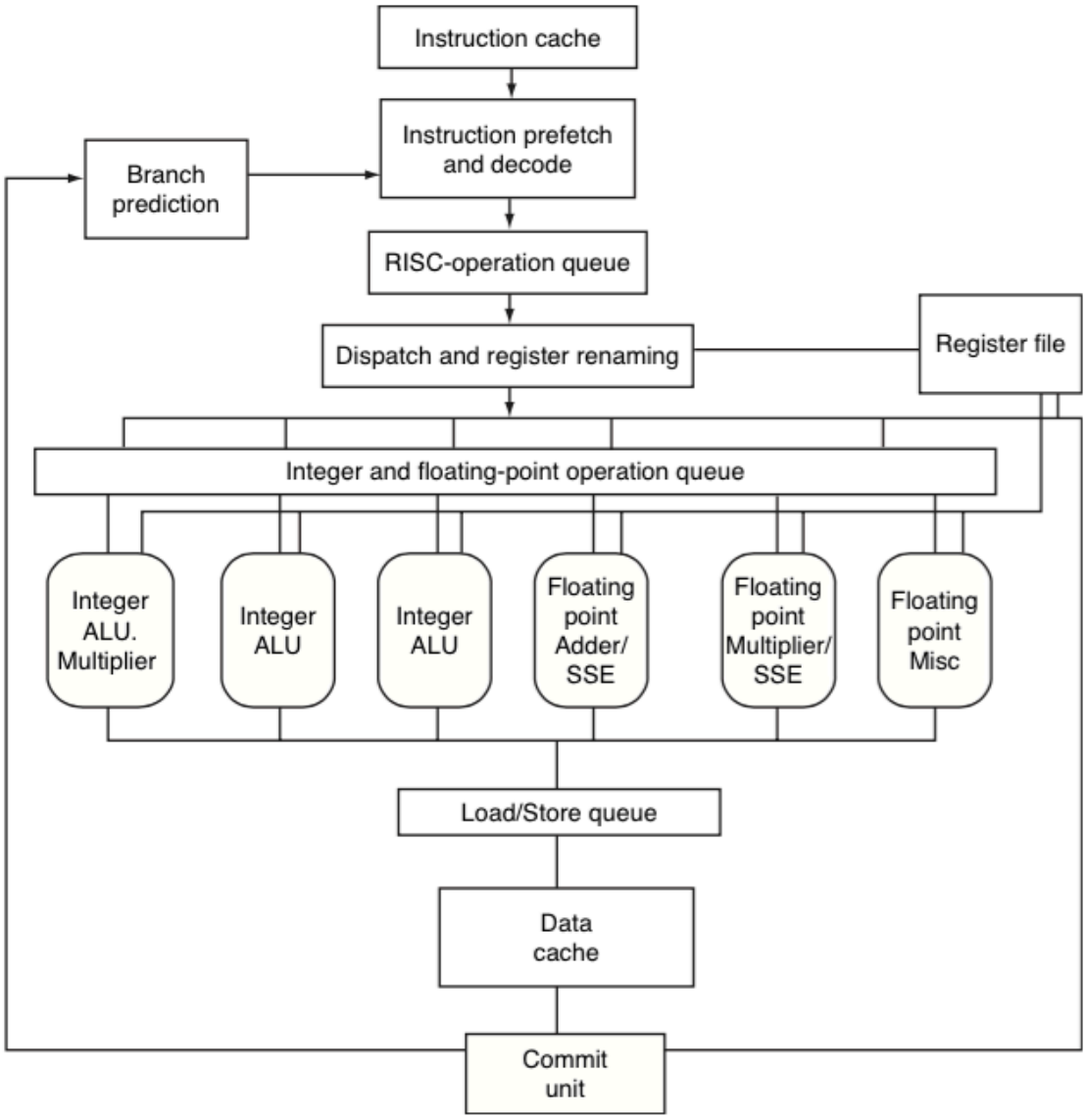

Figure 16: Dynamic pipelined datapath of the AMD Opteron X4 processor's microarchitecture [87].

[13] Carolyn Bertozzi, John F. Ahearne, Francisco J. Ayala, Andrea L. Bertozzi, David J. Bishop, Gary L. Comstock, Frances A. Houle, Deborah G. Johnson, Michael C. Loui, Rebecca R. Richard-Kortum, Nicholas H. Steneck, and Michael J. Zigmond. On being a scientist: A guide to responsible conduct in research. Technical report, National Academies Press, Washington, D.C., 2009.

[14] Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala. Handbook of Signal Processing Systems. Springer Science+Business Media, LCC, New York, NY, 2010.

[15] Christian Bienia. Benchmarking Modern Multiprocessors. PhD thesis, Princeton University, Princeton, NJ, January 2004.

[16] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. ACM SIGARCH Computer Architecture News, 39(2):1–7, May 2011.

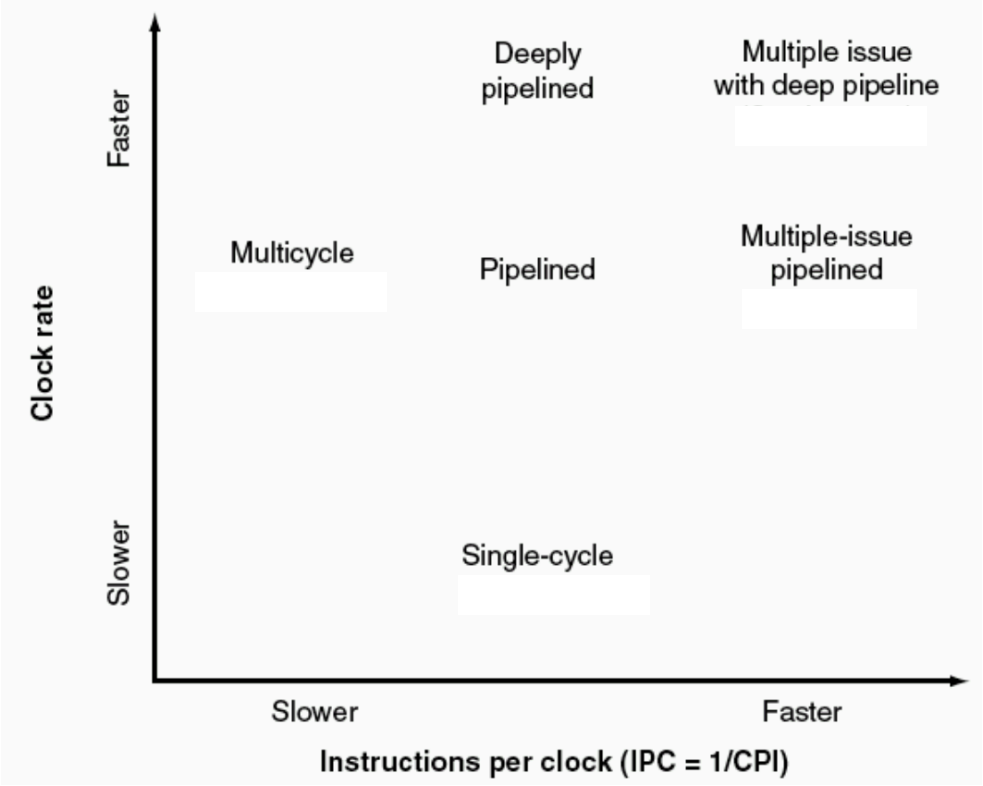

Figure 17: Plot of clock frequency versus instructions per clock cycle (IPC) [85]. $IPC = \frac{1}{CPI}$ (see Equation 8), where CPI refers to clock cycles per instruction (see Section 1). From the "iron law" in Equation 3, the clock rate (CCT) and IPC (or $\frac{1}{CPI}$) affects the performance of the processor. Hence, as we move from design solutions in the bottom-left corner towards those in the top-right corner, we can find design solutions that have better computer performance.

[17] Richard Blum. Linux Command Line and Shell Scripting Bible. Wiley Publishing, Indianapolis, IN, 2008.

[18] Bernd Bruegge and Allen H. Dutoit. Object-Oriented Software Engineering Using UML, Patterns, and Java. Pearson Education, Upper Saddle River, NJ, third edition, 2010.

[19] Sao-Jie Chen, Ying-Cherng Lan, Wen-Chung Tsai, and Yu-Hen Hu. Reconfigurable Networks-on-Chip. Springer Science+Business Media, LCC, New York, NY, 2012.

[20] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. Reliability, Availability and Serviceability of Networks-on-Chip. Springer Science+Business Media, LCC, New York, NY, 2012.

[21] Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, and Peter Z. Onufryk. Network Processor Design: Issues and Practices, volume 1 of The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, New York, NY, 2003.

[22] Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, and Peter Z. Onufryk. Network Processor Design: Issues and Practices, volume 2 of The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, San Francisco, CA, 2004.

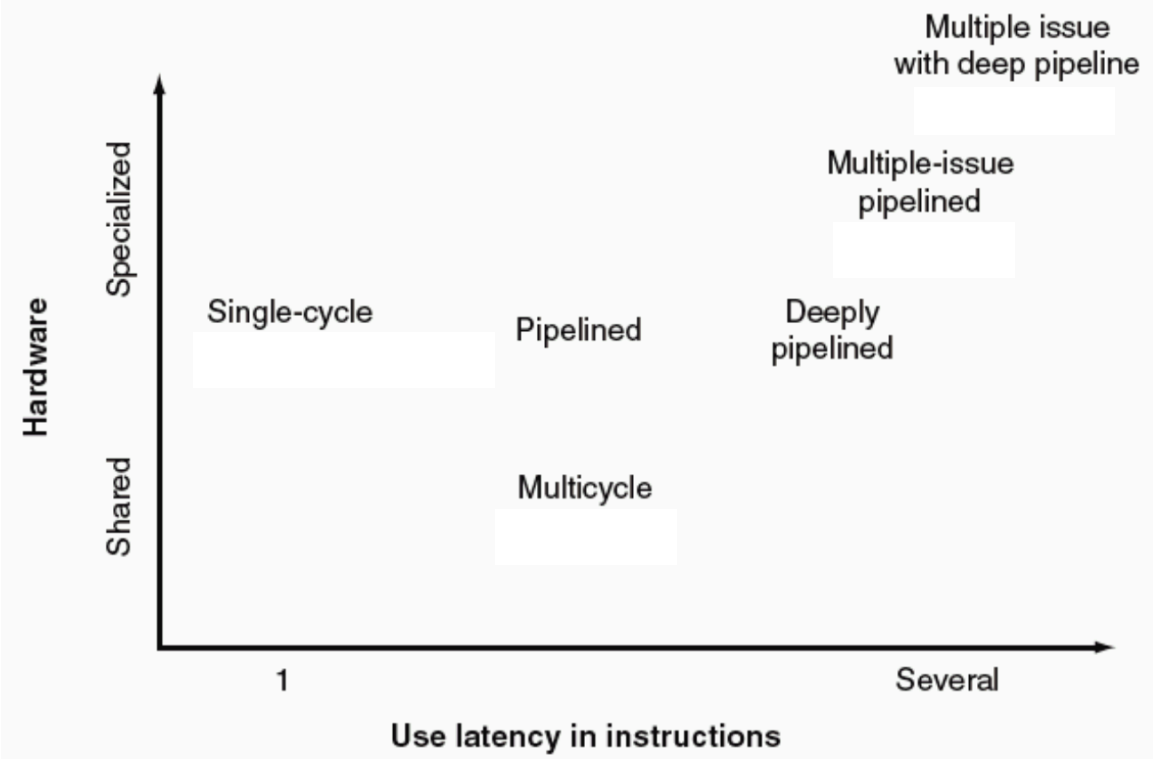

Figure 18: Plot of the amount of hardware resource sharing versus use latency in instructions [85]. The y-axis indicates the amount of specialized hardware that is used to execute instructions in a computer program, while the x-axis represents how easy it is to utilize all pipeline stages in a pipelined processor. As we move from design solutions in the bottom-left corner towards those in the top-right corner, we can find design solutions that have better computer performance.

[23] Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, and Peter Z. Onufryk. Network Processor Design: Issues and Practices, volume 3 of The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, San Francisco, CA, 2005.

[24] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. Parallel Computer Architecture: A Hardware/Software Approach. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, San Francisco, CA, 1999.

[25] William James Dally and Brian Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann, San Francisco, CA, 2004.

[26] Sivarama P. Dandamudi. Fundamentals of Computer Organization and Design. Texts in Computer Science. Springer-Verlag New York, New York, NY, September 22 2003.

[27] Giovanni De Micheli and Luca Benini. Networks on Chips: Technology and Tools. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, San Francisco, CA, 2006.

[28] Jack B. Dennis and David P. Misunas. A preliminary architecture for a basic data-flow processor. In Proceedings of the 2^{nd} Annual International Symposium on Computer Architecture (ISCA '10),

pages 126–132. Association for Computing Machinery, Association for Computing Machinery, 126–132.

[29] Michel Dubois, Murali Annavaram, and Per Stenström. Parallel Computer Organization and Design. Cambridge University Press, New York, NY, 2012.

[30] M. Duranton, D. Black-Schaffer, K. De Bosschere, and J. Maebe. The HiPEAC vision for advanced computing in horizon 2020. Technical report, European Network of Excellence on High Performance and Embedded Architecture and Compilation, Luxembourg, Luxembourg, March 2013.

[31] Miloš D. Ercegovac and Tomás Lang. Digital Arithmetic. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, San Francisco, CA, 2004.

[32] Hassan A. Farhat. Digital Design and Computer Organization. CRC Press, Boca Raton, FL, 2004.

[33] Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. Embedded Computing: A VLIW Approach to Architecture, Compilers, and Tools. Morgan Kaufmann, San Francisco, CA, 2005.

[34] José Flich and Davide Bertozzi. Designing Network-on-Chip Architectures in the Nanoscale Era. Chapman & Hall/CRC Computational Science. CRC Press, Boca Raton, FL, 2011.

[35] Samuel H. Fuller and Lynette I. Millett. The future of computing performance: Game over or next level? Technical report, National Research Council, Washington, D.C., 2011.

[36] Fayez Gebali, Haytham Elmiligi, and Mohamed Watheq El-Kharashi. Networks-on-Chips: Theory and Practice. CRC Press, Boca Raton, FL, 2009.

[37] gem5 developers. The gem5 simulator system: A modular platform for computer system architecture research. Available online at: http://www.gem5.org/Main_Page; October 9, 2014 was the last accessed date, October 27 2014.

[38] Ran Giladi. Network Processors: Architecture, Programming, and Implementation. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, Burlington, MA, 2008.

[39] Matthias Gries and Kurt Keutzer. Building ASIPS: The Mescal Methodology. Springer Science+Business Media, Inc., New York, NY, 2005.

[40] Gary D. Hachtel and Fabio Somenzi. Logic Synthesis and Verification Algorithms. Springer Science+Business Media, LCC, New York, NY, 1996.

[41] Georg Hager and Gerhard Wellein. Introduction to High Performance Computing for Scientists and Engineers. Chapman & Hall/CRC Computational Science. CRC Press, Boca Raton, FL, 2011.

[42] Soha Hassoun and Tsutomu Sasao. Logic Synthesis and Verification. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Norwell, MA, 2002.

[43] Jörg Henkel and Sri Parameswaran. Designing Embedded Processors: A Low Power Perspective. Springer, Dordrecht, The Netherlands, 2007.

[44] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Waltham, MA, fifth edition, 2012.

[45] M. S. Hrishikesh, Doug Burger, Norman P. Jouppi, Stephen W. Keckler, Keith I. Farkas, and Premkishore Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In Proceedings of the 29^{th} Annual International Symposium on Computer Architecture (ISCA '02), pages 14–24, Anchorage, AK, May 25–29 2002. ACM Special Interest Group on Computer Architecture and IEEE Computer Society TCCA, IEEE Press.

[46] Michael Hübner and Jürgen Becker. Multiprocessor System-on-Chip: Hardware Design and Tool Integration. Springer Science+Business Media, LCC, New York, NY, 2011.

[47] Paolo Ienne and Rainer Leupers. Customizable Embedded Processors: Design Technologies and Applications. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, San Francisco, CA, 2007.

[48] Bruce Jacob. The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, 2009.

[49] Bruce Jacob, Spencer Ng, and David Wang. Memory Systems: Cache, DRAM, Disk. Elsevier Inc., Burlington, MA, 2008.

[50] Axel Jantsch and Hannu Tenhunen. Networks on Chip. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[51] Natalie Enright Jerger and Li-Shiuan Peh. On-Chip Networks. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, 2009.

[52] Ahmed Amine Jerraya and Wayne Wolf. Multiprocessor Systems-on-Chips. The Morgan Kaufmann Series in Systems on Silicon. Elsevier Inc., San Francisco, CA, 2005.

[53] Norman P. Jouppi and David W. Wall. Available instruction-level parallelism for superscalar and superpipelined machines. In Proceedings of the 3^{rd} International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), pages 272–282, Boston, MA, April 3–6 1989. Association for Computing Machinery, ACM Press.

[54] Sung-Mo (Steve) Kang and Yusuf Leblebici. CMOS Digital Integrated Circuits: Analysis and Design. McGraw-Hill, New York, NY, third edition, 2003.

[55] Kingshuk Karuri and Rainer Leupers. Application Analysis Tools for ASIP Design: Application Profiling and Instruction-set Customization. Springer Science+Business Media, LCC, New York, NY, 2011.

[56] Stefanos Kaxiras and Margaret Martonosi. Computer Architecture Techniques for Power-Efficiency. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, 2008.

[57] Stephen W. Keckler, Kunle Olukotun, and H. Peter Hofstee. Multicore Processors and Systems. Integrated Circuits and Systems. Springer Science+Business Media, LCC, New York, NY, 2009.

[58] Torsten Kempf, Gerd Ascheid, and Rainer Leupers. Multiprocessor Systems on Chip: Design Space Exploration. Springer Science+Business Media, LCC, New York, NY, 2011.

[59] Hyesoon Kim, Richard Vuduc, Sara Baghsorkhi, Jee Choi, and Wen-mei Hwu. Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU). Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, November 2012.

[60] Yongtae Kim. Energy Efficient and Error Resilient Neuromorphic Computing in VLSI. PhD thesis, Texas A&M University, College Station, TX, December 2013.

[61] Yongtae Kim, Yong Zhang, and Peng Li. A reconfigurable digital neuromorphic processor with memresistive synaptic crossbar for cognitive computing, 2014.

[62] David B. Kirk and Wen-mei W. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. Applications of GPU Computing. Morgan Kaufmann, Burlington, MA, 2010.

[63] Georgios Kornaros. Multi-Core Embedded Systems. CRC Press, Boca Raton, FL, 2010.

[64] Marco Lanzagorta, Stephen Bique, and Robert Rosenberg. Introduction to Reconfigurable Supercomputing. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, 2009.

[65] Panos C. Lekkas. Network Processors: Architectures, Protocols, and Platforms. McGraw-Hill Network Engineering. McGraw-Hill, New York, NY, 2003.

[66] Rainer Leupers and Olivier Temam. Processor and System-on-Chip Simulation. Springer Science+Business Media, LCC, New York, NY, 2010.

[67] John Levesque and Gene Wagenbreth. High Performance Computing: Programming and Applications. Chapman & Hall/CRC Computational Science. CRC Press, Boca Raton, FL, 2011.

[68] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey & Company, New York, NY, June 2011.

[69] Paul Merolla, John Arthur, Filipp Akopyan, Nabil Imam, Rajit Manohar, and Dharmendra S. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In Proceedings of the IEEE Custom Integrated Circuits Conference (CICC 2011), pages 1–4, San Jose, CA, September 19–21 2011. IEEE, IEEE Press.

[70] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing: Artificial Brains, 74(1–3):239–255, December 2010.

[71] Matteo Monchiero, Ramon Canal, and Antonio González. Power/performance/thermal design-space exploration for multicore architectures. IEEE Transactions on Parallel and Distributed Systems, 19(5):666–681, May 2008.

[72] Srinivasan Murali. Designing Reliable and Efficient Networks on Chips, volume 34 of Lecture Notes in Electrical Engineering. Springer Science+Business Media, B.V., Dordrecht, The Netherlands, 2009.

[73] Mario Nemirovsky and Dean M. Tullsen. Multithreading Architecture. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, January 2013.

[74] Chrysostomos Nicopoulos, Vijaykrishnan Narayanan, and Chita R. Das. Network-on-Chip Architectures: A Holistic Design Exploration, volume 45 of Lecture Notes in Electrical Engineering. Springer Science+Business Media, B.V., Dordrecht, The Netherlands, 2009.

[75] Jari Nurmi. Processor Design: System-on-Chip Computing for ASICs and FPGAs. Springer, Dordrecht, The Netherlands, 2007.

[76] Jari Nurmi, Hannu Tenhunen, Jouni Isoaho, and Axel Jantsch. Interconnect-Centric Design for Advanced SoC and NoC. Kluwer Academic Publishers, New York, NY, 2004.

[77] Umit Y. Ogras and Radu Marculescu. Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures, volume 184 of Lecture Notes in Electrical Engineering. Springer Science+Business Media New York, New York, NY, 2013.

[78] Vojin G. Oklobdzija and Ram K. Krishnamurthy. High-Performance Energy-Efficient Microprocessor Design. Integrated Circuits and Systems. Springer, Dordrecht, The Netherlands, 2006.

[79] Kunle Olukotun, Lance Hammond, and James Laudon. Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, 2007.

[80] Subbarao Palacharla. Complexity-Effective Superscalar Processors. PhD thesis, University of Wisconsin–Madison, Madison, WI, 1998.

[81] Behrooz Parhami. Computer Arithmetic: Algorithms and Hardware Designs. The Oxford Series in Electrical and Computer Engineering. Oxford University Press, New York, NY, second edition, 2010.

[82] PARSEC) benchmark suite contributors. Princeton application repository for shared-memory computers (parsec) benchmark suite. Available online at: http://parsec.cs.princeton.edu/; October 9, 2014 was the last accessed date, 2010.

[83] Sudeep Pasricha and Nikil Dutt. On-Chip Communication Architectures: System on Chip Interconnect. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, Burlington, MA, 2008.

[84] David A. Patterson and John L. Hennessy. Companion CD for computer organization and design: The hardware/software interface [CD-ROM]. Available: Companion CD, 2005.

[85] David A. Patterson and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, San Francisco, CA, third edition, 2005.

[86] David A. Patterson and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, Gurgaon, India, fourth edition, 2009.

[87] David A. Patterson and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, Waltham, MA, revised fourth edition, 2012.

[88] Richard Petersen. Linux: The Complete Reference. McGraw-Hill, New York, NY, sixth edition, 2008.

[89] Cong-Vinh Phan. Autonomic Networking-on-Chip: Bio-Inspired Specification, Development, and Verification. Embedded Multi-Core Systems. CRC Press, Boca Raton, FL, 2012.

[90] President's Council of Advisors on Science and Technology. Designing a digital future: Federally funded research and development in networking and information technology. Technical report, President's Council of Advisors on Science and Technology, Office of Science and Technology Policy, Executive Office of the President, Washington, D.C., January 2013.

[91] Roger S. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill, New York, NY, seventh edition, 2010.

[92] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolić. Digital Integrated Circuits: A Design Perspective. Prentice-Hall Electronics and VLSI Series. Pearson Education, Upper Saddle River, NJ, second edition, 2003.

[93] Kenneth H. Rosen, Douglas A. Host, Rachel Klee, James Farber, and Richard Rosinski. UNIX: The Complete Reference. Complete Reference Series. McGraw-Hill, New York, NY, second edition, 2007.

[94] Stephen R. Schach. Object-Oriented Software Engineering. McGraw-Hill, New York, NY, 2008.

[95] Louis Scheffer, Luciano Lavagno, and Grant Martin. EDA for IC Implementation, Circuit Design, and Process Technology, volume 2 of Electronic Design Automation for Integrated Circuits Handbook. CRC Press, Boca Raton, FL, 2006.

[96] Louis Scheffer, Luciano Lavagno, and Grant Martin. EDA for IC System Design, Verification, and Testing, volume 1 of Electronic Design Automation for Integrated Circuits Handbook. CRC Press, Boca Raton, FL, 2006.

[97] Oliver Schliebusch, Heinrich Meyr, and Rainer Leupers. Optimized ASIP Synthesis from Architecture Description Language Models. Springer, Dordrecht, The Netherlands, 2007.

[98] Roberto Sebastiani. Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation, 3:141–224, 2007.

[99] Jae-Sun Seo, Bernard Brezzo, Yong Liu, Benjamin D. Parker, Steven K. Esser, Robert K. Montoye, Bipin Rajendran, José A. Tierno, Leland Chang, Dharmendra S. Modha, and Daniel J. Friedman. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In Proceedings of the IEEE Custom Integrated Circuits Conference (CICC 2011), pages 1–4, San Jose, CA, September 19–21 2011. IEEE, IEEE Press.

[100] John Paul Shen and Mikko H. Lipasti. Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, New York, NY, 2005.

[101] Cristina Silvano, Marcello Lajolo, and Gianluca Palermo. Low Power Networks-on-Chip. Springer Science+Business Media, LCC, New York, NY, 2011.

[102] SimpleScalar LLC. Simplescalar 3.0. Available online at: http://www.simplescalar.com/tools.html, http://www.simplescalar.com/agreement.php3?simplesim-3v0e.tgz, and http://www.simplescalar.com/; October 9, 2014 was the last accessed date, 2004.

[103] Ian Sommerville. Software Engineering. Pearson Education, Harlow, Essex, England, eighth edition, 2007.

[104] Daniel J. Sorin, Mark D. Hill, and David A. Wood. A Primer on Memory Consistency and Cache Coherence. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, San Rafael, CA, November 2011.

[105] Sundarajan Sriram and Shuvra S. Bhattacharyya. Embedded Multiprocessors: Scheduling and Synchronization. CRC Press, Boca Raton, FL, second edition, 2009.

[106] William Stallings. Computer Organization and Architecture: Designing for Performance. Pearson Education, Upper Saddle River, NJ, ninth edition, 2013.

[107] Standard Performance Evaluation Corporation. SPEC CPU 2006. Available online at: http://spec.org/cpu2006/; October 27, 2014 was the last accessed date, June 10 2014.

[108] James E. Stine. Digital Computer Arithmetic Datapath Design Using Verilog HDL: CD-ROM Included. Kluwer Academic Publishers, New York, NY, 2004.

[109] Dominic Sweetman. See MIPS Run Linux. Morgan Kaufmann, San Francisco, CA, second edition, 2007.

[110] Andrew S. Tanenbaum. Structured Computer Organization. Prentice-Hall, Upper Saddle River, NJ, sixth edition, 2013.

[111] Elie K. Track, Tom Conte, Dan Allwood, David Atienza, Jonathan Candelaria, Erik DeBenedictis, Paolo Gargini, Glen Gulak, Bichlien Hoang, Subramanian (Subu) Iyer, Yung-Hsiang Lu, Scott Holmes, Alan M. Kadin, David Mountain, Oleg Mukhanov, Vojin G. Oklobdzijja, Angelos Stavrou, Bill Tonti, and Ian Young. 2^{nd} rebooting computing summit: Summary report. Technical report, Institute of Electrical and Electronics Engineers, Santa Cruz, CA, June 2014.

[112] Manish Vachharajani. Microarchitecture Modeling for Design-Space Exploration. PhD thesis, Princeton University, Princeton, NJ, November 2004.

[113] Ioannis E. Venetis and Guang R. Gao. The modified SPLASH-2 home page. Available online from Computer Architecture and Parallel Systems Laboratory (CAPSL), Department of Electrical & Computer Engineering, College of Engineering, University of Delaware at: http://www.capsl.udel.edu/splash/; November 29, 2014 was the last accessed date, July 3 2007.

[114] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting (Tim) Cheng. Electronic Design Automation: Synthesis, Verification, and Test. The Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann, Burlington, MA, 2009.

[115] Robin Wedewer. Rebooting computing summit – event summary. Technical report, The Wedewer Group, Washington, D.C., December 11–12 2013.

[116] Neil H. E. Weste and David Money Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Pearson Education, Boston, MA, fourth edition, 2011.

[117] Clint Witchalls. The Internet of Things business index: A quiet revolution – a report from The Economist Intelligence Unit. Available online at: http://www.arm.com/files/pdf/EIU_Internet_Business_Index_WEB.PDF; November 7, 2013 was the last accessed date, 2013.

[118] Wayne Wolf. High-Performance Embedded Computing: Architectures, Applications, and Methodologies. Morgan Kaufmann, San Francisco, CA, 2007.

[119] Wayne Wolf. Modern VLSI Design: IP-Based Design. Prentice Hall Modern Semiconductor Design. Prentice-Hall, Boston, MA, fourth edition, 2009.

[120] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. ACM SIGARCH Computer Architecture News, 23(1):20–24, March 1995.

[121] Laurence T. Yang and Minyi Guo. High-Performance Computing: Paradigm and Infrastructure. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, Hoboken, NJ, 2006.

[122] William Yurcik and Edward F. Gehringer. A survey of web resources for teaching computer architecture. In Proceedings of the 2002 Workshop on Computer Architecture Education (WCAE '02), Anchorage, AK, May 26 2002. ACM, ACM Press.