CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

ONE

COMPUTER OVERVIEW

1.0 INTRODUCTION

The general-purpose computer has assumed a dominant role in our world-wide society. From controlling the ignition of automobiles to maintaining the records of Olympic Games, computers are truly everywhere. In this book a one-semester course is provided for undergraduates that introduces the basic concepts of computers without focusing on distinctions between particular implementations such as mainframe, server, workstation, PC, or embedded controller. Instead the interest lies in conveying principles, with illustrations from specific processors.

In the modern world, the role of computers is multifaceted. They can store information such as a personnel database, record and transform information as with a word processor system, and generate information as when preparing tax tables. Computers must also have the ability to search for information on the World Wide Web.

The all-pervasive use of computers today is due to the fact that they are general purpose. That is, the computer hardware can be transformed by means of a stored program to be a vast number of different machines. An example of this power to become a special machine is found in word processing. For many years, Wang Laboratories was the dominant supplier of word processing machines based on a special-purpose processor with wired-in commands. Unable to see the revolution that was coming with the PC, Wang held to a losing product strategy and eventually was forced out of the business.

Another example of the general-purpose nature of computers is found in the electronic control of the automobile ignition. When electronic control was forced on the auto industry because of pollution problems, Ford took a different direction from that of Chrysler and General Motors. Chrysler and General Motors were relatively well off financially and opted to design special-purpose electronic controls for their ignition systems. Ford on the other hand was in severe financial difficulties and decided to use a microprocessor that cost a little more in production but did not require the development costs of the special-purpose circuits. With a microprocessor, Ford could, at relatively low cost, customize the controller for various engine configurations by changing the read-only memory (ROM) holding the program. Chrysler and General Motors, however, found that they had to have a unique controller for each configuration of

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

auto – a very expensive design burden. Microprocessor control, as first practiced by Ford, is now accepted by all and is the industry design standard.

A number of special-purpose computers were designed and built before the era of the general-purpose computer. These include the Babbage difference engine (circa 1835), the Anatasoff–Berry Computer (ABC) at Iowa State University in the late 1930s, and the Z3 of Konrad Zuse, also in the late 1930s. Other computers are the Colossus at Benchly Park (used for breaking German codes in World War II) and the ENIAC (which stands for electronic numerical integrater and computer, a plug-board-programmed machine at The University of Pennsylvania). These computers are discussed in Subsection 1.3.1.

Each of the early computers noted above was a one-of-a-kind machine. What was lacking was a design standard that would unify the basic architecture of a computer and allow the designers of future machines to simplify their designs around a common theme. This simplification is found in the von Neumann Model.

1.1 VON NEUMANN MODEL

The von Neumann model of computer architecture was first described in 1946 in the famous paper by Burks, Goldstein, and von Neumann (1946). A number of very early computers or computerlike devices had been built, starting with the work of Charles Babbage, but the simple structure of a stored-program computer was first described in this landmark paper. The authors pointed out that instructions and data consist of bits with no distinguishing characteristics. Thus a common memory can be used to store both instructions and data. The differentiation between these two is made by the accessing mechanism and context; the program counter accesses instructions while the effective address register accesses data. If by some chance, such as a programming error, instructions and data are exchanged in memory, the performance of the program is indeterminate. Before von Neumann posited the single address space architecture, a number of computers were built that had disjoint instruction and data memories. One of these machines was built by Howard Aiken at Harvard University, leading to this design style being called a Harvard architecture.[1]

A variation on the von Neumann architecture that is widely used for implementing calculators today is called a tagged architecture. With these machines, each data type in memory has an associated tag that describes the data type: instruction, floating-point value (engineering notation), integer, etc. When the calculator is commanded to add a floating-point number to an integer, the tags are compared; the integer is converted to floating point, the addition is performed, and the result is displayed in floating point. You can try this yourself with your scientific calculator.

All variations of the von Neumann that have been designed since 1946 confirm that the von Neumann architecture is classical and enduring. This architecture can be embellished but its underlying simplicity remains. In this section the von Neumann

[1] The von Neumann architecture is also known as a Princeton architecture, as compared with a Harvard architecture.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

architecture is described in terms of a set of nested state machines. Subsection 1.2.1 explores the details of the von Neumann architecture.

We should not underestimate the impact of the von Neumann architecture, which has been the unifying concept in all computer designs since 1950. This design permits an orderly design methodology and interface to the programmer. One can look at the description of any modern computer or microprocessor and immediately identify the major components: memory, central processing unit (CPU), control, and input/output (I/O).

The programmer interface with the von Neumann architecture is orderly. The programmer knows that the instructions will be executed one at a time and will be completed in the order issued. For concurrent processors, discussed in Chapter 6, order is not preserved, but as far as the programmer is concerned order is preserved.

A number of computer architectures that differ from the von Neumann architecture have been proposed over the years. However, the simplicity and the order of the von Neumann architecture have prevented these proposals from taking hold; none of these proposed machines has been built commercially.

State Machine Equivalent

A computer is defined as the combination of the memory, the processor, and the I/O system. Because of the centrality of memory, Chapter 4 discusses memory before Chapters 5 and 6 discuss the processor.

The three components of a computer can be viewed as a set of nested state machines. Fundamentally, the memory holds instructions and data. The instructions and the data flow to the logic, then the data (and in some designs the instructions) are modified by the processor logic and returned to the memory. This flow is represented as a state machine, shown in Figure 1.1.

The information in memory is called the process state. Inputs into the computer are routed to memory and become part of the process state. Outputs from the computer are provided from the process state in the memory.

The next level of abstraction is illustrated in Figure 1.2. The logic block of Figure 1.1 is replaced with another state machine. This second state machine has for its memory the processor registers. These registers, discussed in Chapter 3, include the program counter, general-purpose registers, and various dedicated registers. The logic consists of the arithmetic and logic unit (ALU) plus the logic required to support the interpretation of instructions.

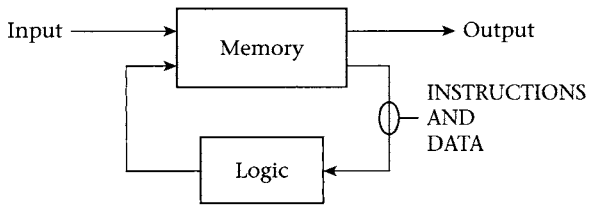


Figure 1.1 State machine

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

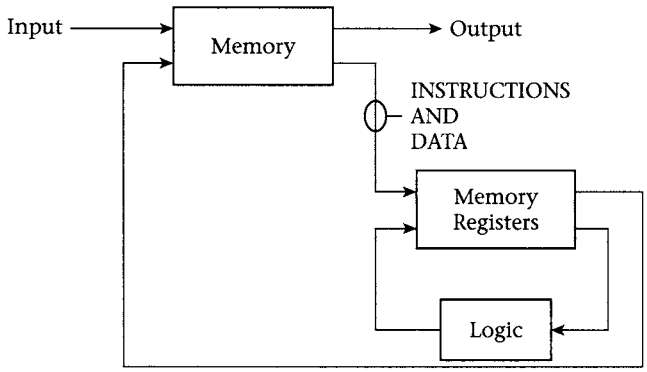


Figure 1.2 State machine II

The information contained in the registers is called the processor state. The processor state consists of (1) the information needed to interpret an instruction, and (2) information carried forward from instruction to instruction such as the program counter value and various tags and flags. When there is a processor context switch, it is the processor state that is saved, so the interrupted processor state can be restored.[2]

When microprogramming is discussed in Chapter 5, we will see that the logic block of Figure 1.2 can also be implemented as a state machine. For these implementations, there are three levels of state machine: process state, processor state, and micromachine processor state.

We now examine the major components of a computer, starting with the memory. As discussed in the preceding paragraphs, the memory is the space that holds the process state, consisting of instructions and data. The instruction space is not only for the program in execution but also for the operating system, compilers, interpreters, and other system software.

The processor reads instructions and data, processes the data, and returns results to memory, where the process state is updated. Thus a primary requirement for memory is that it be fast; that is, reads and writes must be accomplished with a small latency.

In addition, there are two conflicting requirements for memory: memory should be both very large and very fast. Memory cost is always a factor, with low cost being very desirable. These requirements lead to the concept of hierarchical memory. The memory closest to the processor is relatively small but is very fast and relatively expensive. The memory most distant from the processor is disk memory that is very slow but very low cost. Hierarchical memory systems have performance that approaches the fast memory while the cost approaches that of the low-cost disk memory. This characteristic is the result of the concept of locality, discussed in Chapter 4. Locality of programs and data results in a high probability that a request by the processor for either an instruction or a datum will be served in the memory closest to the processor.

The processor, sometimes called the CPU, is the realization of the logic and registers of Figure 1.2. This portion of the system fetches instructions, decodes these

[2] A context switch saves the processor state and restores a previously saved processor state.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

instructions, finds operands, performs the operation, and returns the result to memory. The complexity of the CPU is determined by (1) the complexity of the instruction set, and (2) the amount of hardware concurrency provided for performance enhancement.

As shown in Figure 1.2, a computer must have some method of moving input data and instructions into the memory system and moving results from the memory to the outside world. This movement is the responsibility of the I/O system. Input and output devices can have differing bandwidths and latency. For example, keyboards are low-bandwidth devices whereas color display units have high bandwidth. In between we find such devices as disks, modems, and scanners.

The control of I/O can take a number of forms. At one extreme, each transfer can be performed by the CPU. Fully autonomous systems, such as direct memory access (DMA), however, provide high-bandwidth transfers with little CPU involvement. I/O systems are discussed in Chapter 7.

The formal specification of a processor, its interaction with memory, and its I/O capabilities are found in its instruction set architecture (ISA). The ISA is the programmer's view of the computer. The details of how the ISA is implemented in hardware, details that affect performance, are known as the implementation of the ISA.

1.2 THE VON NEUMANN ARCHITECTURE

The von Neumann ISA is described in this section. Except for the I/O, this architecture is complete and represents a starting point for the discussion in the following chapters. The features found in this architecture can be found in any of today's architectures; thus a thorough understanding of the von Neumann architecture is a good starting point for a general study of computer architecture. This architecture, of which a number were actually built, is used in this book for a simple example rather than the presentation of a contrived example of a simple architecture. When a von Neumann computer was actually completed at Princeton University in 1952, it was named the Institute for Advanced Studies (IAS) computer.

The von Neumann architecture consists of three major subsystems: instruction processing, arithmetic unit, and memory, as shown in Figure 1.3. A key feature of this architecture is that instructions and data share the same address space. Thus there is one source of addresses, the instruction processing unit, to the memory. The output of the memory is routed to either the Instruction Processing Unit or the Arithmetic Unit

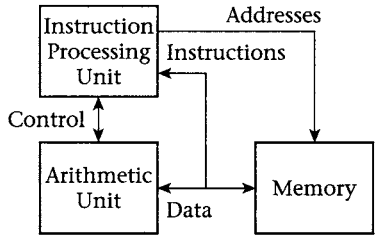


Figure 1.3 von Neumann architecture

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

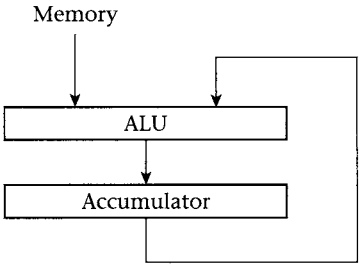


Figure 1.4 Accumulator local storage

depending upon whether an instruction or a datum is being fetched. A corollary to the key feature is that instructions can be processed as data. As will be discussed in later chapters, processing instructions as data can be viewed as either a blessing or a curse.

1.2.1 THE VON NEUMANN INSTRUCTION SET ARCHITECTURE

The von Neumann ISA is quite simple, having only 21 instructions. In fact, this ISA could be called an early reduced instruction set computer (RISC) processor.[3] As with any ISA, there are three components: addresses, data types, and operations. The taxonomy of these three components is developed further in Chapter 3; the three components of the von Neumann ISA are discussed below.

Addresses

The addresses of an ISA establish the architectural style – the organization of memory and how operands are referenced and results are stored. Being a simple ISA, there are only two memories addressed: the main memory and the accumulator.

The main memory of the von Neumann ISA is linear random access and is equivalent to the dynamic random-access memory (DRAM) found in today's processors. The technology of the 1940s restricted random-access memory (RAM) to very small sizes; thus the memory is addressed by a 12-bit direct address allocated to the 20-bit instructions.[4] There are no modifications to the address, such as base register relative or indexing. The formats of the instructions are described below in the subsection on data types.

Local storage in the processor is a single accumulator, as shown in Figure 1.4. An accumulator register receives results from the ALU that has two inputs, a datum from memory, and the datum held in the accumulator. Thus only a memory address is needed in the instruction as the accumulator is implicitly addressed.

Data Types

The von Neumann ISA has two data types: fractions and instructions. Instructions are considered to be a data type since the instructions can be operated on as data, a feature called self-modifying code. Today, the use of self-modifying code is considered

[3] An instruction set design posited by Van der Poel in 1956 has only one instruction.
[4] The Princeton IAS designers had so much difficulty with memory that only 1K words were installed with a 10-bit address.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

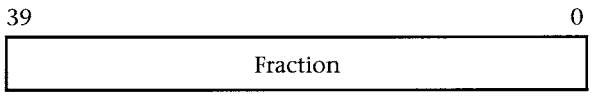
Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

to be poor programming practice. However, architectures such as the Intel x86 family must support this feature because of legacy software such as MSDOS.

Memory is organized with 4096 words with 40 bits per word; one fraction or two instructions are stored in one memory word.

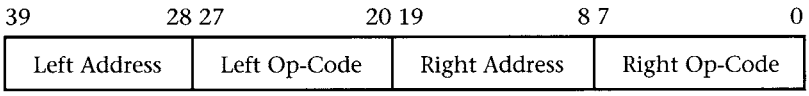
FRACTIONS

The 40-bit word is typed as a 2's complement fraction; the range is $-1 \leq f < +1$:



INSTRUCTIONS

Two 20-bit instructions are allocated to the 40-bit memory word. An 8-bit operation code, or op-code, and a 12-bit address are allocated to each of the instructions. Note that, with only 21 instructions, fewer op-code bits and more address bits could have been allocated. The direct memory address is allocated to the 12 most significant bits (MSBs) of each instruction. The address and the op-code pairs are referred to in terms of left and right:



Registers

A block diagram of the von Neumann computer is shown in Figure 1.5. Note that I/O connections are not shown. Although only sketchily described in the original paper on this architecture, I/O was added to all implementations of this design.

The von Neumann processor has seven registers that support the interpretation of the instructions fetched from memory. These registers and their functions are listed in Table 1.1. Note that two of the registers are explicitly addressed by the instructions and defined in the ISA (called architected registers) while the other six are not defined

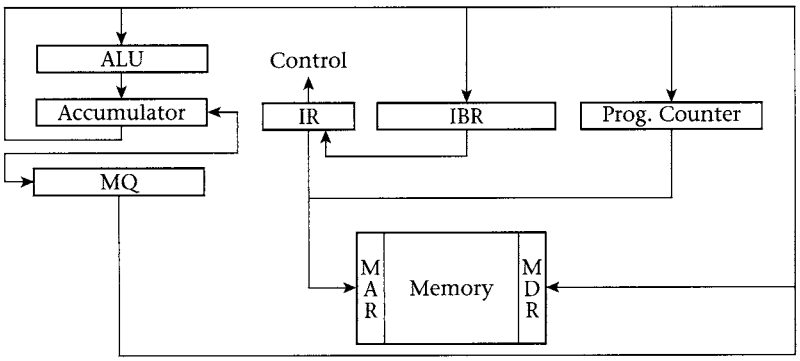


Figure 1.5 Block diagram of the von Neumann architecture: MQ, multiplier quotient register; IR, instruction register; IBR, instruction buffer register; MAR, memory address register; MDR, memory data register

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

TABLE 1.1 VON NEUMANN ISA REGISTERS

Name	Function
Architected Registers	
Accumulator, AC, 40 bits	Holds the output of the ALU after an arithmetic operation, a datum loaded from memory, the most-significant digits of a product, and the divisor for division.
Multiplier quotient register, MQ, 40 bits	Holds a temporary data value such as the multiplier, the least-significant bits of the product as multiplication proceeds, and the quotient from division.
Implemented Registers	
Program counter, PC, 12 bits*	Holds the pointer to memory. The PC contains the address of the instruction pair to be fetched next.
Instruction buffer register, IBR, 40 bits	Holds the instruction pair when fetched from the memory.
Instruction register, IR, 20 bits	Holds the active instruction while it is decoded in the control unit.
Memory address register, MAR, 12 bits	Holds the memory address while the memory is being cycled (read or write). The MAR receives input from the program counter for an instruction fetch and from the address field of an instruction for a datum read or write.
Memory data register, MDR, 40 bits	Holds the datum (instruction or data) for a memory read or write cycle.

* The program counter is a special case. The PC can be loaded with a value by a branch instruction, making it architected, but cannot be read and stored, making it implemented.

but are used by the control for moving bits during the execution of an instruction (called implemented registers).

Operations

The operations of the von Neumann ISA are of three types:

- moves between the accumulator, multiplier quotient register, and memory
- ALU operations such as add, subtract, multiply, and divide
- Unconditional and conditional branch instructions that redirect program flow.[5]

The von Neumann ISA consists of 21 instructions, shown in Table 1.2, which are sufficient to program any algorithm. However, the number of instructions that must

[5] Many computer historians credit the von Neumann ISA with the first use of conditional branching with a stored program computer. No prior computer possessed this feature and subprograms were incorporated as in-line code.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

TABLE 1.2 THE VON NEUMANN ISA

	Move Instructions	
1.	$AC \leftarrow MQ$	Move the number held in the MQ into the accumulator.
2.	$M(x) \leftarrow AC$	Move the number in the accumulator to location x in memory. The memory address x is found in the 12 least-significant bits of the instruction.
3.*	$M(x,28:39) \leftarrow AC(28:39)$	Replace the left-hand 12 bits of the left-hand instruction located at position x in the memory with the left-hand 12 bits in the accumulator.†
4.*	$M(x,8:19) \leftarrow AC(28:39)$	Replace the left-hand 12 bits of the right-hand instruction in location x in the memory with the left-hand 12 bits in the accumulator.
	ALU Instructions	
5.	$AC_c \leftarrow M(x)$	Clear the accumulator and add the number from location x in the memory.
6.	$AC \leftarrow AC_c - M(x)$	Clear the accumulator and subtract the number at location x in the memory.
7.	$AC \leftarrow AC_c + \lvert M(x)\rvert$	Clear the accumulator and add the absolute value of the number at location x in the memory.
8.	$AC \leftarrow AC_c - \lvert M(x)\rvert$	Clear the accumulator and subtract the absolute value of the number at location x in the memory.
9.	$AC \leftarrow AC + M(x)$	Add the number at location x in the memory into the accumulator.
10.	$AC \leftarrow AC - M(x)$	Subtract the number at location x in the memory from the accumulator.
11.	$AC \leftarrow AC + \lvert M(x)\rvert$	Add the absolute value of the number at location x in the memory to the accumulator.
12.	$AC \leftarrow AC - \lvert M(x)\rvert$	Subtract the absolute value of the number at location position x in the memory into the accumulator.
13.	$MQ_c \leftarrow M(x)$	Clear the MQ register and add the number at location x in the memory into it.
14.	$AC_c, MQ \leftarrow M(x) \times MQ$	Clear the accumulator and multiply the number at location x in the memory by the number in the MQ, placing the most-significant 39 bits of the answer in the accumulator and the least-significant 39 bits of the answer in the MQ.
15.	$MQ_c, AC \leftarrow AC \div M(x)$	Clear the register and divide the number in the accumulator by the number at location x of the memory, leaving the remainder in the accumulator and placing the quotient in MQ.
16.	$AC \leftarrow AC \times 2$	Multiply the number in the accumulator by 2.
17.	$AC \leftarrow AC \div 2$	Divide the number in the accumulator by 2.
	Control Instructions	
18.	Go to $M(x,2\ 0:39)$	Shift the control to the left-hand instruction of the pair in $M(x)$.
19.	Go to $M(x,0:19)$	Shift the control to the right-hand instruction of the pair in $M(x)$.
20.	If $AC \geq 0$, then $PC \leftarrow M(x,0:19)$	If the number in the accumulator is ≥ 0, go to the right-hand instruction in $M(x)$.
21.	If $AC \geq 0$, then $PC \leftarrow M(x,20:39)$	If the number in the accumulator is ≥ 0, go to the left-hand instruction in $M(x)$.

* These instructions move the address portion of an instruction between memory and the accumulator. These instructions are required to support address modification. Indexing, common today in all computer's ISAs, had not yet been invented.

† The notation $M(x,0:19)$ means the right-hand 20 bits of location $M(x)$; $M(x,20:39)$ means the left-hand 20 bits, and so on.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
0521657059 - Computer Architecture and Implementation
Harvey G. Cragon
Excerpt
More information

be executed is considerably greater than that required by more modern ISAs. The instructions are grouped into three groups: move, ALU, and control. This grouping is typical of all computers based on the von Neumann architecture. A more modern terminology, not the terminology of von Neumann, is used in Table 1.2.

1.2.2 INSTRUCTION INTERPRETATION CYCLE

Interpretation of an instruction proceeds in three steps or cycles. The instruction is fetched, decoded, and executed. These three steps are discussed in the following subsections.

Instruction Fetch

A partial flow chart for the instruction fetch cycle is shown in Figure 1.6. Because two instructions are fetched at once, the first step is to determine if a fetch from memory is required. This test is made by testing the least-significant bit (LSB) of the program counter. Thus, an instruction fetch from memory occurs only on every other state of the PC or if the previous instruction is a taken branch. The fetch from memory places a left (L) and a right (R) instruction in the instruction buffer register (IBR).

Instructions are executed, except for the case of a branch instruction, left, right, left, right, etc. For example, consider that an R instruction has just been completed.

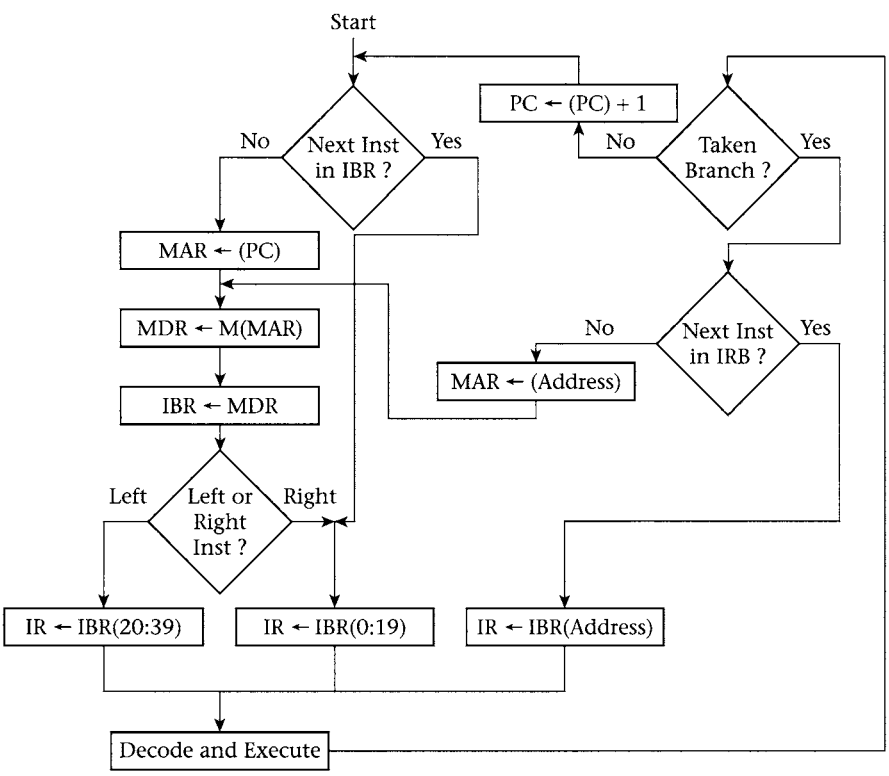


Figure 1.6 Instruction fetch cycle

© Cambridge University Press www.cambridge.org