16-bit Microprocessors

IAN R. WHITWORTH
Royal Military College of Science, Shrivenham

COLLINS
8 Grafton Street, London W1

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Granada Publishing 1984 (Previous ISBN 0-246-11572-6)
Reprinted with amendments by
Collins Professional and Technical Books 1985

British Library Cataloguing in Publication Data
Whitworth, Ian R.
16-bit microprocessors.
1. Microprocessors
I. Title
001.64'04 QA76.5

ISBN 0-00-383113 2

Typeset by Columns of Reading
Printed and bound in Great Britain by
Richard Clay (The Chaucer Press) Ltd, Bungay, Suffolk

Preface

The world of the microprocessor is becoming increasingly complex, with new developments being announced every day. Despite this complexity, the microprocessor is becoming ever more pervasive, entering, in one form or another, all aspects of the lives of those fortunate to be resident in technologically advanced countries. Even underdeveloped nations are feeling the impact of the microprocessor. In the short space of twelve years, semiconductor companies have evolved through several generations of microprocessor designs; at first, their devices were looked upon as logic systems replacements, or microcontrollers, but now their most sophisticated products rival the upper end of the minicomputer spectrum in speed and power.

At the time of writing, nearly all microprocessor-based products have been designed around 4-bit or 8-bit microprocessor central processor units (CPUs), or single-chip computers with on-chip memories and interfaces. These devices have matured into well-proven designs, backed-up by a wealth of knowledge and applications experience. From the point of view of the professional engineer, a knowledge of 8-bit microprocessors – hardware and systems design, software design and maintenance, and applications – is almost mandatory. Indeed, a recently-graduated engineer is likely to have used 8-bit microprocessors in practical work during his College or University courses, whilst many older practising engineers will have become familiar with them through manufacturers' training courses, followed by the practical experience of designing devices into useful products. For anyone wanting to become familiar with the 8-bit microprocessor, there are numerous elementary textbooks available which will serve as a good introduction to the field. Accordingly, a knowledge of 8-bit microprocessors is assumed, and in the interests of space, this book does not start from scratch. A 'revision chapter', Chapter 1, serves to remind the reader of the concepts and terminology of the 8-bit processor. The rest of the book is devoted to the new, and perhaps unfamiliar, ideas of the 16-bit processor, and other relevant developments.

The 16-bit microprocessor is much more than an 8-bit design with twice the word length. Although some of the early 16-bit processors did not offer much more than the speed advantage gained by their word length, the modern processor is completely different from its 8-bit counterpart. Modern 16-bit designs have attempted to ease the systems designer's task by providing instruction sets of considerable sophistication, operating system support in hardware, good

support of high-level languages, and ease of interfacing to the sort of bus structure which will form the basis of a multiple-processor installation. Whereas the first 8-bit designs embodied the view of what was needed in a processor from a hardware logic designer's standpoint, the modern 16-bit designs reflect the importance of software in overall systems design, and incorporate features requested by the systems designers who had soon uncovered the limitations of the 8-bit designs. Whilst many digital design engineers found the transition from logic design to 8-bit system design easy because of the similarities, the transition to 16-bit design may be thought more difficult, since the territory is less familiar. 16-bit terminology may seem strange, with its references to 'exceptions', 'privilege levels', 'memory management', 'virtual memory', 'coprocessors', 'semaphores', and many other new terms. The main objective of this book is to explore these new features, and give the reader an idea of how they work, how they are applied, and of their advantages and limitations. An appreciation is also given of the communications facilities which will become increasingly important as the new field of information technology expands, and with it the 16-bit processor, built into all types of information-processing equipment.

The author is indebted to various associates for their help and encouragement during preparation of the book. In particular, acknowledgement should be accorded to Professor C J Harris, Head of E and EE Department, RMCS, and Professor P C J Hill, Head of Electronics Branch, RMCS, for their tolerance and encouragement, to Graham Turner, for many helpful discussions, to Mrs A Hare, for typing parts of the manuscript, and to Dr F Hartley, Acting Dean, RMCS, for giving permission to publish this work.

Contents

CHAPTER 1
Introduction

The microprocessor, since its development in the early 1970s, has revolutionised not only electronics, but also many other fields, manufacturing industry, and even leisure and domestic products. The speed of evolution has been breath-taking; in little over a decade, the microprocessor has passed through several generations of devices, starting with the 4-bit 4004, and standing now on the threshold of minicomputer and mainframe domains. No semiconductor manufacturer with a product line encompassing digital integrated circuits has dared not compete with a microprocessor of his own, or a second-source device supporting another's design. The result has been to create a wide choice, in the current industry-standard 8-bit microprocessor field, and fierce competition to produce the next generation, 16-bit microprocessor which will attain industry-standard status. A rough chronology of microprocessor evolution might run:

1971	Introduction of first 4-bit microprocessor.
1972	First 8-bit microprocessor.
1973-5	Introduction of current industry-standard 8-bit CPUs and interfaces, ultraviolet-erasable PROM, 4K dynamic RAMs.
1976-7	First 8-bit single-chip computers.
1978-80	Introduction of modern 16-bit microprocessors, 16K static RAM, dynamic RAM to 64K, larger EPROM.
1980-date	Announcement of 32-bit microprocessor, second generation 8-bit microcontrollers, some second generation 16-bit microprocessors, electrically erasable PROM (E^2PROM).

Many engineers have become familiar with 8-bit microprocessors, either general purpose CPUs or single-chip microcontrollers, making the transition from electronic engineering to microprocessor applications fairly readily, assimilating the hardware techniques fairly readily, and perhaps taking a little longer to become fluent with (in most cases) assembly-code software. The degree of support provided by the semiconductor manufacturers for their devices, somewhat limited at first, has grown, with a vast investment in development system design and in software. As industry has woken up to the possibilities afforded by the 8-bit microprocessor, the average engineer is nowadays likely to have a good working knowledge of 8-bit microprocessors, but many feel overawed by the 'new' 16-bit CPUs, with their apparent complexity, and unfamiliar features. The problems of making the transition from 8-bit to 16-bit microprocessor applications are not trivial. Many new concepts in hardware, in software, in languages,

in operating systems, and in networks, are necessary, and it is the intention of this book to introduce them.

The rest of this chapter will be devoted to a description of typical 8-bit microprocessors, their support devices, and their software, partly to refresh the reader's knowledge, and partly for comparison with the 16-bit systems which form the bulk of the book.

1.1 8-bit microprocessor hardware

The 8-bit microprocessor field is characterised by its relative uniformity – although many 8-bit devices are available, they differ in speed and in some features, but, with only one or two exceptions, they have similar structure and philosophy. Certainly the 'industry-standard' microprocessors are all register oriented, all have the same addressing range, and all have broadly similar instruction sets. The block diagram of a typical 8-bit microprocessor is shown in fig. 1.1.

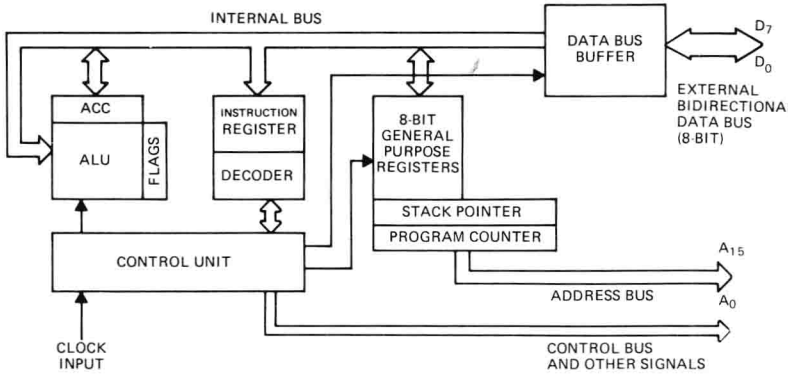

Fig. 1.1 8-bit CPU architecture.

An 8-bit arithmetic-and-logic unit (ALU) operates upon arguments held in programmer-accessible registers or accummulators, returning a result in a register and setting single-bit flags grouped into an 8-bit condition code register, in response to the result of an arithmetic or logical instruction. Timing is performed by a control unit driven by an externally-generated clock signal; the control unit takes signals from an instruction decoder, linked to an 8-bit instruction register. A block of programmer-accessible general-purpose registers for holding data and operands may be available. The microprocessor will certainly possess some 16-bit registers with dedicated functions, in particular, a program counter (or instruction pointer), automatically incremented by the control unit so as to keep track of instruction operation codes, a stack pointer, used as an address register to support the stack, a data structure in read-write memory which is controlled as a last-in, first-out (LIFO) buffer, indirect address registers, and possibly index registers. All these functional blocks communicate using an internal parallel

8-bit data bus, shared using time-division multiplexing.

The signals available at the pins of the microprocessor, which is usually in a 40-pin package, generally consist of separate parallel address and data buses, of width 16 bits and 8 bits, respectively. The 16-bit address bus width allows an addressing range of 64 kbytes, and conveniently, because an address is twice the width of a data word, any address values held in memory will, of course, occupy two bytes. To control data transfers over the bus, a set of control signals are generated by the CPU control unit. The bus control usually exercised is for *synchronous* transfers, controlled in timing exclusively by signals derived from the CPU system clock. Data transfers over the bus require two discrete pieces of information: directional information (transfers from memory or interfaces to the CPU are considered read operations, those in the reverse direction are write operations); and timing or 'strobe' information (to control the time when an addressed location or device should place its data on the bus, or accept data from the bus). Microprocessors may use bus control signals which combine direction and timing information, or may use discrete direction and timing signals. The two possibilities are illustrated in fig. 1.2. Those used by the general class of processors allied to the 8080, for both read and write operations, are shown in fig. 1.2(a). A read operation starts with the CPU issuing the address of the memory or input-output location, followed by the assertion of the $\overline{RD}$ control line. In response to the $\overline{RD}$ signal, the device will place data on the data bus, where it will be accepted by the CPU, followed by the release of $\overline{RD}$ and the removal of the address. The access time of the device must fit in with the CPU bus timing. The write operation starts, like the read, with the CPU issuing an address, followed by the CPU placing data on the data bus. When address and data are stable, the CPU asserts $\overline{WR}$, which the addressed device uses to latch the data from the bus.

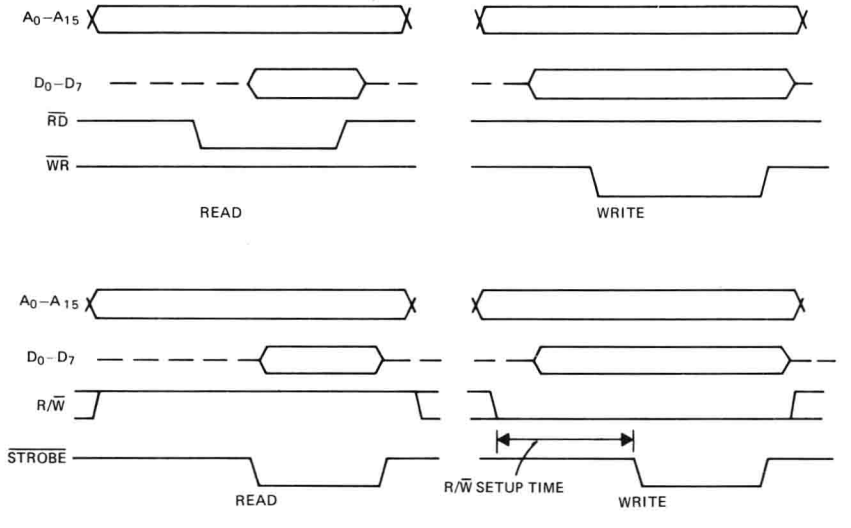

Fig. 1.2 (a) 8080-type bus control, (b) 6800-type bus control.

The second class of control signals, used by 6800-type microprocessors, is shown in fig. 1.2(b). The read operation starts with the CPU issuing an address, and simultaneously taking the R/$\overline{W}$ control signal high. When the address and R/$\overline{W}$ signals are stable, the CPU asserts the strobe signal E. The addressed device responds to the assertion of E by placing its data on the data bus, to be accepted by the CPU. The write operation starts with the CPU issuing an address, and simultaneously asserting the R/$\overline{W}$ line low and placing data on the data bus, followed by E, once address, data and R/$\overline{W}$ signals are stable. E may be used as a strobe signal by the addressed device, to latch data from the bus. For the 8080 class of microprocessors, any device must respond to directional information at the same time as it is responding to the timing strobe signal, whereas for the 6800 class, the directional information (R/$\overline{W}$) 'set-up' time with respect to the timing strobe signal is the same as that for the address information.

As well as these synchronous bus control signals, a negative-acknowledgement signal, READY (8080) or $\overline{\text{WAIT}}$ (Z80), may be available to allow memories or interface devices with longer access time than that implied by the CPU bus cycle timing. This signal is not a handshaking signal, since it is only asserted when the addressed device is unable to respond; no positive acknowledgement that a device has been able to respond in time is provided. The usual CPU response to this READY or $\overline{\text{WAIT}}$ signal is to insert 'idle' or wait state clock cycles into the read or write bus cycle, keeping address and control signals stable, and effectively stretching the bus cycle by an integral number of CPU clock cycles until the READY or $\overline{\text{WAIT}}$ signal is removed. To distinguish between memory and input-output operations, two alternatives are possible: the first, and conceptually possibly the 'cleanest', is to make no distinction at all, so that all input-output devices must occupy memory address space, and respond to all bus cycles generated by memory reference instructions. This memory-mapped input-output will also imply that the CPU instruction set need not include explicit input-output instructions. The second style of handling input-output is the provision of explicit input-output control signals, either IO/$\overline{M}$ (8085) or separate $\overline{\text{IORD}}$, $\overline{\text{IOW}}$ (8080). In this case, input-output devices have their own address space (I/O address space), distinguished from memory address space by the different control signals. I/O addresses can thus overlap memory addresses, and I/O devices require explicit instructions, usually given mnemonics IN and OUT.

All operations of the CPU are controlled by the CPU clock, and each elementary bus cycle or *machine cycle* (read, write, I/O write, instruction fetch, etc.) consists of a number of clock cycles. Each instruction consists of at least one machine cycle (instruction fetch), with memory reference instructions taking several machine cycles (instruction fetch, data read or write, etc.). Microprocessors such as the Z80 and 8080 take several clock cycles to perform each machine cycle, whereas in 6800-style microprocessors, clock cycles and machine cycles are much the same (a memory read, for example, takes just one clock cycle). To illustrate the sort of timing involved, a typical instruction cycle for the Z80 is shown in fig. 1.3 for an output (OUT) instruction. This takes three machine cycles (instruction fetch, read I/O port address [8-bit] from memory location following the location containing the OUT opcode [operation code],

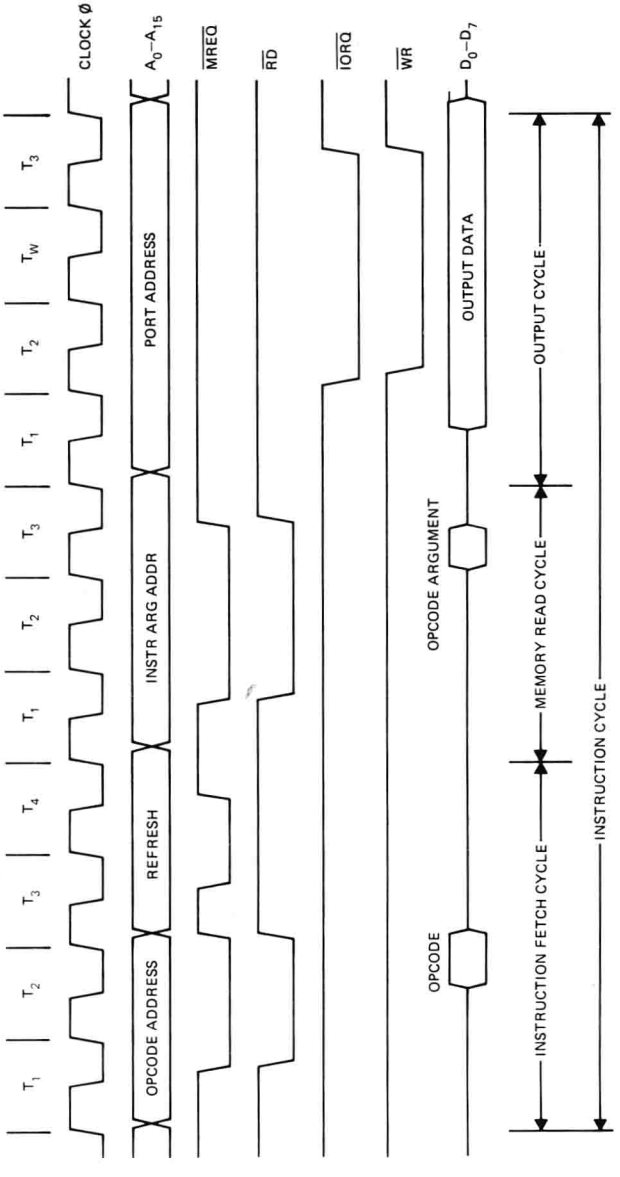

Fig. 1.3 Z80 output instruction execution.

and transfer the contents of the A [accumulator] register to the device). Note that this particular OUT instruction automatically inserts a WAIT state into the write machine cycle, a feature included so that slightly slower devices can be used without any external $\overline{WAIT}$ signal generation logic. This automatic stretching of any I/O read or write cycles is unique to the Z80 microprocessor. In many 8-bit microprocessors, a hardware signal (possibly part of a coded status signal multiplexed with others) is provided to identify the instruction fetch cycle. It may be used for emulation, trace and debugging, and for special bus cycles (e.g. interrupt acknowledgement).

Bus request and grant signals are relatively simple and centred around use of the bus for direct memory access (DMA). Typically, a HOLD or $\overline{BUSREQ}$ signal from a device (such as a DMA controller) will cause the CPU to complete its current machine cycle (or maybe instruction cycle) and to suspend its operation by entering an idle state where it simply keeps any internal data and status information refreshed. As soon as the CPU has completed the cycle, the external buses (address, data and control) may be relinquished, and floated to their tristate high-impedance mode (all CPU internal bus drivers have three logic states: the usual '0', approximately 0 V, '1', approximately 2.4 V minimum for a 5 V CPU, and an 'off' state, where the bus is not driven at all, and the only load on it is driver leakage current). Once the buses have achieved their high-impedance state, the CPU can issue an acknowledgement signal HOLDA, or $\overline{BUSAK}$, which may be used by the requesting device to gain control of the bus. When a device requesting the bus has completed its data transfers, it can release HOLD or $\overline{BUSREQ}$, and the CPU will regain control of its buses, continuing operation where it left off. This style of bus control is well suited to the requirements of a direct memory access (DMA) controller, which will transfer data to or from memory without CPU intervention, but less suited to multiprocessor shared-bus operation.

Other signals commonly available on an 8-bit CPU are interrupt inputs, which allow a logic signal which is not synchronised with the CPU clock to communicate with the synchronous logic of the CPU. Interrupts may be *maskable*, that is, they may be prevented from occurring by software control, programming a bit in a status or interrupt 'mask' register, or non-maskable, where they cannot be prevented from occurring. A separate hardware input for each type is usual. When a signal occurs on an enabled interrupt input, the CPU will latch it during a normal instruction cycle, so that the signal is remembered, and at the end of the instruction cycle a 'context switch' takes place. The CPU will stop executing its current program, and will begin executing a different program which is specific to the interrupt (and the device causing the interrupt). Usually, the CPU will be returned by software to where it left off in the original interrupted program when this special 'interrupt service routine' is complete, so some means of preserving and restoring the program counter value associated with the interrupted program are necessary. The virtually universal way of achieving this is by using the microprocessor stack. When an interrupt occurs, it is acknowledged, and during the acknowledgement process (which may invoke a read cycle on the data bus immediately following the instruction cycle during which the interrupt

occurred) an interrupt code or 'vector' may be acquired. The CPU will then 'push' the program counter onto the stack (writing it in two bytes using the stack pointer as an address pointer, decremented between successive 8-bit writes) which grows from high to low addresses. Once the program counter contents have been saved in this way, it can be loaded with a new value, which points to the start of the interrupt service routine. This address value may be acquired in a number of ways:

(a) It may be the contents of an interrupt location, used as an indirect address location.
(b) It may be the address of the interrupt location itself, where the service routine (or unconditional jump to the service routine) must be placed.
(c) The interrupt input may be 'vectored', that is, the interrupting device will supply, during the interrupt acknowledge cycle, a number (or vector) which identifies it. The address of the interrupt location may be derived directly from the vector (the 8080, for example, has eight interrupt locations, each identified by a 3-bit vector N, and located at an absolute address 8*N) or may be used as an index to a table of interrupt routine addresses. Alternatively, the 8080 allows a 'CALL' instruction to be generated directly by the interrupting hardware, using three successive interrupt acknowledge cycles. The subroutine call may be to an address anywhere in the 64K memory address space of the CPU.

When the interrupt service routine has completed its execution, a RET statement will cause a return to the interrupted program, automatically 'POPping' the saved program counter value off the top of the stack (incrementing the stack pointer register between POPs) into the program counter register. All 8-bit microprocessors are provided with a non-maskable RESET input, which causes the CPU to start operation from a fixed location (usually absolute memory address 0) with an instruction fetch from that location, when RESET is released.

Various 8-bit microprocessors have additional features unique to their own family of microprocessors: multiplexed data bus and lower byte of address bus, with an address strobe signal ALE provided for off-chip demultiplexing (8085); separate tristate bus control (6800); software-testable single-bit inputs (8085); interrupts vectoring to single fixed locations (8085); refresh signals for dynamic memory (see chapter 5, section 5.3) $\overline{\text{RFRSH}}$, indicating, with refresh address, that dynamic memory can be refreshed in a distributed refresh manner during the instruction decode part of an instruction fetch cycle (Z80). This description, however, covers most of the general 8-bit CPU features.

1.2 8-bit microprocessor machine code software

The 8-bit microprocessor is limited in its instruction set compared with an average minicomputer. Its arithmetic is performed as 8-bit, two's complement binary, with single-bit flag registers set according to the results of an arithmetic or logical instruction. Arithmetic instructions are limited in scope too, with no hardware multiply-divide unit on-chip. Addressing modes and memory reference

instructions reflect the fact that in most embedded applications (where the microprocessor is built into equipment and runs completely autonomously), program code will be in read-only memory (ROM), and changeable data in read-write random-access memory (RAM). Little provision is made for operating system support, error checking, or regularity in instruction sets or in internal registers to support those instruction sets.

1.2.1 8-BIT PROGRAMMING MODELS

A programming model of a microprocessor is one which shows none of the hardware detail, just the registers of the CPU which are accessible by the programmer. Typical programming models are shown in fig. 1.4. Arithmetic and logical instructions may be confined to one or more accumulator registers (A in fig. 1.4(a), the 8080/8085, A and B in fig. 1.4(b), the 6800). The other general-purpose registers may be used as address pointers (in pairs) as 8-bit or 16-bit counter registers (e.g. iteration counters), or for holding data. An index register (X in fig. 1.4(b)) is used, when its contents are combined with address information, to access arrays or tables of data by referring to an entry by its position in the array. The program counter (PC) is a 16-bit register which keeps track of the instruction stream of the program, and is automatically incremented during the last part of an instruction fetch cycle, ready to point to the next location. The stack pointer has already been commented upon earlier in this chapter. The flag register, or condition code register, contains a number of 1-bit flags, set automatically by the CPU in response to the result of an arithmetic or logical instruction (including perhaps increment and decrement); these are typically a selection from:

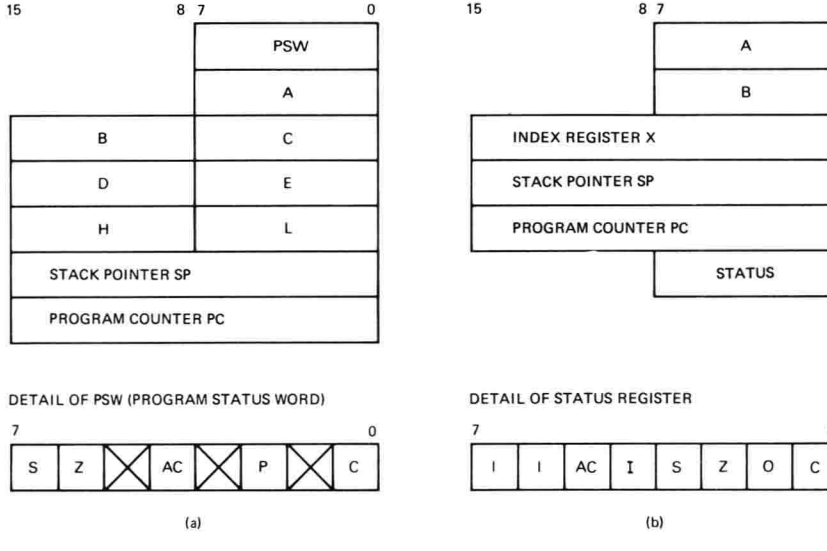

Fig. 1.4 8080, 6800 register models.

S sign (two's complement)
Z zero
CY carry
V overflow (arithmetic)
AC auxiliary (binary-coded decimal, BCD) carry, or half carry. Set when a carry, caused by a result greater than nine, occurs when two BCD numbers are used in an instruction execution. Used to restore a BCD result, represented as two 4-bit BCD numbers packed into an 8-bit word
P parity

1.2.2 ADDRESSING MODES

The 8-bit microprocessor addressing modes are appropriate to devices intended for applications which are relatively simple from a computing point of view. An 8-bit CPU cannot be considered an effective CPU for numerically intensive or multiuser applications (although 8-bit microprocessors have been used for both), and, as a consequence, addressing modes are unsophisticated. Usually a single addressing mode is used on its own, and not combined with others to form complex modes. In the context of addressing modes, the *effective address* is usually taken to mean the physical address formed as a result of the address computation implied by the mode. Typical modes are:

(a) Register: The operand is held in a register, identified by a field in the instruction operation code (opcode).
(b) Direct (or absolute): An absolute address is specified in the program (usually as two bytes following the opcode).
(c) Indirect: The effective address is the contents of a specified register or memory location.
(d) Indexed: The effective address is the sum of the contents of an index register (usually 16-bit register) and an offset (usually 8 bits) specified following the opcode.
(e) Relative (to PC): The effective address is the sum of the contents of the program counter (PC) register and (usually) an 8-bit signed two's complement offset, giving a range of addressing from −125 bytes to +127 bytes relative to PC, for the 6800. Usually reserved for jump instructions only.
(f) Immediate: The 8-bit operand follows the opcode immediately (i.e. is in the next memory location).
(g) Base page: As absolute, but with only one byte specified, giving an address in the range 0 to FFH.

Addressing modes are illustrated in fig. 1.5.

1.2.3 INSTRUCTION SETS

The instructions available in 8-bit microprocessors fall into a number of categories: arithmetic; logical; data moves; branches and calls, and CPU control. Taking each category at a time, typical instructions are shown: