1.3 The IAS Computer

The first generation of computers used vacuum tubes for digital logic elements and memory. A number of research and then commercial computers were built using vacuum tubes. For our purposes, it will be instructive to examine perhaps the most famous first-generation computer, known as the IAS computer. This example illustrates many of the fundamental concepts found in all computer systems.

A fundamental design approach first implemented in the IAS computer is known as the *stored-program concept*. This idea is usually attributed to the mathematician John von Neumann. Alan Turing developed the idea at about the same time. The first publication of the idea was in a 1945 proposal by von Neumann for a new computer, the EDVAC (Electronic Discrete Variable Computer).[2]

[2] The 1945 report on EDVAC is available at box.com/COA11e.

In 1946, von Neumann and his colleagues began the design of a new stored-program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies. The IAS computer, although not completed until 1952, is the prototype of all subsequent general-purpose computers.[3]

[3] A 1954 report [GOLD54] describes the implemented IAS machine and lists the final instruction set. It is available at box.com/COA11e.

Figure 1.6 shows the structure of the IAS computer (compare with **Figure 1.1**). It consists of

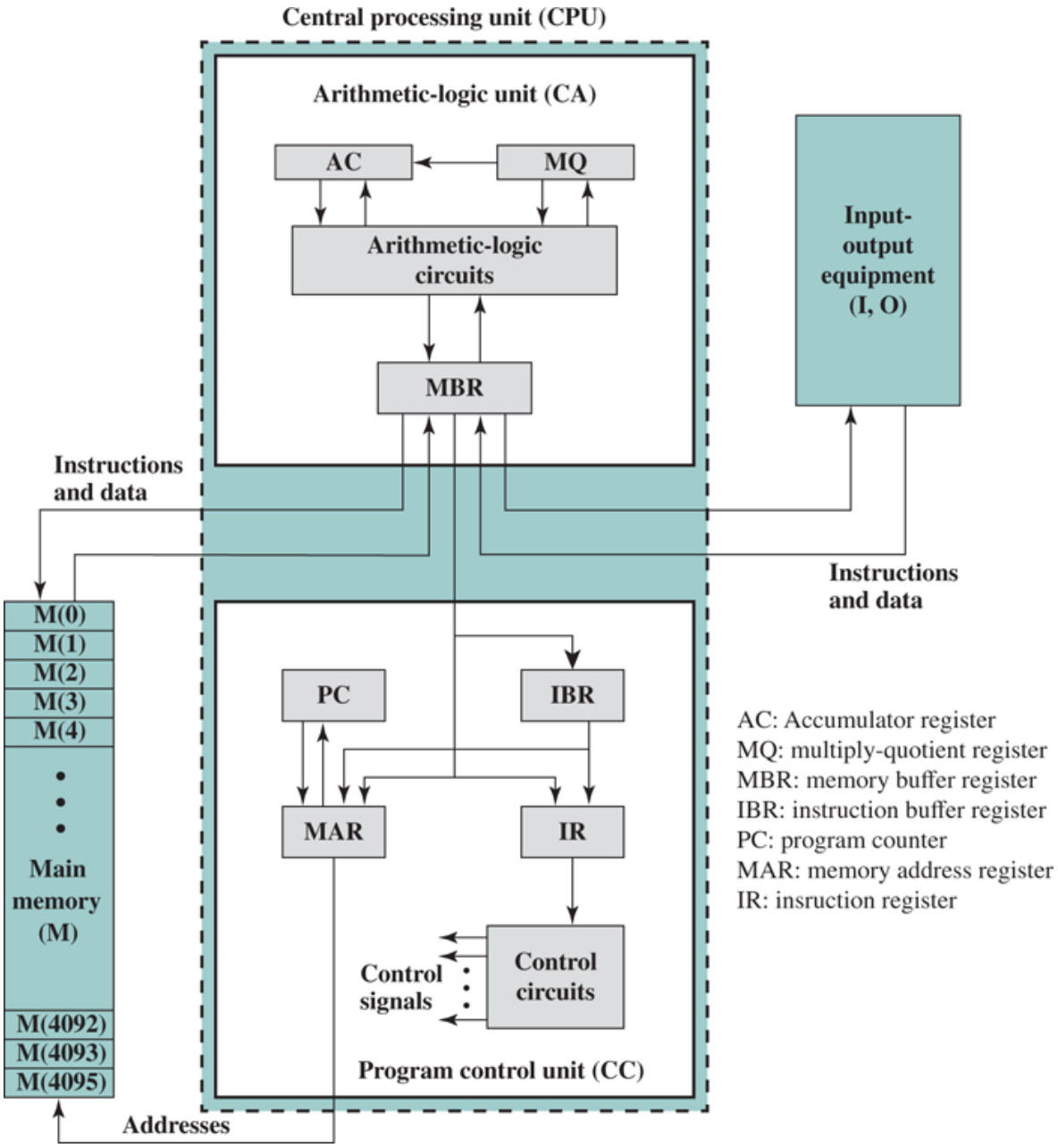

Figure 1.6 IAS Structure

- A **main memory** , which stores both data and instructions[4]

[4] In this book text, unless otherwise noted, the term *instruction* refers to a machine instruction that is directly interpreted and executed by the processor, in contrast to a statement in a high-level language, such as Ada or C++, which must first be compiled into a series of machine instructions before being executed.

- An **arithmetic and logic unit (ALU)** capable of operating on binary data
- A **control unit** , which interprets the instructions in memory and causes them to be executed
- **Input–output (I/O)** equipment operated by the control unit

This structure was outlined in von Neumann's earlier proposal, which is worth quoting in part at this

point [VONN45]:

> 2.2 **First:** Since the device is primarily a computer, it will have to perform the elementary operations of arithmetic most frequently. These are addition, subtraction, multiplication, and division. It is therefore reasonable that it should contain specialized organs for just these operations.
>
> It must be observed, however, that while this principle as such is probably sound, the specific way in which it is realized requires close scrutiny. At any rate a *central arithmetical* part of the device will probably have to exist, and this constitutes *the first specific part: CA*.
>
> 2.3 **Second:** The logical control of the device, that is, the proper sequencing of its operations, can be most efficiently carried out by a central control organ. If the device is to be *elastic*, that is, as nearly as possible *all purpose*, then a distinction must be made between the specific instructions given for and defining a particular problem, and the general control organs that see to it that these instructions—no matter what they are—are carried out. The former must be stored in some way; the latter are represented by definite operating parts of the device. By the *central control* we mean this latter function only, and the organs that perform it form *the second specific part: CC*.
>
> 2.4 **Third:** Any device that is to carry out long and complicated sequences of operations (specifically of calculations) must have a considerable memory . . .
>
> The instructions which govern a complicated problem may constitute considerable material, particularly so if the code is circumstantial (which it is in most arrangements). This material must be remembered.
>
> At any rate, the total *memory* constitutes *the third specific part of the device: M.*
>
> 2.6 The three specific parts CA, CC (together C), and M correspond to the *associative* neurons in the human nervous system. It remains to discuss the equivalents of the *sensory* or *afferent* and the *motor* or *efferent* neurons. These are the *input* and *output* organs of the device.
>
> The device must be endowed with the ability to maintain input and output (sensory and motor) contact with some specific medium of this type. The medium will be called the *outside recording medium of the device: R*.
>
> 2.7 **Fourth:** The device must have organs to transfer information from R into its specific parts C and M. These organs form its *input*, the *fourth specific part: I*. It will be seen that it is best to make all transfers from R (by I) into M and never directly from C.
>
> 2.8 **Fifth:** The device must have organs to transfer from its specific parts C and M into R. These organs form its *output*, *the fifth specific part: O*. It will be seen that it is again best to make all

> transfers from M (by O) into R, and never directly from C.

With rare exceptions, all of today's computers have this same general structure and function and are thus referred to as *von Neumann machines*. Thus, it is worthwhile at this point to describe briefly the operation of the IAS computer [BURK46, GOLD54]. Following [HAYE98], the terminology and notation of von Neumann are changed in the following to conform more closely to modern usage; the examples accompanying this discussion are based on that latter text.

The memory of the IAS consists of 4,096 storage locations, called *words*, of 40 binary digits (bits) each.[5] Both data and instructions are stored there. Numbers are represented in binary form, and each instruction is a binary code. **Figure 1.7** illustrates these formats. Each number is represented by a sign bit and a 39-bit value. A word may alternatively contain two 20-bit instructions, with each instruction consisting of an 8-bit operation code (opcode) specifying the operation to be performed and a 12-bit address designating one of the words in memory (numbered from 0 to 999).

[5] There is no universal definition of the term *word*. In general, a word is an ordered set of bytes or bits that is the normal unit in which information may be stored, transmitted, or operated on within a given computer. Typically, if a processor has a fixed-length instruction set, then the instruction length equals the word length.

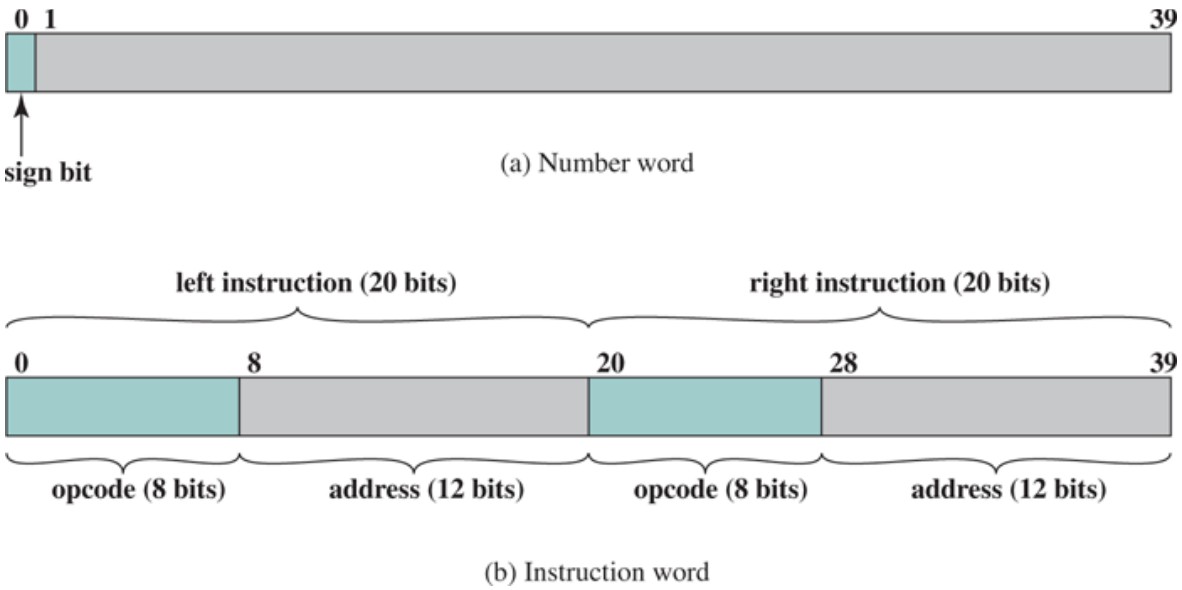

Figure 1.7 IAS Memory Formats

The control unit operates the IAS by fetching instructions from memory and executing them one at a time. We explain these operations with reference to **Figure 1.6**. This figure reveals that both the control unit and the ALU contain storage locations, called *registers*, defined as follows:

- **Memory buffer register (MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
- **Memory address register (MAR):** Specifies the address in memory of the word to be written from or read into the MBR.
- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.
- **Instruction buffer register (IBR):** Employed to hold temporarily the right-hand instruction from a word in memory.

- **Program counter (PC):** Contains the address of the next instruction pair to be fetched from memory.
- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ.

The IAS operates by repetitively performing an *instruction cycle*, as shown in **Figure 1.8**. Each instruction cycle consists of two subcycles. During the *fetch cycle*, the opcode of the next instruction is loaded into the IR and the address portion is loaded into the MAR. This instruction may be taken from the IBR, or it can be obtained from memory by loading a word into the MBR, and then down to the IBR, IR, and MAR.

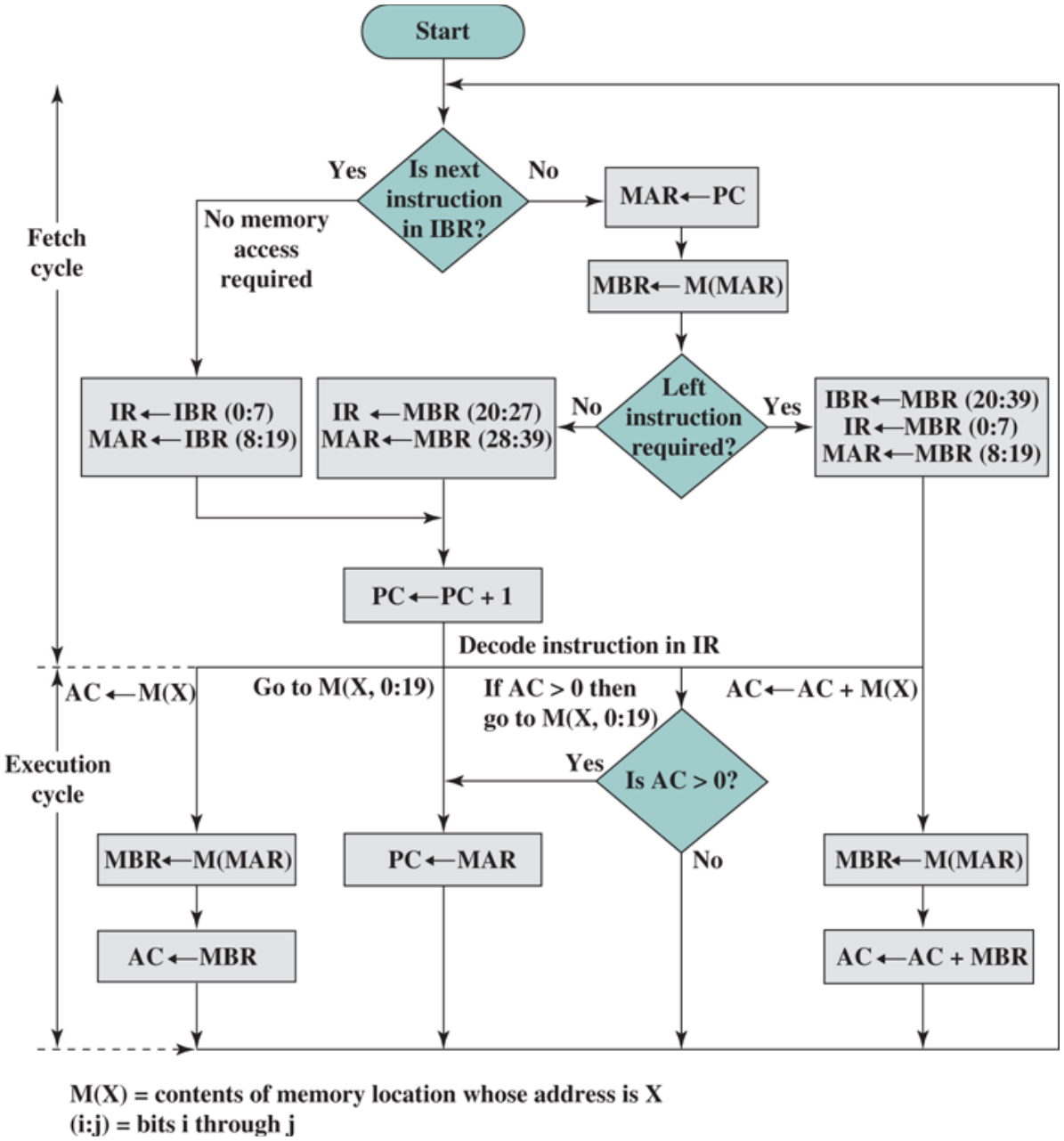

Figure 1.8 Partial Flowchart of IAS Operation

Why the indirection? These operations are controlled by electronic circuitry and result in the use of data paths. To simplify the electronics, there is only one register that is used to specify the address in memory for a read or write and only one register used for the source or destination.

Once the opcode is in the IR, the *execute cycle* is performed. Control circuitry interprets the opcode and executes the instruction by sending out the appropriate control signals to cause data to be moved or an operation to be performed by the ALU.

The IAS computer had a total of 21 instructions, which are listed in **Table 1.1**. These can be grouped as follows:

Table 1.1 The IAS Instruction Set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD –M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD –|M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	$\text{JUMP} + \text{M}(\text{X}, 0:19)$	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	$\text{JUMP} + \text{M}(\text{X}, 20:39)$	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC

	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; that is, shift left one bit position
	00010101	RSH	Divide accumulator by 2; that is, shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

- **Data transfer:** Move data between memory and ALU registers or between two ALU registers.
- **Unconditional branch:** Normally, the control unit executes instructions in sequence from memory. This sequence can be changed by a branch instruction, which facilitates repetitive operations.
- **Conditional branch:** The branch can be made dependent on a condition, thus allowing decision points.
- **Arithmetic:** Operations performed by the ALU.
- **Address modify:** Permits addresses to be computed in the ALU and then inserted into instructions stored in memory. This allows a program considerable addressing flexibility.

Table 1.1 presents instructions (excluding I/O instructions) in a symbolic, easy-to-read form. In binary form, each instruction must conform to the format of **Figure 1.7**b. The opcode portion (first 8 bits) specifies which of the 21 instructions is to be executed. The address portion (remaining 12 bits) specifies which of the 4,096 memory locations is to be involved in the execution of the instruction.

Figure 1.8 shows several examples of instruction execution by the control unit. Note that each operation requires several steps, some of which are quite elaborate. The multiplication operation requires 39 suboperations, one for each bit position except that of the sign bit.