CHAPTER 3

Microprocessor Primer

Contents

There are only 10 types of people in the world: Those who understand binary and those who don't.

Anonymous

Real-Time Embedded Systems. http://dx.doi.org/10.1016/B978-0-12-801507-0.00003-1

3.1 Introduction to Microprocessors

A microprocessor is a general-purpose central processing unit (CPU) manufactured on a single *integrated circuit*. To be useful, a microprocessor has to work with other components, such as a memory system for storing instructions and data, and external circuits to communicate with the peripheral environment.

A microprocessor is merely the processing core of an application system. Sometimes, a microprocessor and some commonly used circuits and components (e.g., memory, parallel I/O, serial I/O, clock circuit, etc.) are integrated together on a single chip, which is typically called a *microcontroller*. A microcontroller is truly a computer on a chip. Using a microcontroller can greatly ease the hardware architecture design, especially when it has all the necessary peripherals for the system to be developed.

The first commercial microprocessor was the Intel 4004, introduced by Intel Corporation in 1971 for electronic calculators. Since then, powerful, low-cost microprocessors have been used in myriads of embedded systems found almost everywhere: household appliances, cars, toys, DVD players, AV receivers, cell phones, and high-definition TVs, to mention only a few.

3.1.1 Commonly Used Microprocessors

A few commonly used microprocessors are given in Table 3.1.

Intel 4004 is a 4-bit processor with only 2048 bits of addressable memory. Intel 8080 was introduced in 1974. It is an 8-bit processor with 16 address lines that can access 64 KB memories. The most successful family of processors started with the 16-bit Intel 8086, which sets the basis for the Intel x86 architecture (also referred to as IA-32).

Introduced in 1985, Intel 80386 features three operating modes: real mode, protected mode, and virtual mode. The *protected mode* allows the processor to address up to 4 GB of memory. The *virtual mode* makes it possible to run one or more programs in a protected environment.

The Pentium processor was the first x86 processor with superscalar architecture.[1] The Pentium processor also features a 64-bit external data bus, which doubles the amount of information it is possible to read or write on each memory access.

Pentium Pro was introduced in 1995. It can not only access a bigger memory space, but also implements a new architecture called microarchitecture—a decoupled, 12-stage superpipelined architecture. In 2001, Intel introduced Itanium, a family of 64-bit

[1] A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. A superscalar processor can be envisioned as having multiple parallel pipelines, each of which processes instructions from a single instruction thread.

Table 3.1 The characteristics of some commonly used microprocessors

Architecture	Year	Family	Processing Width (bits)[a]	Address Width (bits)	Addressable Memory Space (1 byte = (8 bits)	External Memory Bus: Data (address)
Intel	1971	4004	4	12	$2^{12} = 4\,\text{K}(\frac{1}{2}\,\text{B})$	4
	1974	8080	8	16	$2^{16} = 64\,\text{KB}$[b]	8
	1978	8086	16	20	$2^{20} = 1\,\text{MB}$	16
	1985	80386	32	32	$2^{32} = 4\,\text{GB}$	32
	1993	Pentium	32	32	$2^{32} = 4\,\text{GB}$	64
	1995	Pentium Pro	32	36	$2^{36} = 64\,\text{GB}$	64
	1999	Pentium III	32	36	$2^{36} = 64\,\text{GB}$	64
	2001	Itanium	64	64	$2^{64}\,\text{B}$	128
Motorola	1979	68000	32 (16)	24	$2^{24} = 16\,\text{MB}$	16
Microchip	2000	PIC18F2XK20	16	C[c]: 21	$2^{21} = 2\,\text{MB}$	No
				D[d]: 12	$2^{12} = 4\,\text{KB}$	
	2000	PIC18F8X20 PIC18F97J60	16	C: 21	$2^{21} = 2\,\text{MB}$	16 (20)
				D: 12	$2^{12} = 4\,\text{KB}$	
	2007	PIC32MX	32	32	$2^{32} = 4\,\text{GB}$[e]	Not yet. 16 (16) through PMP
ARM	2001	ARM9E	32	C: 18	$2^{18} = 256\,\text{KB}$	32
				D: 18	$2^{18} = 256\,\text{KB}$	32
	2006	Cortex-M3	32	32	$2^{32} = 4\,\text{GB}$[e]	32

PMP, parallel master port.
[a]The processing width of a processor is the same as the width of the general-purpose registers and the arithmetic logic unit of the processor. It is also called the word width of the processor.
[b]When the address width is larger than the processing width, memory segmentation is done by the processor internally.
[c]Program code space.
[d]Data (random-access memory) space arranged in banks of size 256 bytes.
[e]Program code space and data space are still separated, but they are mapped into a unified virtual memory space.

microprocessors that implement the IA-64 architecture. Itanium processors are targeted for enterprise servers and high-performance computing systems.

Motorola 68000 (also called 68K) is a 16-/32-bit microprocessor. Although introduced in 1979, it is still in use in embedded applications. For example, it has been used in low-end printers

such as HP's LaserJet introduced in 1984, and Apple's LaserWriter introduced in 1985, in game consoles such as Sega's System 16, Mega Drive (Genesis) console, and Saturn console.

The PIC family of microcontrollers is made by Microchip Technology. PIC microcontrollers are popular in both industry and education because of their low cost, large user base, and wide availability of development tools. PIC microcontrollers feature on-chip program and data memory as well as many peripheral components. Some PIC microcontrollers (e.g., PIC18F8X20 and PIC18F97J60) even have an external memory interface to expand the internal memory space.

ARM processors have been used extensively in consumer electronics, including personal digital assistants, tablets, cell phones, music players, handheld game consoles, and computer peripherals such as hard drives and routers. As of 2009, ARM processors accounted for approximately 90% of all embedded 32-bit reduced instruction set computing (RISC) processors [35]. In 2010, over 6.1 billion ARM-based chips were sold, representing over 95% of the smartphone market, 10% of the mobile computer market, and 35% of the digital TV and set-top box market [53].

As integrated circuit technology advances, it becomes feasible to manufacture more and more complex processors on a single chip. It is clear that high-end applications demand more address space and computing power than those offered by 32-bit processors, and the transition from 32-bit computing to 64-bit computing has become the trend.

3.1.2 Microprocessor Characteristics

3.1.2.1 Architectures

As far as memory access is concerned, a microprocessor may belong to either von Neumann architecture or Harvard architecture.

The *von Neumann architecture* is named after the mathematician and early computer scientist John von Neumann. The von Neumann architecture has two features:

- Data and instructions (executable code) are stored in the same address space. On the one hand, this allows for self-modifying code. On the other hand, this opens security holes. For instance, a program may attempt to run instructions from memory that contains data, or a program might write data into memory containing instructions.
- The processor interfaces with memory through a single set of address/data buses. Since an instruction fetch and a data operation cannot occur at the same time because they share a common bus, this often limits the performance of the system. This is referred to as the *von Neumann bottleneck*.

The *Harvard architecture* is named after the Harvard Mark I computer. The Harvard architecture has two features:

- Data and instructions (executable code) are stored in separate address spaces. For instance, the instruction space may be accessed by 20 address lines, while addresses in the data space may only have 16 bits. In addition, program memory is typically read-only memory (ROM). It is impossible for program contents to be modified by the program itself.
- There are two sets of address/data buses between the processor and the memory. Since instruction fetches and data operations are carried on separate buses, it is possible to access program memory and data memory simultaneously.

Modern microprocessor designs (e.g., ARM) incorporate aspects of both Harvard and von Neumann architectures. For instance, a modified Harvard architecture has separate instruction and data caches that are backed by a common address space. While the processor executes from cache, it acts as a pure Harvard architecture. When accessing backing memory, it acts like a von Neumann architecture (where code can be moved around like data).

As far as the instruction set is concerned, a microprocessor may be one of two types: a complex instruction set computing (CISC) processor or an RISC processor.

A *CISC* processor has multiple addressing modes and runs "complex instructions" where a single instruction may execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store). While this leads to high code density, it often requires manual optimization of assembly code for embedded systems.

An *RISC* processor runs compact, uniform instructions where the amount of work any single instruction accomplishes is reduced (e.g., separate instructions for I/O and data processing). Such compact instructions allow effective compiler optimization and facilitate pipelining, typically leading to higher performance than CISC. As an overhead, however, RISC inevitably produces more lines of code (larger memory footprint) than CISC.

Today, CISC processors dominate the personal computer market and are used in a significant fraction of the low- and mid-range servers and workstations. RISC processors have been successfully used across a wide range of platforms, from mobile devices such as cellphones and tablets to some of the world's fastest supercomputers.

Figure 3.1 gives some example microprocessors for each category, where SHARC (for "*s*uper *H*arvard *a*rchitecture") is a processor family that offers many features required by digital signal processor applications: exceptional core and memory performance and outstanding I/O throughput.

	Von Neumann	Harvard
CISC	X86 (8086, Pentium) Motorola 68000	SHARC (DSP)
RISC	ARM7, SPARC, MIPS, PowerPC	ARM9, PIC

Figure 3.1
Microprocessor architecture types.

3.1.2.2 Processing width

The *processing width* is an important characteristic of a processor design. For example, the Intel 8086 is a 16-bit processor, where the number 16 is its processing width. The processing width of a processor is exactly the width of its working memory (registers).

The term "word" also refers to the largest processing unit that can be transferred to and from the working memory in a single operation of a processor. In other words, the number of bits in a word (the word size or word width) is the same as the processing width (register width). For instance, PIC18F8720 is a 16-bit processor; its word size is 16 bits and a word is composed of 2 bytes. ARM926EJ-S is a 32-bit processor; its word size is 32 bits and a word is composed of 4 bytes.

Modern processors usually have a word size of 16, 32, or 64 bits.

3.1.2.3 I/O addressing

A microprocessor typically accesses I/O devices in two ways.

I/O devices can be placed in a microprocessor's memory address space. This approach is called *memory-mapped I/O*, where I/O devices and memory components are indistinguishable to the processor. Memory-mapped I/O simplifies coding and testing. Any instruction that references memory may be used to access an I/O port located in the memory space. For example, the Intel 80386 processor can use the MOV instruction to transfer data between any register and a memory-mapped I/O port. The Motorola 68000 uses memory-mapped I/O. PIC family processors also have all peripherals memory mapped (through ports and port registers).

Some processors may feature a special signal pin (e.g., $M/\overline{IO}$ in the Intel 8086 and Pentium processors) to indicate whether a memory device or an I/O device is accessed. In such a case, the designer can choose whether to map I/O devices into the memory space or use a separate *I/O address space*. In the latter case, all I/O devices are treated differently from normal memory locations. Typically special I/O instructions are needed to access I/O ports.

3.1.2.4 Reset vector

The *reset vector* of a processor is the default location where, upon a reset, the processor will go to find the first instruction to execute. In other words, the reset vector is a pointer or address where the processor should always begin its execution. This first instruction typically branches to the system initialization code.

Most microprocessors have reset vectors fixed at the two ends of their address spaces. Here are a few examples:

- The Intel 8086 processor has its reset vector at FFFF0h, the high end of its address space.
- PIC18 processors have the reset vector located at the low end, 0000h.

- The Motorola 68000 processor has a 1024-byte vector table beginning at 000000h. The first entry is the reset vector, which is at 000000h. This vector table also contains pointers to routines used by the processor, operating system, and users.
- ARM family processors have the address 0x00000000 reserved for the vector table (including reset vector, undefined instruction vector, software interrupt vector, interrupt request vector, fast interrupt vector), and the reset vector is at 0x00000000. On some processors (e.g., ARM926EJ-S) the vector table can be optionally located at a higher address in memory, 0xffff0000.

3.1.2.5 Endianness

The term "endian" or "*endianness*" refers to the ordering of individually addressable units (e.g., bytes) within a larger data item (e.g., word) as stored in external memory (or, sometimes, as sent on a serial connection).

As far as microprocessors are concerned, a *big-endian* processor stores the most significant byte (MSB) first (i.e., at the lowest byte address), while a *little-endian* processor stores the least significant byte (LSB) first (see Figure 3.2). When we write the hex value 0x0a0b0c0d from left to right, we are implicitly writing in big-endian style.

Different processors order their multibyte data (i.e., 16-, 32-, or 64-bit words) in different ways. For example, the Intel X86 family and PIC processors use little-endian mode, and the Motorola 68000 family uses big-endian mode. Some processors, such as ARM processors, can be set up to be in either little-endian or big-endian mode.

Endianness is typically transparent to a user of a computer. However, difficulty arises when different types of computers attempt to communicate with one another over a network. Internet Protocol (IP) defines *big endian* as the standard network byte order used by IP and many higher-level protocols over IP for all numeric values. When communicating over a

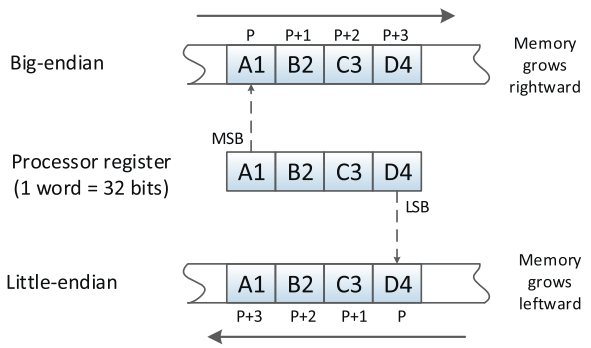

Figure 3.2
Microprocessor endianness.

network composed of both big-endian and little-endian machines, *high-level software should format packets of data in network byte order*. The Berkeley sockets application programming interface defines a set of functions[2] for little-endian host machines to convert 16-bit and 32-bit integers to and from network byte order.

A software system is called "endian clean" when it always reads and writes a multibyte data item as a whole rather than byte-by-byte. An "endian-clean" software system can be recompiled with no changes for big-endian or little-endian machines. It is encouraged to write "endian-clean" software, whenever possible, for networked embedded systems.

3.2 Microchip PIC18F8720

Microchip PIC18F8720 is a family of high-performance RISC microprocessors [4]. PIC18F8720 features a hardware stack for storing return addresses and a wide range of hardware interfaces including 10-bit A/D converters with 16 input channels, five timers, an external memory interface, and nine general-purpose I/O ports (one of which can be reconfigured as an 8-bit parallel slave port for direct processor-to-processor communications). Figure 3.3 shows the pin diagram of an 80-pin PIC18F8720 microprocessor, where many pins are physically multiplexed. For instance, the external memory interface (A19:A16, AD15:AD0) is multiplexed with I/O ports D, E, and H.

The PIC18F8720 device runs from a clock (the `OSC1` pin) that is four times faster than its instruction cycle. The four clock pulses are a quarter of the *instruction cycle* in length and are referred to as Q1, Q2, Q3, and Q4. Generally speaking, instruction fetching and execution are pipelined: while a 2-byte instruction is executed in an instruction cycle, the next 2-byte instruction can be fetched during the same cycle.

3.2.1 Memory Organization

PIC18F8720 devices have separate spaces for data memory and program memory, which is illustrated in Figure 3.4.

Data space in electrically erasable programmable ROM (EEPROM). PIC18F8720 devices have 1024 bytes of data EEPROM, which is a byte-addressable memory space that has been optimized for the storage of frequently changing values (e.g., program variables that are updated often). Variables that change infrequently (such as constants, IDs, etc.) should be stored in flash program memory.

[2] Functions htons() and htonl(), respectively, convert a short and a long integer from a small-endian host to the network format. Functions ntohs() and ntohl() are for the opposite direction.

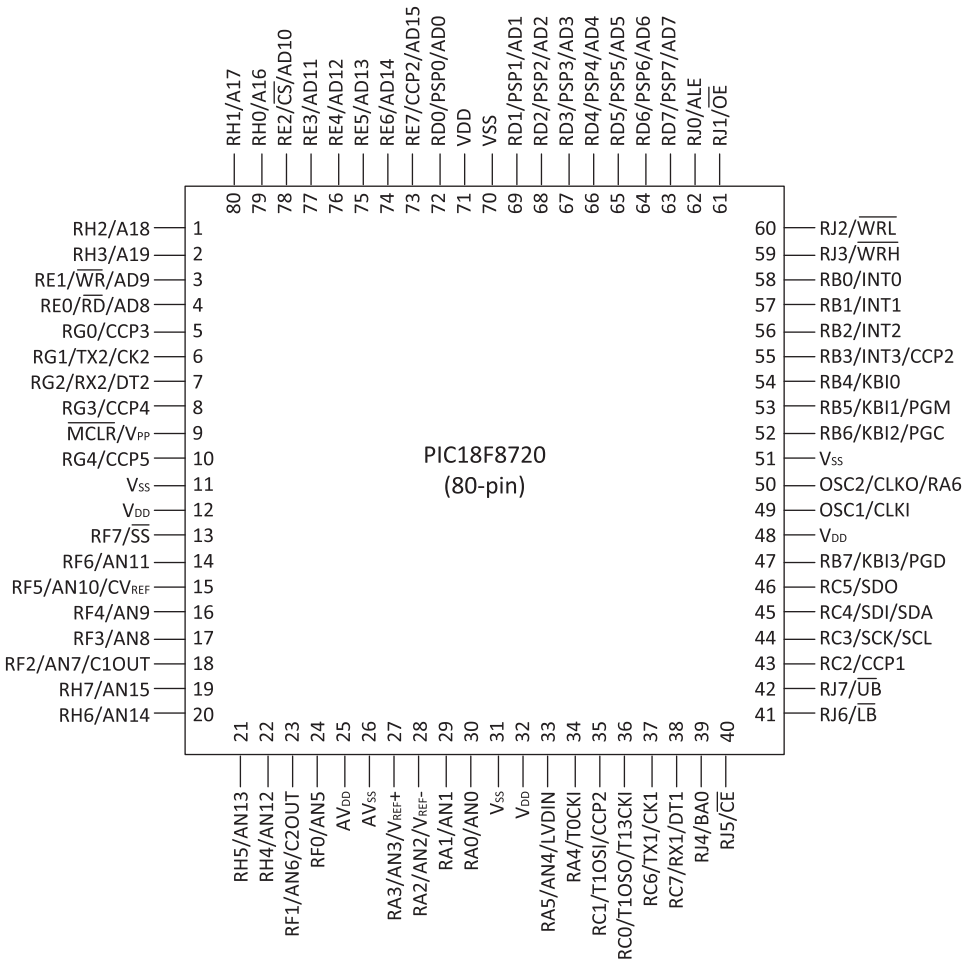

Figure 3.3
Microchip 16-bit 80-pin PIC microprocessor.

When interfacing with the data EEPROM block, the register EEDATA holds the 8-bit data for read/write, while the register EEADR and the two least significant bits (LSbs) of EEADRH hold the address of the EEPROM location being accessed.

Data space in random-access memory (RAM). As shown in Figure 3.4, PIC18F8720 devices also have a data RAM space that contains both special-function registers (SFR) and general-purpose registers (GPR). The SFRs are used for control and status of the microcontroller and peripheral functions, while GPRs are used for storing application data such as intermediate computational values, local variables of subroutines, and task contexts.

Each register in the RAM space has a 12-bit address. To enable rapid access to those registers, a *banking scheme* is implemented where the whole space is partitioned into 16 banks of

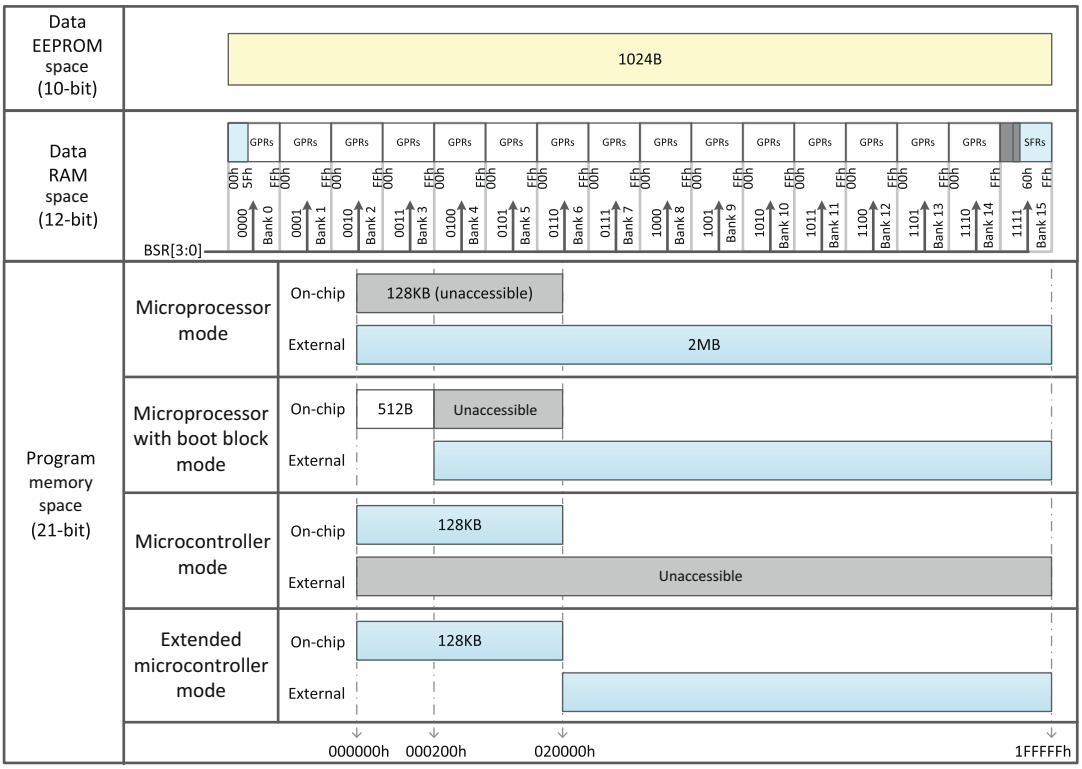

Figure 3.4
PIC18F8720 memory organization.

256 bytes each. To *directly access*[3] a register, the lower 8 bits of its address are embedded in the instruction, and the upper 4 bits of its address, which specify the bank to be accessed, are taken from the lower 4 bits of the bank select register (BSR[3:0]).

GPRs start at the first location of bank 0 and grow upward up to the last location of bank 14. Bank 15 contains 160 SFRs with addresses ranging from F60h to FFFh; its lower portion (from F00h to F5Fh) is unused.

The first segment (96 bytes) of bank 0 and the SFR segment (160 bytes) of bank 15 also form a so-called *access bank*, which allows the commonly used registers (SFRs and some GPRs) to be accessed in a single instruction cycle. A reserved bit in the instruction word is used to specify if the operation is to occur in the bank specified by the bank select register or in the access bank.

[3] There is also an *indirect addressing mechanism*, where any instruction using an INDF register actually accesses the data memory location pointed to by the 12-bit file select register that is associated with that INDF register.

Program memory space. PIC18F8720 has 128 KB of on-chip flash memory. Moreover, it implements a 21-bit program counter, which is capable of addressing a 2 MB program memory space through the external memory interface. Depending on the two LSbs of the CONFIG3L register[4], PIC18F8720 may operate in one of four modes:

(i) Microcontroller mode. This is the default mode. Under this mode, the processor can access only the on-chip flash memory. Attempts to read above the physical limit of the on-chip flash memory return 0's.
(ii) Microprocessor: Under this mode, the processor can access only the external program memory; the contents of the on-chip flash memory are ignored. The 21-bit program counter permits access to a 2 MB linear program memory space.
(iii) Microprocessor with boot block. Under this mode, the processor accesses the boot block (from addresses 000000h to 0001FFh)[5] of the on-chip flash memory. Above this, external program memory is accessed all the way up to the 2 MB limit. Program execution automatically switches between the two memories, as required.
(iv) Extended microcontroller mode. Under this mode, the processor can access its entire on-chip flash memory; above this, the device accesses external program memory up to the 2 MB program space limit. The execution automatically switches between the two memories, as required.

The program memory space is typically used for storing instructions. The address of the instruction to fetch for execution is specified by the 21-bit program counter, where the low byte, the high byte, and the upper 5 bits are contained in registers PCL, PCH, and PCU, respectively.

The program memory is addressed by bytes. However, since PIC18F8720 uses a 16-bit instruction set, the LSb of the PCL register is fixed to a value of 0 to prevent the program counter from becoming misaligned with word instructions. In other words, the program counter increments by 2 to address sequential instructions in the program memory.

The program memory space can also be used for storing data. A block containing data is not required to be word aligned. There are two instructions that allow a processor to move bytes between the program memory space and the data RAM: (a) table read `TBLRD`, which retrieves data from program memory (or data EEPROM[6]) and places it into the data RAM space, and

[4] The configuration bits can be programmed (set to "0") to select various device configurations. These bits are mapped to the *configuration memory space*, starting from 300000h through 3FFFFFh, which is beyond the user program memory space (2 MB: 000000h to 1FFFFFh), and can be accessed using only table reads and table writes.
[5] By use of a bootloader routine located in the protected boot block at the top of program memory, it becomes possible to create an application that can update itself in the field.
[6] Depending on the settings of the EECON1 register.

(b) table write TBLWT, which stores data from the data RAM space in program memory (or data EEPROM). Table read and table write operations move data between these memory spaces through an 8-bit table latch register (TABLAT).

A table pointer TBLPTR is used in reads, writes, and erases of the program memory. TBLPTR comprises three SFRs—TBLPTRU, TBLPTRH, and TBLPTRL—which jointly form a 21-bit-wide address pointer.

To read a byte from the flash program memory, simply set TBLPTR to point to its address in the program space, then execute the TBLRD instruction, which will place the byte into TABLAT.

Writing to the flash program memory is a little complicated because the minimum programming block is 8 bytes (four words). In particular, prior to a flash programming operation, the TBLWT instruction has to be executed eight times, with each essentially writing 1 byte contained in TABLAT to one of the eight holding registers. At the end of updating the holding registers, instruction execution is halted and a long write cycle is started to write the contents of the eight holding registers to the flash program memory.

In all modes, a PIC18F8720 processor has complete access to the data RAM and data EEPROM spaces. In addition, the processor, except for the microcontroller mode, can access external memory devices (such as flash memory, erasable programmable ROM (EPROM), and static RAM (SRAM)) as program or data memory through the external memory interface. Depending on the values of the two LSbs of the MEMCON register, there are three operating modes for the *external memory interface*: word write, byte select, and byte write.

3.2.2 Word Write Mode

This mode allows instruction fetches and table reads from, and table writes to, all forms of 16-bit (word-wide) external memories (EPROM, flash memory, or SRAM). An example configuration of word write mode is shown in Figure 3.5, where, in contrast to Figure 3.3, only those pins that are relevant to the external memory interface are shown.

Whenever the microprocessor accesses external memory, the chip enable signal $\overline{\text{CE}}$ is active (asserted low); it is thus connected to the $\overline{\text{CE}}$ pin of the flash memory device.

To allow the microprocessor to fetch data from external memory, the corresponding output enable signals $\overline{\text{OE}}$ (active low) are connected. $\overline{\text{OE}}$ is inactive during a cycle where a table write is executed. During a cycle where an instruction is fetched or a table read is executed, $\overline{\text{OE}}$ is asserted at the beginning of Q3 and deasserted at the end of Q4. Data (16-bit word) are fetched from external memory at the low-to-high transition edge of $\overline{\text{OE}}$.

The write high control signal $\overline{\text{WRH}}$ is active (asserted low) whenever the microprocessor writes to an odd address (or the high byte of a word). It is connected to $\overline{\text{WR}}$ of the memory device so

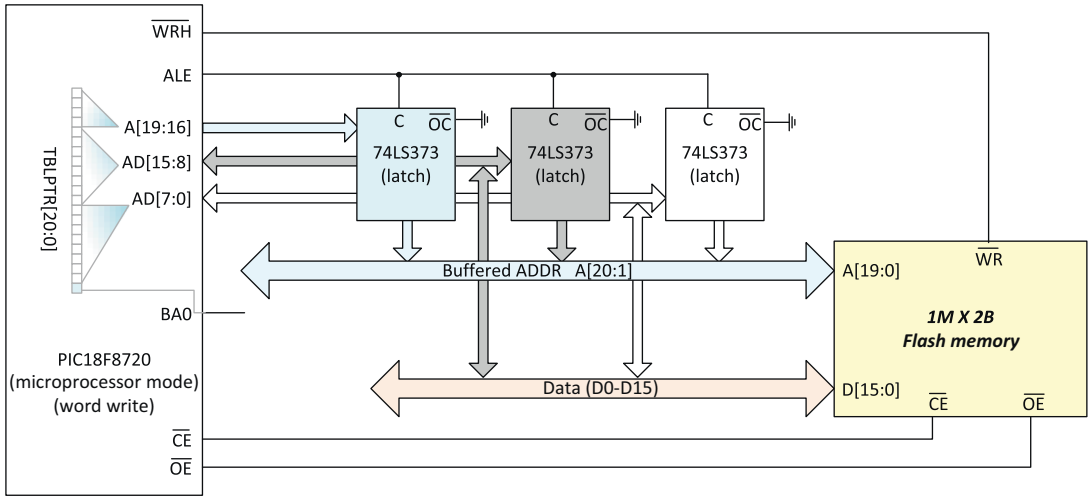

Figure 3.5
PIC18F8720 word write mode.

that the device is ready for writing whenever the microprocessor attempts to write the high byte of a word.

The external memory interface has 20 address lines: `A[19:16]` and `AD[15:0]`. Internally, the 21-bit table pointer `TBLPTR` (similarly the program counter) is mapped to the external memory interface as follows:

- the least significant bit `TBLPTR[0]` is copied to `BA0`;
- `TBLPTR[20:17]` appear on address pins `A[19:16]`;
- `TBLPTR[16:9]` appear on pins `AD[15:8]`; and
- `TBLPTR[8:1]` appear on pins `AD[7:0]`.

Consequently, `A[19:16]` together with `AD[15:0]` allow memory access based on a word boundary. If necessary, `BA0` can be used to access memory by byte.

Because the pins `AD[15:0]` are multiplexed for both data and addresses, the control pin `ALE` (for "address latch enable") is used to enable the address latch devices (74LS373), so that the address on the external address bus is still valid while data are being transferred. Note that to be consistent with the table pointer, `A[20:1]` are used to name the buffered address lines.

Figure 3.6 gives a timing diagram illustrating the signal interactions for two consecutive table write operations while the microprocessor accesses the external memory in word write mode.

First, during Q1 of an instruction cycle, `ALE` is enabled while the address information is placed on pins `AD[15:0]` and `A[19:16]`. On the falling edge of `ALE`, the address is latched and available on the external bus.

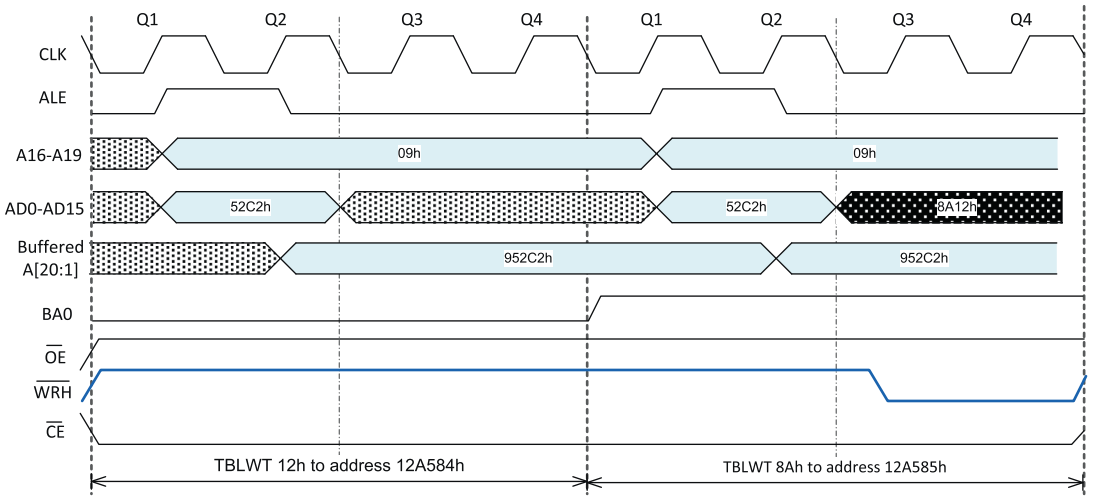

Figure 3.6
PIC18F8720 word write mode timing diagram.

Second, this mode makes a distinction between TBLWT instruction cycles for even or odd addresses. During a TBLWT cycle to an even address, that is, TBLPTR[0] or BA0 is 0, the TABLAT data are transferred to a holding latch and AD[15:0] is tristated[7] for the data portion of the bus cycle. The $\overline{\text{WRH}}$ signal is not activated. During a TBLWT cycle to an odd address i.e., TBLPTR[0] or BA0 is 1, the TABLAT data are presented on the upper byte of the AD[15:0] bus. At the same time, the contents of the holding latch are presented on the lower byte of the AD[15:0] bus. The $\overline{\text{WRH}}$ signal is activated and the 16-bit data are written to the corresponding word location.

Last but not least, it is worth noting that the address 12A584h used by the table pointer becomes 952C2h; this buffered address does not reflect the LSb of 12A584h. As an exercise, check that 952C2h×2 XOR BA0 equals 12A584h in the first cycle, and equals 12A585h in the second cycle. This is because the LSB of a word location is accessed in the first cycle (BA0 = 0), while the MSB of the word location is accessed in the second cycle (BA0 = 1).

3.2.3 Byte Select Mode

This mode allows instruction fetches and table reads from, and table writes to, 16-bit external memories with byte selection capability, as well as 8-bit external memories. In this mode, the

[7] A tristate logic allows an output port to assume a high impedance state in addition to the logic high and low levels. When an output is tristated, its influence on the rest of the circuit is removed. A pull-up or pull-down resistor is typically used to try to pull the node to high or low voltage levels. If the node is not in a high-impedance state, extra current from the resistor will not significantly affect its voltage level.

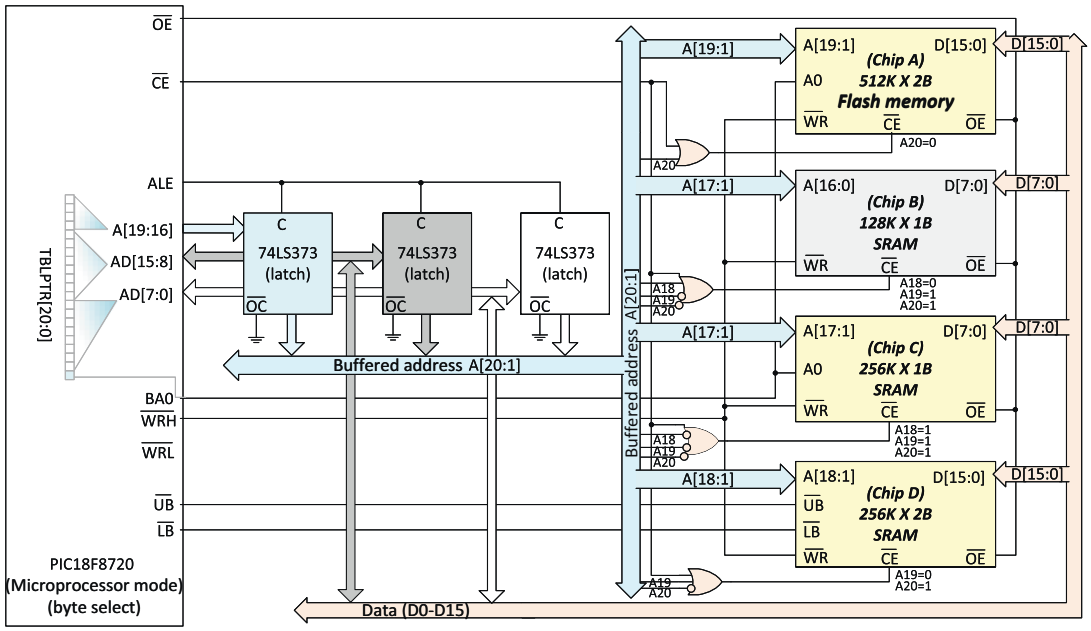

Figure 3.7
PIC18F8720 byte select mode.

write high signal $\overline{\texttt{WRH}}$ is asserted whenever the microprocessor writes to a memory location, regardless of an odd or even address. An example configuration of byte select mode is shown in Figure 3.7, where $\overline{\texttt{WRH}}$ is connected to the $\overline{\texttt{WR}}$ pin of each memory chip.

First, notice that there are four memory chips: chip A and chip D are word-wide memory devices, and chip B and chip C are byte-wide memory devices. At any given time, only one of the four devices can be enabled for reading or writing. This is achieved by the OR gates. Specifically, the chip enable signal $\overline{\texttt{CE}}$ together with some of the address lines is used to select memory chips:

- Chip A is selected (active) only when $\overline{\texttt{CE}}$ is asserted and `A[20]`=0.
- Chip B is selected only when $\overline{\texttt{CE}}$ is asserted, `A[20]`=1, `A[19]`=1, and `A[18]`=0.
- Chip C is selected only when $\overline{\texttt{CE}}$ is asserted, `A[20]`=1, `A[19]`=1, and `A[18]`=1.
- Chip D is selected only when $\overline{\texttt{CE}}$ is asserted, `A[20]`=1, and `A[19]`=0.

Second, `BA0` is connected to the `A0` pin of chip A. While `A[19:1]` allows the microprocessor to address a word location on chip A, `A0` will allow the microprocessor to individually access the MSB or LSB of the word. The `A[0]` pin of chip C is also connected to `BA0`. This ensures that chip C gets a contiguous block of addresses. Chip B has almost the same wiring as chip C except that its `A[0]` is connected to bus line `A[1]` rather than `BA0`. Because of this, the block

of addresses allocated to chip B is not contiguous: each location on chip B gets one odd address and one even address! In other words, the microprocessor can use two different addresses to access each location on chip B.

Third, chip D is a word-wide SRAM, which, according to the JEDEC memory standards, uses $\overline{\text{UB}}$ or $\overline{\text{LB}}$ to indicate whether the upper byte or the lower byte is to be selected. Thus, pins $\overline{\text{UB}}$ and $\overline{\text{LB}}$ of chip D are connected to the corresponding microprocessor pins, respectively. $\overline{\text{UB}}$ is asserted when the microprocessor accesses an odd address, and $\overline{\text{LB}}$ is asserted when the microprocessor accesses an even address.

The memory mappings of the memory devices are given in Table 3.2. These four chips together form a space with 2 MB addresses, but remember that although chip B has 128 KB, it is allocated with 256K addresses: two for each byte location.

Figure 3.8 gives a timing diagram illustrating the signal interactions for two consecutive table write operations while the microprocessor is in byte select mode.

It is worth noting that $\overline{\text{WRH}}$ is asserted in each write cycle, and the TABLAT data are copied on both the MSB and the LSB of the data bus (e.g., 1212h) This is not a problem for word-wide memory devices (say, chip A or chip D), because either BA0 or $\overline{\text{UB}}$/$\overline{\text{LB}}$ (they originate from the LSb of the TBLPTR register) can be used to select the right byte to be written.

Also note that the microprocessor writes to an even address in the first cycle, so BA0 = 0 and $\overline{\text{LB}}$ is asserted. It then writes to an odd address in the second cycle, so BA0 = 1 and $\overline{\text{UB}}$ is asserted. As an exercise, check on which chip the address 12A584h appears.

Table 3.2 The memory mapping of Figure 3.7

Chip	Memory width	Address Bus: A[20] —— A[1] $\overline{\text{CE}}$	Chip Address	A0	Address Range	Size
A	Word	0	++++++++++++++++++	+[a]	0 0000 0000 0000 0000 0000~ 0 1111 1111 1111 1111 1111	1 MB
B	Byte	110	++++++++++++++++		1 10 00 0000 0000 0000 000x~ 1 10 11 1111 1111 1111 111x[b]	128 KB
C	Byte	111	++++++++++++++++	+	1 11 00 0000 0000 0000 0000~ 1 11 11 1111 1111 1111 1111	256 KB
D	Word	10	++++++++++++++++++	+	1 0 000 0000 0000 0000 0000~ 1 0 111 1111 1111 1111 1111	512 KB

[a] A bit marked by + indicates that it is used by the device; different value combinations of those "+" bits refer to different locations on the device.

[b] A bit marked by x indicates that it is unused by the device. When there are unused bits, each location of the device may be accessed by more than one address. In this case, the last bit can be 0 or 1, implying that each location on chip B virtually has two addresses.

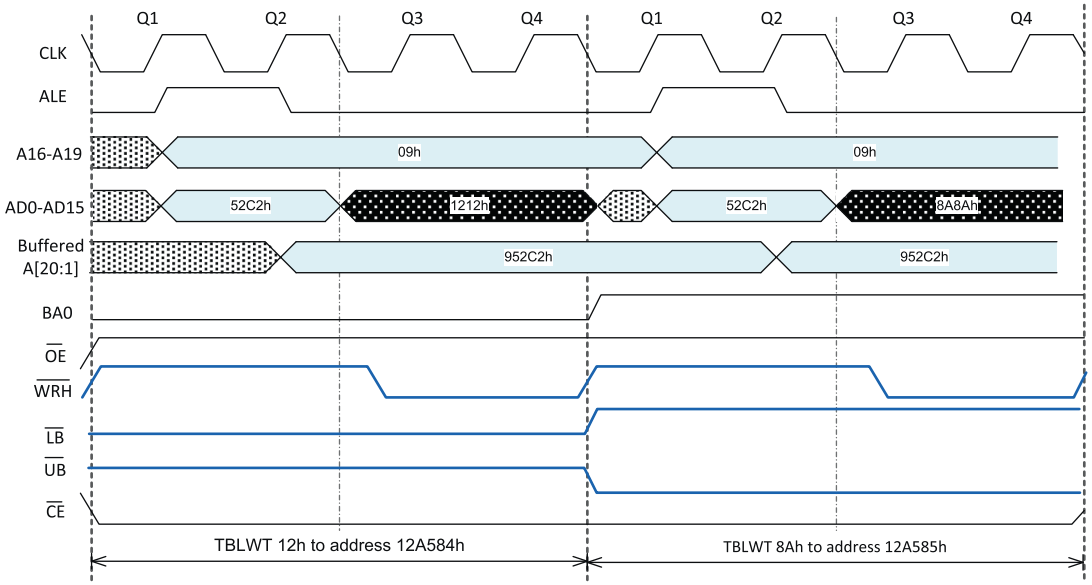

Figure 3.8
PIC18F8720 byte select mode timing diagram.

3.2.4 Byte Write Mode

This mode allows table reads from, and table writes to, 8-bit-wide external memories. In this mode, during a TBLWT instruction cycle, the TABLAT data are presented on both the upper and the lower bytes of the AD[15:0] bus. The appropriate $\overline{\text{WRH}}$ or $\overline{\text{WRL}}$ control line is asserted on the LSb of TBLPTR.

An example configuration of byte write mode is shown in Figure 3.9. Like the other modes, AD<15:0> from the microprocessor are mapped through the latch to A<16:1> of the buffered address bus, and A<19:16> are mapped directly to A<20:17>.

Let us first look at the wiring of chip A and chip B. They are of the same type, of the same size (512 KB), and even have the same chip selection signal (A[20]=0). They differ in two ways: (a) the data pins of chip A are connected with the upper byte (D[15:8]) of the data bus, while the data pins of chip B are connected with the lower byte (D[7:0]) of the data bus; (b) the write enable pin $\overline{\text{WR}}$ of chip A is connected to $\overline{\text{WRH}}$, while the write enable pin $\overline{\text{WR}}$ of chip B is connected to $\overline{\text{WRL}}$. This implies that data are written to chip A when the address is odd ($\overline{\text{WRH}}$ is asserted as the LSb of TBLPTR is 1), written to chip B when the address is even ($\overline{\text{WRL}}$ is asserted as the LSb of TBLPTR is 0).

The wiring of chip C is almost the same as that of chip B, except that it is selected when A[20] is 1. This ensures that chip C will get a different block of addresses. Note that chip C

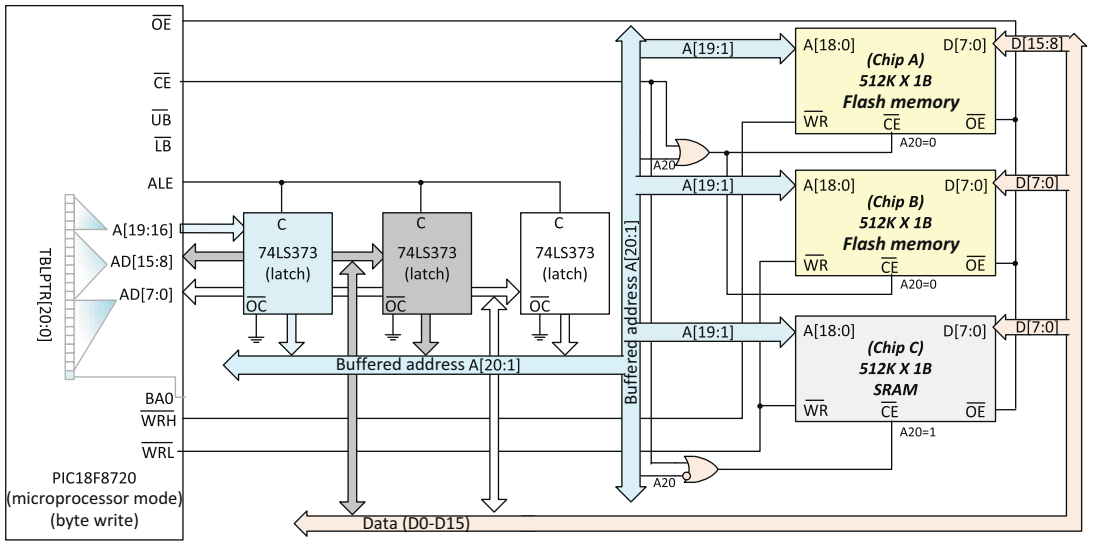

Figure 3.9
PIC18F8720 byte write mode.

permits table operations to only even addresses. Particularly, a table write to an even address of chip C will copy the contents of TABLAT to both bytes of the AD<15:0> bus and activate the $\overline{\text{WRL}}$ signal. The lower byte will then be written to chip C. A table write to an odd address of chip C will also copy the contents of TABLAT to both bytes of the AD<15:0> bus. However, this time, the $\overline{\text{WRH}}$ signal is activated instead. In this case, nothing will be written to chip C. This is similar to the case for chip B in Figure 3.7: although the usage of the memory chips is 100%, a noncontiguous block of addresses is assigned to them. One difference is that each location on chip C gets an even address, while each location on chip B in Figure 3.7 gets an even address and an odd address. The reason is that in byte select mode, $\overline{\text{WRH}}$ is asserted regardless of whether the address is even or odd.

Figure 3.10 gives a timing diagram illustrating the signal interactions for two consecutive table write operations while the microprocessor is in byte write mode. Note that $\overline{\text{WRL}}$ is asserted in the first cycle because an even address is accessed, while $\overline{\text{WRH}}$ is asserted in the second cycle because an odd address is accessed. In each write cycle, the TABLAT data are copied on both the MSB and the LSB of the data bus (e.g., 1212h).

3.3 Intel 8086

Intel 8086 is a 16-bit microprocessor that represents the first design of the x86-family architecture [1]. The Intel 8086 has a von Neumann architecture and belongs to the CISC category. Its pin diagram is shown in Figure 3.11.

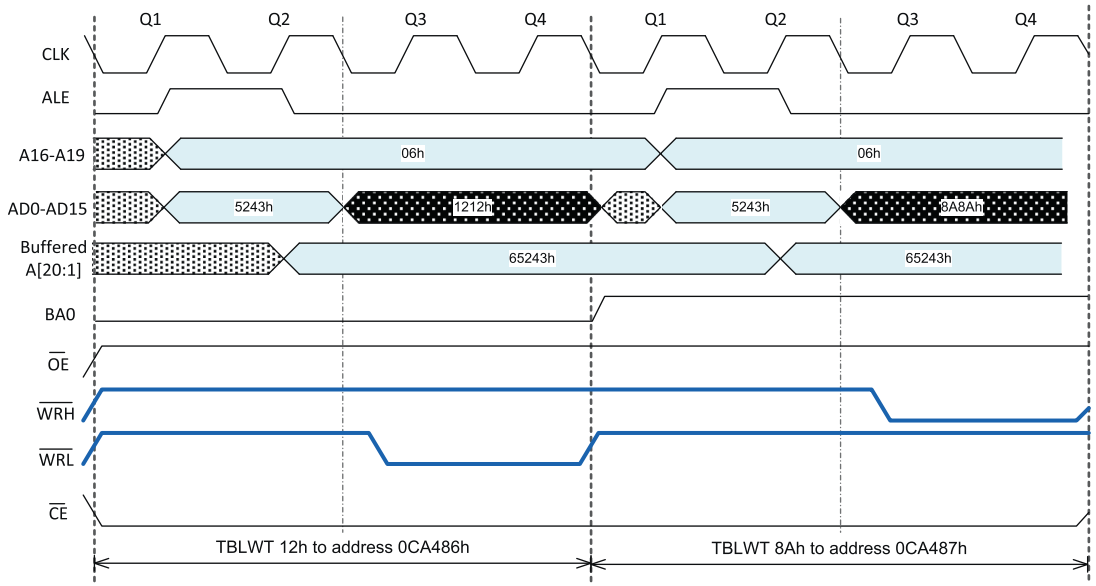

Figure 3.10
PIC18F8720 byte write mode timing diagram.

Intel 8086 supports two hardware modes (hard wired into the circuit and unchangeable by software): maximum mode and minimum mode. In maximum mode pin 33 (MN/$\overline{\text{MX}}$) is wired to ground, and in minimum mode pin 33 is wired to voltage. Changing the state of pin 33 also changes the function of some other pins as indicated in Figure 3.11. The minimum mode is intended for small single processor systems, while the maximum mode is for medium-sized or large systems that use more than one processor (e.g., 8087 coprocessor in an IBM PC).

CLK provides the basic timing for the processor and bus controller. Each *bus cycle* (read, write, or interrupt acknowledge) consists of at least four clock cycles, which are referred to as T1, T2, T3, and T4. The 8086 processor has its address lines and data lines multiplexed (AD0-AD15). This means that the same pins are used to carry both address and data information at different times: the address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. In the minimum mode, the address latch enable signal ALE can be used to enable chips for latching addresses.

The bus high enable signal $\overline{\text{BHE}}$ is asserted (active low) during T1 of a bus cycle when a byte is to be transferred on the high portion (the most significant half) of the data bus, that is, pins D15:D8.

The interrupt request signal INTR is a level-triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. It can be internally masked by software resetting the interrupt enable

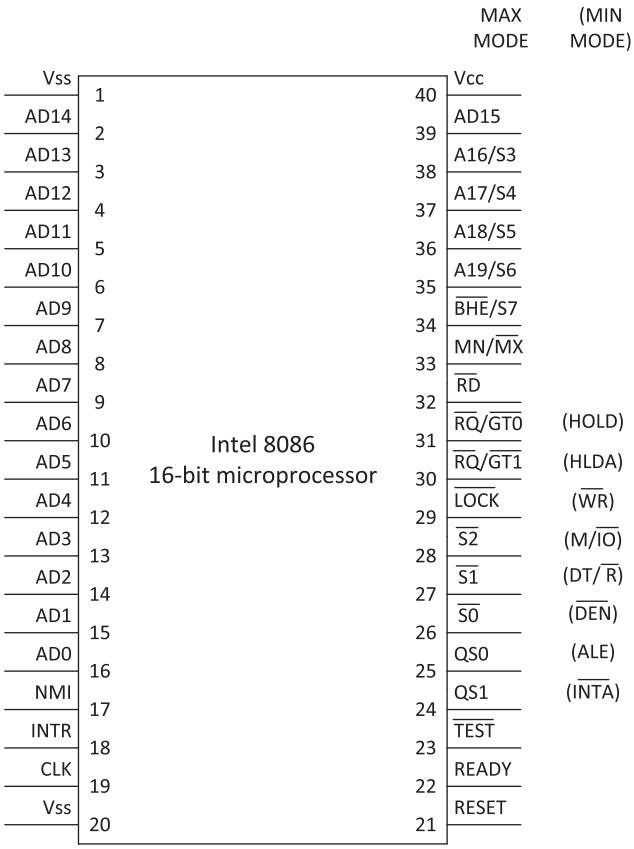

Figure 3.11
Intel 8086 16-bit microprocessor.

bit. The nonmaskable interrupt signal `NMI` is an edge-triggered input,[8] which has higher priority than the maskable interrupt request signal `INTR`. The interrupt acknowledge signal $\overline{\texttt{INTA}}$ is active (low) during T2 and T3 of each interrupt acknowledge cycle.

3.3.1 Memory Organization

The 8086 processor has a 20-bit address bus, which gives a physical address space of up to 1 MB (2^{20}), addressed as 00000h to FFFFFh. However, the maximum linear address space was limited to 64 KB, simply because the internal registers are only 16 bits wide. A technique called "internal segmentation" has to be used for applications that are above the 64 KB boundary. There are four 16-bit segment registers (CS for "code segment," DS for "data

[8] A transition from low to high initiates the interrupt at the end of the current instruction.

segment," ES for "extra data segment," and SS for "stack segment"). All memory references are of the form [segment:offset], relative to the base address contained in the corresponding segment register.

In particular, given a [segment:offset] pair, a 20-bit external (or physical) address is produced by segment $\times 2^4$ + offset, where segment $\times 2^4$ is called the *segment address*, which has its 16 most significant bits from the 16-bit segment register, and its four LSbs are all zeros. A 16-bit offset is always added to the 20-bit segment address to yield an external address. As an example, consider the pair [000Ah:000Ch] (i.e., 10:12). The segment value of 000Ah (10) would give a segment address at 000A0h (160) in the linear address space. The address offset 000Ch can then be added to the segment address: 000A0h + 000Ch = 000ACh (160 + 12 = 172). Such address translations are carried out automatically by the segmentation unit of the processor.

Owing to the 4-bit shift, each segment virtually begins at a multiple of 16 bytes, from the beginning of the 1 MB address space. Particularly, the last segment, FFFFh, begins at linear address FFFF0h, 16 bytes before the end of the 20-bit address space. Since all segments are 64 KB long (with wraparound at the high end) and they increment by 16 bytes, there are huge overlaps among segments, and each location in the linear memory address space can be accessed by $2^{16}/2^4 = 2^{12} = 4096$ different [segment:offset] pairs. For example, the memory location 00100h can be referred to by [0000h:0100h], [0001h:00F0h], [0002h:00E0h], [0003h:00D0h], etc. Although complicated, this scheme has its advantages. For instance, a small program (less than 64 KB) can be loaded starting at a fixed offset in its own segment, avoiding the need for relocation, with at most 15 bytes of alignment waste.

Locations from address FFFF0h to address FFFFFh are reserved. Following reset, the CPU will always begin execution at location FFFF0h, which is a jump to the initial program loading routine. The block from 00000h to 003FFh is reserved for the interrupt table, where each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address.

The memory is physically organized as a high bank (D15-D8) and a low bank (D7-D0) of 512 KB addressed in parallel by the processor's address lines A19-A1. The processor provides two enable signals, $\overline{\text{BHE}}$ and A0, to selectively allow reading from or writing into either an odd byte location or an even byte location, or both. Byte data with an even address (A0 is low) are transferred on the D7-D0 bus lines, and byte data with an odd address (A0 is high) are transferred on the D15-D8 bus lines.

3.3.2 Separate I/O Address Space

An I/O device may be enabled, and the ports or registers on the device may be accessed, merely by the address lines (A19-A0). In such a case, the I/O devices are placed in the 8086

memory address space. As long as an I/O device responds like a memory chip, they are indistinguishable to the processor.

The 8086 processor also allows I/O devices to be placed in a separate I/O address space. This is achieved by the $\overline{S2}$ signal in maximum mode, and by the M/$\overline{IO}$ signal in minimum mode. M/$\overline{IO}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle. The I/O address space is accessed by AD15-AD0 (the address lines A19-A16 are zero in I/O operations), which can maximally accommodate 64 KB registers or 32K word registers on I/O devices.

Figure 3.12 shows the wiring of a universal asynchronous receiver/transmitter (UART) which is placed in the I/O address space. The 8086 processor operates in minimum mode, because the MN/$\overline{MX}$ pin is driven high.

3.3.2.1 Timing clock

The processor is connected to an external clock generator, 8284A. A crystal is attached to pins X1 and X2 of 8284A; CLK has an output frequency which is one third of the crystal frequency.

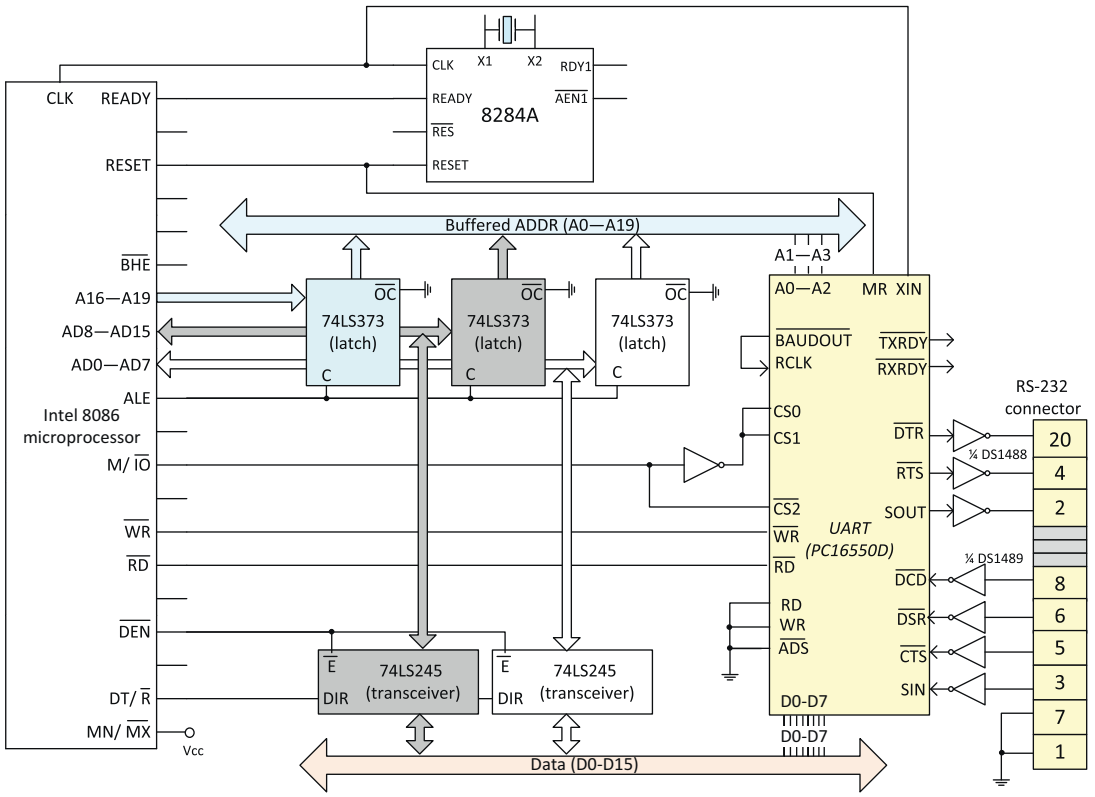

Figure 3.12
Intel 8086 memory I/O mapping.

The processor gets its clock by connecting its CLK input to the 8284A's CLK output. 8284A also has an input pin $\overline{\text{RES}}$, which is used to generate a reset signal through the output pin RESET to reset the 8086 processor. The input pin RDY1 is a bus-ready signal; it is typically driven by memory or I/O devices, acknowledging that data have been received or are available on the data bus. RDY1 is validated by $\overline{\text{AEN1}}$, which is connected to the ground when RDY1 is in use. The input signal RDY1 is synchronized by the 8284A clock generator to generate an output signal READY, which in turn is connected to the READY input of the 8086 processor, indicating the completion of data transfer.

3.3.2.2 External bus

Since AD15-AD0 are multiplexed to carry both data and address information, three 8-bit 74LS373 chips are used to implement an address latch. The address latch enable signal ALE is active (high) during T1 of any bus cycle. ALE is used to enable the three 74LS373 chips so that the address information is captured by the address latch during T1 and remains available on the buffered bus (A19-A0) until the next time ALE transits to high.

AD15-AD0 and the external data bus D15-D0 are connected through a bus transmitter/receiver implemented by two 8-bit 74LS245 chips. The direction input DIR controls transmission of data: from AD15-AD0 to D15-D0 when it is high (say, in a write operation), and from D15-D0 to AD15-AD0 when it is low (say, in a read operation). DIR is driven by the data transmit/receive signal DT/$\overline{\text{R}}$,[9] which is valid in the T4 preceding a bus cycle and remains valid until the final T4 of the bus cycle.

The 74LS245 chip has an enable input $\overline{\text{E}}$; the two buses are isolated when $\overline{\text{E}}$ is high. $\overline{\text{E}}$ is driven by the data enable output signal $\overline{\text{DEN}}$, which, for a read or interrupt acknowledge cycle, is active from the middle of T2 until the middle of T4, while for a write cycle it is active from the beginning of T2 until the middle of T4.

3.3.2.3 I/O device: UART

The processor is wired with a PC16550D UART chip for serial transmission of digital information (bits) through a single wire. The UART performs serial-to-parallel conversion on data characters received from a peripheral device or a modem, and parallel-to-serial conversion on data characters received from the processor.

PC16550D is mainly composed of a programmable baud rate generator, a transmitter, a receiver, and several registers (e.g., 16-bit divisor register) that control UART operations, including transmission and reception of data. Note that the UART chip has three address signals (A0, A1, A2) and eight data signals (D7-D0). Refer to its data sheet [9] for other pin descriptions.

---

[9] $\overline{\text{S1}}$ is used instead in the maximum mode.

In Figure 3.12, pins $\overline{RD}$ and $\overline{WR}$ of the UART are connected to the corresponding pins of the processor. This allows the processor to transmit data to, and receive data from, the UART. Also, the UART chip is selected only when the M/$\overline{IO}$ signal is low; this indicates that the registers of the UART are mapped into the I/O address space rather than the memory space.

3.3.3 Memory Address Space

Figure 3.13 shows a configuration where the Intel 2142 memory chips are mapped into the memory address space. This is because the memory chips are readable or writable only when M/$\overline{IO}$ is high. Specifically, the write enable signal $\overline{WE}$ of the memory chips is from $\overline{MEMW}$, which is asserted when M/$\overline{IO}$ is high and $\overline{WR}$ is low, where $\overline{WR}$ indicates that the processor is performing a write cycle. Similarly, the output disable signal OD of the memory chips is from $\overline{MEMR}$, which is asserted low when M/$\overline{IO}$ is high and $\overline{RD}$ is low, where $\overline{RD}$ indicates that the processor is performing a read cycle.

A 74LS138 1-of-8 decoder/demultiplexer is used to produce chip selection signals. Its truth table is given in Table 3.3. Since the output signal $\overline{O_0}$ is used in Figure 3.13, the memory chips are enabled only when [A18A17A16] = [000].

The 8086 processor is wired with a high memory bank and a low memory bank, each composed of two Intel 2142 chips. A 2142 chip is a 1024 × 4-bit SRAM. The chip select signal $\overline{CS}$ of the high bank is asserted (active low) whenever $\overline{O_0}$ and $\overline{BHE}$ are low. In such a case, data are written to or read from the high bank through bus lines D8-D15. The chip select signal $\overline{CS}$ of the low bank is asserted (active low) whenever $\overline{O_0}$ and A0 are low. In such a case, data are written to or read from the low bank through bus lines D0-D7. In other words, byte data with an even address (A0 = 0) are read from or written to the low memory bank through

Table 3.3 74LS138 truth table

A	B	C	$\overline{O_0}$	$\overline{O_1}$	$\overline{O_2}$	$\overline{O_3}$	$\overline{O_4}$	$\overline{O_5}$	$\overline{O_6}$	$\overline{O_7}$
0	0	0	L	H	H	H	H	H	H	H
0	0	1	H	L	H	H	H	H	H	H
0	1	0	H	H	L	H	H	H	H	H
0	1	1	H	H	H	L	H	H	H	H
1	0	0	H	H	H	H	L	H	H	H
1	0	1	H	H	H	H	H	L	H	H
1	1	0	H	H	H	H	H	H	L	H
1	1	1	H	H	H	H	H	H	H	L

D0-D7, while byte data with an odd address ($\overline{\text{BHE}}$ is asserted as A0 = 1) are read from or written to the high memory bank through D8-D15.

A0 is also called a bank selection signal since it is reserved for that purpose. Care must be taken to ensure that each register within an 8-bit peripheral device located on the lower portion of the data bus be addressed as even. Now, you should understand why in Figure 3.12 it is address lines A1-A3, rather than A0-A2, that are connected to pins A0-A2 of the UART. If A0 of the address bus were used, some of the UART registers would become inaccessible because the UART is wired only to the low portion of the data bus.

3.3.4 Wait States

Normally, a bus cycle (e.g., accessing a memory or I/O port) on an 8086 processor is composed of four clock cycles—T1 to T4. Wait states, called Tw, can be inserted in a bus cycle to help the processor to interface with slow memory or I/O devices. The READY input signal on the 8086 is used to request the processor to stretch out its read or write cycle, to accommodate slow devices. In particular, the 8086 processor samples the READY line at the rising edge of T3. If READY is low, a Tw state is inserted after T3. A "wait" state is of the same duration as a clock cycle. During the Tw state, the READY is sampled again at the next rising edge of the clock, and another Tw is inserted if READY is still low. Whenever READY is sampled high at the rising edge of T3 or Tw, the T4 clock cycle follows.

In Figure 3.13, the processor's READY pin is driven by the READY output of the 8284A clock generator, which, in turn, is synchronized with its RDY1 input. A memory or I/O device can initiate WAIT state generation by bringing RDY1 low.

74LS164 used in Figure 3.13 is an 8-bit shift register, where the two serial inputs A and B are connected to voltage (steady high). When the clear signal $\overline{\text{CLR}}$ is asserted, all the outputs (Q_A, ..., Q_H) become low. When $\overline{\text{CLR}}$ is not active, Q_A becomes high on the low-to-high transition of the clock input. The outputs Q_B, ..., Q_H, respectively, take the level of Q_A, ..., Q_G before the most recent low-to-high transition of the clock, indicating a 1-bit shift every clock.

Now, let us examine the time diagram in Figure 3.14 to see how the 74LS164 configuration given in Figure 3.13 works.

First, note that the bus cycle begins in T1 with the assertion of the address latch enable signal ALE. The trailing edge of this signal is used to latch the address information, which is valid on the buffered address bus at this time. At T2 the address is removed from the local bus, which goes to a high impedance state.

The 74LS164 shift register is cleared whenever $\overline{\text{INTA}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ are all high. In Figure 3.14 this is indicated by the two shaded areas bounded by the level changes of $\overline{\text{RD}}$.

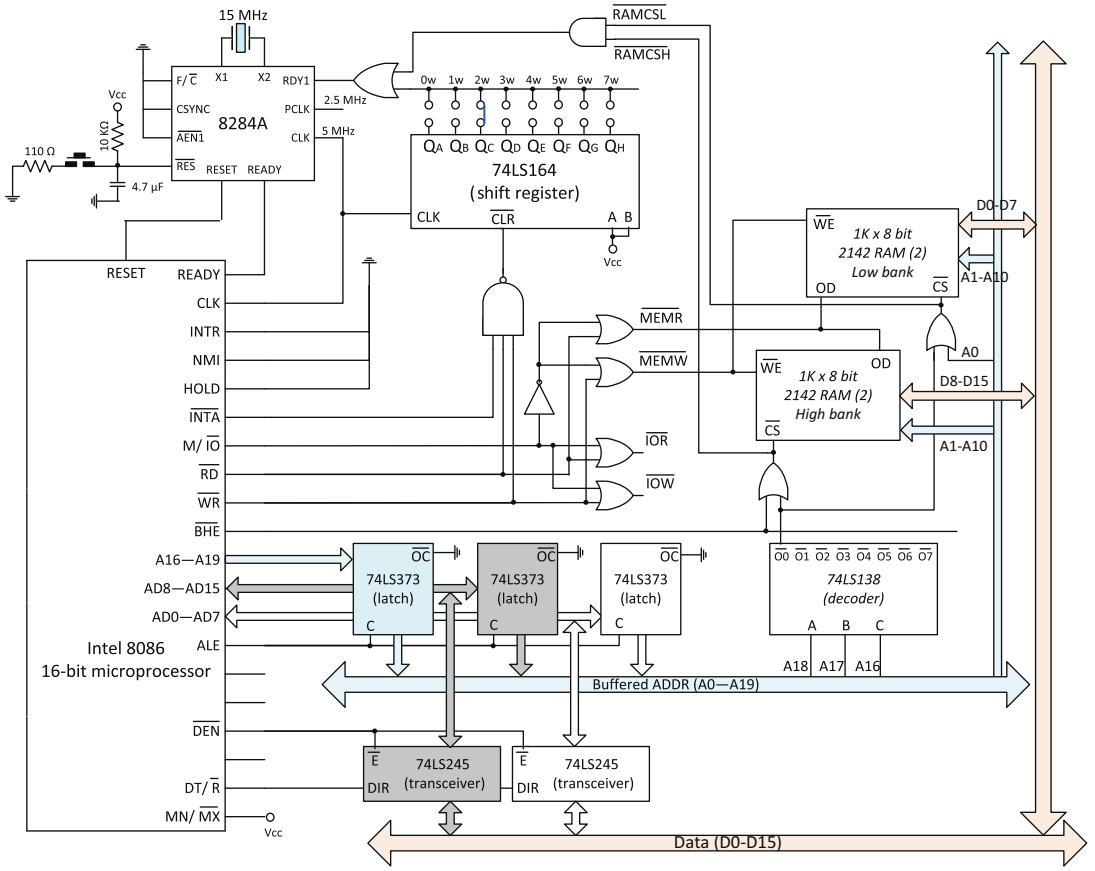

Figure 3.13
Intel 8086 memory mapping.

The read control signal $\overline{RD}$ is asserted at T2.[10] Whenever $\overline{INTA}$, $\overline{RD}$, or $\overline{WR}$ is asserted, the $\overline{CLR}$ input of the 74LS164 shift register is driven high, and the 74LS164 shift register becomes ready to drive its outputs. The 74LS164 shift register drives Q_A high when it comes to the rising edge of the next clock, which is the T2 cycle. Q_A remains high until the 74LS164 shift register is cleared. One clock cycle later, Q_B is driven high, taking the level of Q_A; one more clock cycle later, Q_C is driven high, taking the level of Q_B.

To ensure that the 8086 timing requirements are met, the device needs to bring RDY1 low prior to the rising edge of the 8086's T2 clock. As indicated in Figure 3.14, both $\overline{RAMCSH}$ and $\overline{RAMCSL}$, which are simply the select signals of the memory chips, are low in clock cycle T1.

[10] $\overline{WR}$ and $\overline{RD}$ are active for T2, T3, and Tw of any bus cycle.

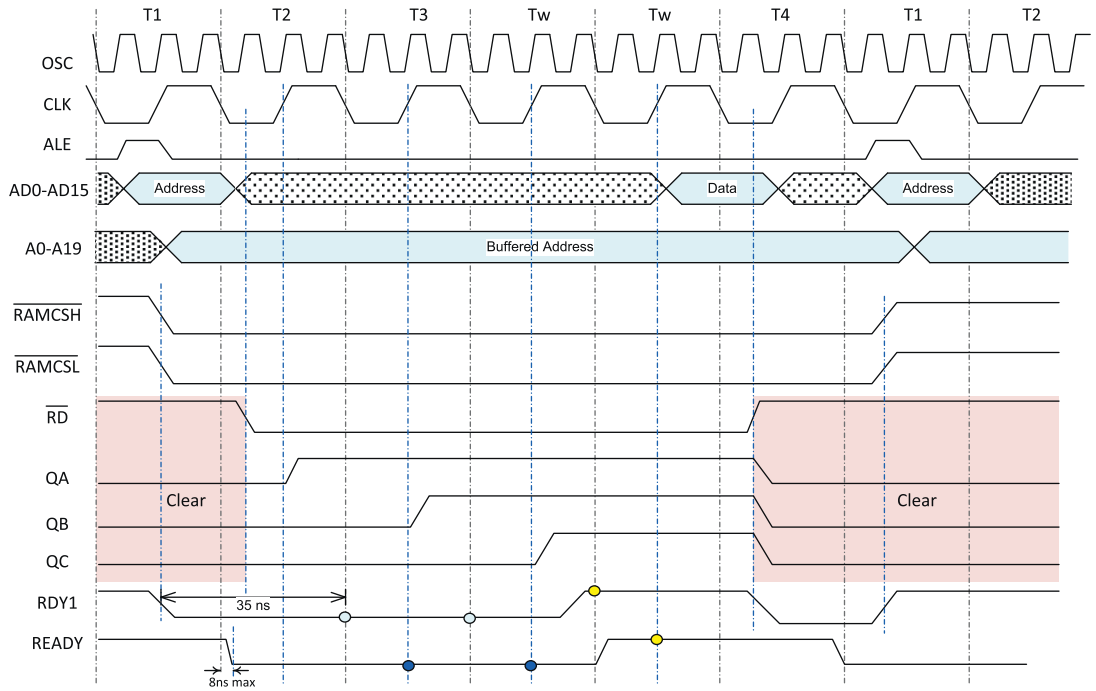

Figure 3.14
Intel 8086 timing diagram.

Since Q_C of the 74LS164 shift register is also low in cycle T1, RDY1 is driven low, which indicates the "not ready" status of the memory devices.

The 8284A clock generator drives the 8086's READY signal low at the falling edge of T1. The 8086 processor samples the READY line at the rising edge of T3. In our case, it finds that READY is low, and it inserts a Tw clock cycle after T3. The 8086 processor samples the READY line again at the rising edge of Tw, and inserts another Tw clock cycle after the first Tw.

Another timing constraint is that whenever a new Tw cycle is no longer needed, the memory device has to bring RDY1 high early in T3 or Tw so that the 8284A clock generator can bring READY high before the rising edge of T3 or Tw. In order to bring RDY1 high, the 74LS164 shift register has to drive Q_C high because only Q_C is wired to the OR gate. Q_C is driven high at the rising edge of the first Tw, which is two clock cycles later than when Q_A changed to high. The 8284A clock generator then starts to drive the 8086's READY signal high at the falling edge of the first Tw. This happens before the 8086 processor samples the READY line at the rising edge of the second Tw. As a result, no more Tw cycle is added before the T4 clock cycle.

3.4 Intel Pentium

The Intel Pentium series is the fifth generation of the x86 microprocessor family. The Pentium series is positioned as Intel's mid-range processor series, in between the Celeron and Core series.

The Pentium processor is a 32-bit processor with a 64-bit data bus. Its pin diagram is given in Figure 3.15, where the pins are grouped according to their functions.

The Pentium processor can initiate the following bus cycles:

- Single-transfer cycle: the transfer of one data item (up to 8 bytes or 64 bits).
  - Interrupt acknowledge cycle: The Pentium processor has 256 possible interrupt vectors. It generates interrupt acknowledge cycles in response to maskable interrupt requests.
  - Special bus cycle: It is generated in response to certain instructions (e.g., the halt instruction is executed).
  - Read cycle: It is generated by the processor to read data (or code) from memory or I/O devices.
  - Write cycle: It is generated by the processor to write data to memory or I/O devices.
- Burst-transfer cycle: the transfer of four data items (up to 32 bytes or 256 bits).
  - Burst read cycle: 32 bytes are read consecutively from memory into the internal cache in one bus cycle.
  - Burst writeback cycle: Modified lines in the processor's data cache are written back to memory in blocks of 32 bytes.

The pins that are pertinent to the bus operations are described below (interested readers can refer to the data sheet [3] to get more information):

- A31-A3: Address lines. As outputs, A31-A3 along with $\overline{\text{BE7}}$-$\overline{\text{BE0}}$ define the physical address space (memory or I/O).
- $\overline{\text{ADS}}$: Output signal for address status. When asserted, it indicates that a new valid bus cycle is currently being driven by the processor.
- AHOLD: Input signal for address hold. An external system can initiate an *inquire cycle* to a Pentium processor to check whether a particular address location is cached in the processor's internal cache, and if so, what state it is in. To prepare for an inquire cycle, an external system first needs to assert AHOLD to force the Pentium processor to float its address bus,[11] then wait two clocks for the processor to finish housekeeping (so that data can be returned or driven for previously issued bus cycles).

[11] A signal is floating if no part on the circuit is driving it.

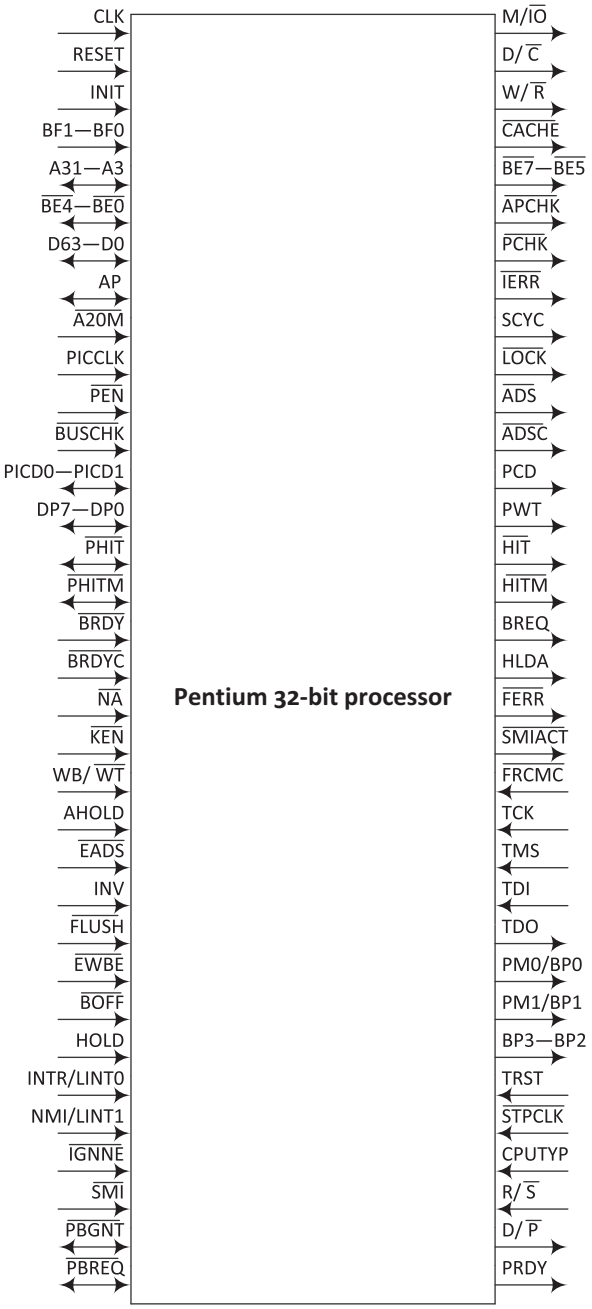

Figure 3.15
Intel Pentium 32-bit processor.

- $\overline{\text{BE7}}$-$\overline{\text{BE0}}$: Byte enable signals. $\overline{\text{BE7}}$-$\overline{\text{BE0}}$ are driven in the same clock as address lines A31-A3, used to determine which bytes are transferred (read or written) by the processor for the current bus cycle.
- $\overline{\text{BOFF}}$: Input signal for backoff. It is used to abort all outstanding bus cycles that have not yet finished. When it is asserted low, the processor will float all pins in the next clock and maintain its bus in this state until $\overline{\text{BOFF}}$ is deasserted, at which time the processor will restart the aborted bus cycle(s).
- $\overline{\text{BRDY}}$: Input signal for burst ready. It indicates that the external system has presented valid data on the data pins in response to a read request or that the external system has accepted the processor's data in response to a write request.
- $\overline{\text{CACHE}}$: Output signal indicating the internal cacheability of a bus cycle.
- $\overline{\text{KEN}}$: Input signal for cache enable. When the processor generates a read cycle that can be cached ($\overline{\text{CACHE}}$ asserted) and $\overline{\text{KEN}}$ is active, the cycle will be transformed into a burst read cycle.
- D/$\overline{\text{C}}$: Output signal for data/code indication. It is driven valid in the same clock as the $\overline{\text{ADS}}$ signal is asserted. The current cycle is for data when D/$\overline{\text{C}}$ is high, and is for code or special bus cycles when D/$\overline{\text{C}}$ is low.
- D63-D0: Data lines. Lines D7-D0 define the LSB of the data bus; lines D63-D56 define the MSB of the data bus.
- $\overline{\text{EADS}}$: Input signal for external address status. It indicates that an external system has initiated an *inquire cycle* to the Pentium processor, and a valid inquire address has been driven onto the processor's address pins by the external system.
- $\overline{\text{HITM}}$: Output signal for hit to a modified line. It is asserted to indicate to the external system that the cache of the processor contains the most current copy of the data and any device needing to read that data should wait for the processor to write it back. $\overline{\text{HITM}}$ remains asserted until two clocks after the last $\overline{\text{BRDY}}$ of the writeback cycle is asserted.
- HOLD: Input signal for bus hold request. When it is asserted by another local bus master, the processor will float most of its output pins. After completing all the outstanding bus cycles, the processor will also assert HLDA (bus hold acknowledge) to relinquish the bus to the requester. The processor will maintain its bus in this "bus-hold" state until HOLD is deasserted. In such a case, the processor will deassert HLDA and resume driving the bus.
- $\overline{\text{NA}}$: Input signal for next address. When $\overline{\text{NA}}$ is asserted, it indicates that the external memory system is ready to accept a new bus cycle although all data transfers for the current cycle have not yet finished. The processor will issue $\overline{\text{ADS}}$ for a pending cycle two clocks after $\overline{\text{NA}}$ is asserted.
- RESET: It forces the Pentium processor to restart execution at a known state.

3.4.1 Bus State Transition

We are now ready to examine the state transitions of Pentium bus cycles. A *bus cycle* begins with the processor driving an address and status and asserting $\overline{\text{ADS}}$, and ends when the last $\overline{\text{BRDY}}$ is returned.

The Pentium processor supports up to two outstanding bus cycles. Each bus cycle is composed of several states (clock cycles). Below is a list of valid bus states for the Pentium processor:

- Ti: This is the *bus idle* state. An asserted $\overline{\text{BOFF}}$ or RESET will always force the bus back to this state. HLDA will be driven only in this state.
- T1: there is only one outstanding bus cycle, and this is the *first clock* of that bus cycle. Valid address and status are driven out and $\overline{\text{ADS}}$ is asserted.
- T2: There is only one outstanding bus cycle, and this is the *second and subsequent clock* of the bus cycle. In state T2, data are driven out (if the cycle is a write cycle), or data are expected (if the cycle is a read cycle), and the $\overline{\text{BRDY}}$ pin is sampled. A bus cycle may have multiple T2 states.
- T12: There are two outstanding bus cycles. While the second (newer one) bus cycle is in its T1 state the first bus cycle is in its T2 state. This implies that (a) for the second outstanding bus cycle, the processor drives the address and status and asserts $\overline{\text{ADS}}$, and (b) for the first outstanding cycle, data are transferred and $\overline{\text{BRDY}}$ is sampled.
- T2P: There are two outstanding bus cycles, and both are in their respective T2 states. In T2P, data are being transferred and $\overline{\text{BRDY}}$ is sampled for the first outstanding cycle.
- TD: There is one outstanding bus cycle. Write cycles can be pipelined into read cycles and read cycles can be pipelined into write cycles, but one dead clock is required between consecutive read and write cycles to allow bus turnover. The processor enters TD if (a) a dead clock is needed, and (b) in the previous clock there were two outstanding cycles, and $\overline{\text{BRDY}}$ was sampled active (low) for the first bus cycle.

The possible bus state transitions are described in detail in the UML state diagram given in Figure 3.16 (state diagram notations are covered in Chapter 9).

First, the notion of request pending, *ReqPend*, is defined as a Boolean expression:

$$ReqPend \triangleq P \wedge \neg \text{HOLD} \wedge \overline{\text{BOFF}} \wedge (\neg \text{AHOLD} \vee \neg \overline{\text{HITM}}),$$

where the proposition P states that the processor has generated a new bus cycle internally. Note that the term $\overline{\text{BOFF}}$ says that $\overline{\text{BOFF}}$ is *not* asserted (i.e., $\overline{\text{BOFF}}$=1).

A composite state named "Active" is used to encapsulate the nonidle states. The bus stays within the idle state Ti if there is no request pending. Whenever the processor has generated a new bus cycle (*ReqPend* is true), in the next clock (i.e., as the current clock cycle expires) the bus transits from Ti to T1. The bus will transit back to Ti upon the next clock if:

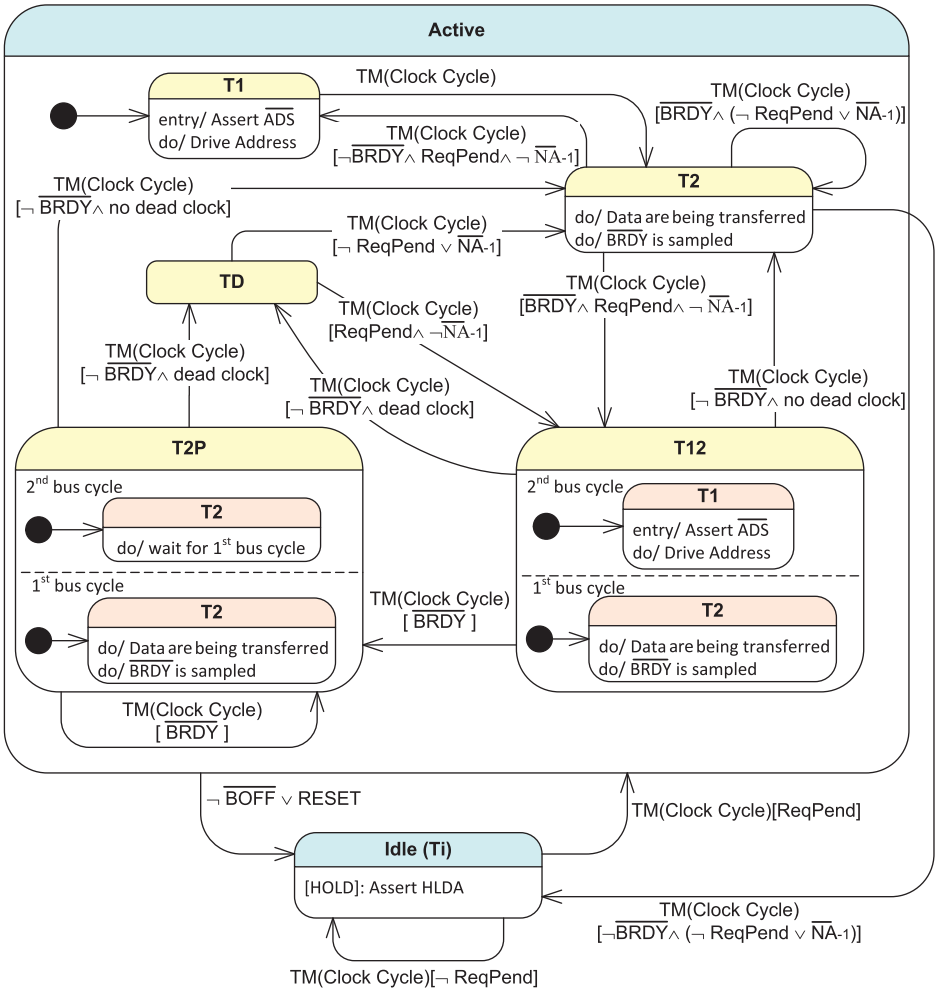

Figure 3.16
Pentium bus control state machine.

- in the current clock, $\overline{\texttt{BOFF}}$ is asserted (low) or RESET is asserted (high); or
- the current clock is T2, the data transfer has finished ($\overline{\texttt{BRDY}}$ was sampled low), and there is no request pending or $\overline{\texttt{NA}}_{-1}$ is high (not asserted). Note here that $\overline{\texttt{NA}}_{-1}$ represents the logical level of $\overline{\texttt{NA}}$ in the clock before this T2.

In a well-designed state diagram, if a state has multiple outgoing transitions triggered by the same event, their guard conditions—say, $C_1, C_2, \ldots, C_k$—ought to be able to form a complete Boolean expression; that is, $C_1 \vee C_2 \vee \cdots \vee C_k = 1$. The state diagram in Figure 3.16 is well designed for the following reasons:

- The guard conditions of the four transitions leaving state T2 are complete. That is, $[\overline{\text{BRDY}} \wedge (\neg \mathit{ReqPend} \vee \overline{\text{NA}}_{-1})] \vee [\overline{\text{BRDY}} \wedge \mathit{ReqPend} \wedge \neg\overline{\text{NA}}_{-1})] \vee [\neg\overline{\text{BRDY}} \wedge (\neg \mathit{ReqPend} \vee \overline{\text{NA}}_{-1})] \vee [\neg\overline{\text{BRDY}} \wedge \mathit{ReqPend} \wedge \neg\overline{\text{NA}}_{-1})] = 1$.
- The guard conditions of the three transitions leaving state T12 are complete.
- The guard conditions of the four transitions leaving state T2P are complete.
- The guard conditions of the two transitions leaving state TD are complete.

Also note that both T12 and T2P contain two parallel sessions, owing to the existence of two outstanding bus cycles. In T2P, the second bus cycle is waiting for the first bus cycle to finish its data transfer.

Let us use some timing diagrams to examine how the bus changes its states as the timing clock goes on.

Figure 3.17 shows a read cycle followed by an idle state, then a write cycle. Notice that $\overline{\text{ADS}}$ and W/$\overline{\text{R}}$ are asserted in clock T1; address lines are also driven in clock T1. $\overline{\text{BRDY}}$ is sampled in clock T2 to check whether data transfer has finished or not. The transition from T2 to Ti happens because the guard condition $\neg\overline{\text{BRDY}} \wedge (\neg \mathit{ReqPend} \vee \overline{\text{NA}}_{-1})$ is true.

Figure 3.18 shows a read cycle followed by an idle state, then a write cycle. Notice that the read cycle takes two T2 clocks and the write cycle takes three T2 clocks. This can happen when the processor interfaces with a slow memory device.

Figure 3.19 shows a burst read cycle. Notice that $\overline{\text{CACHE}}$ is driven active (low) in clock T1. The read cycle is transformed into a multiple-transfer burst cycle by $\overline{\text{KEN}}$ being asserted active on the clock on which the first active $\overline{\text{BRDY}}$ is sampled. Consequently, the processor is able to read one data item (i.e., 8 bytes or 64 bits) in each T2 clock. It may take more than one T2 clock to read one data item when the processor interfaces with slow memory.

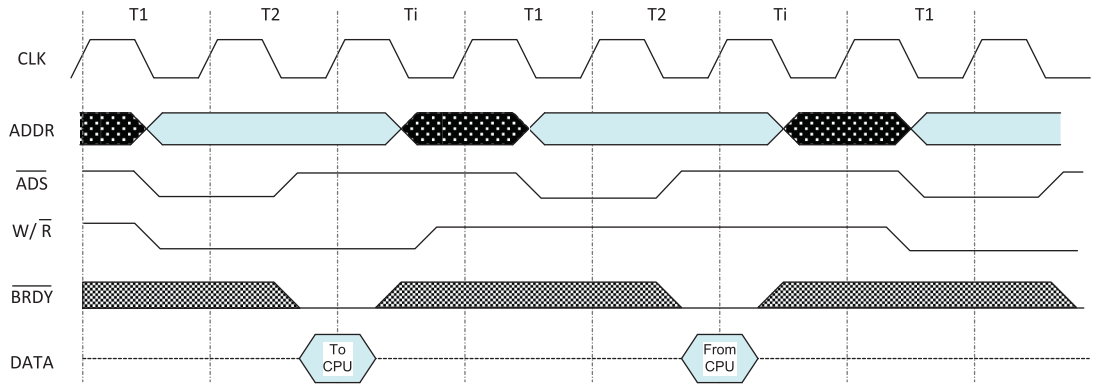

Figure 3.17
Pentium timing diagram example.

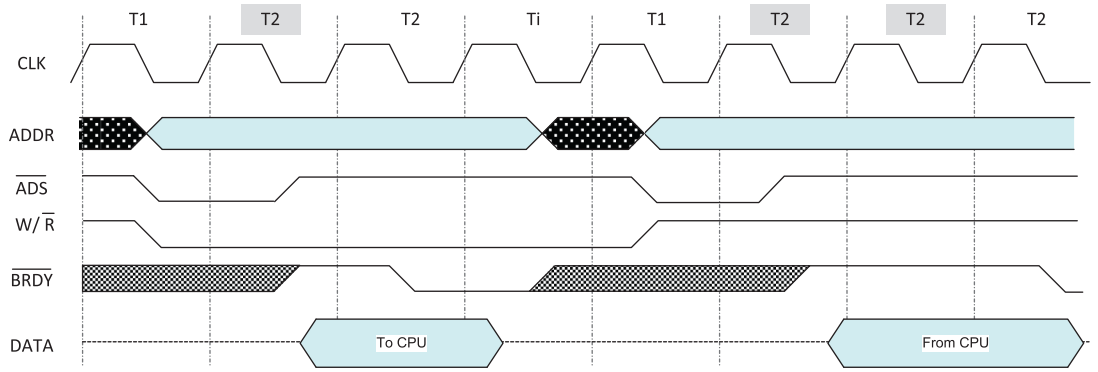

Figure 3.18
Pentium timing diagram: wait states.

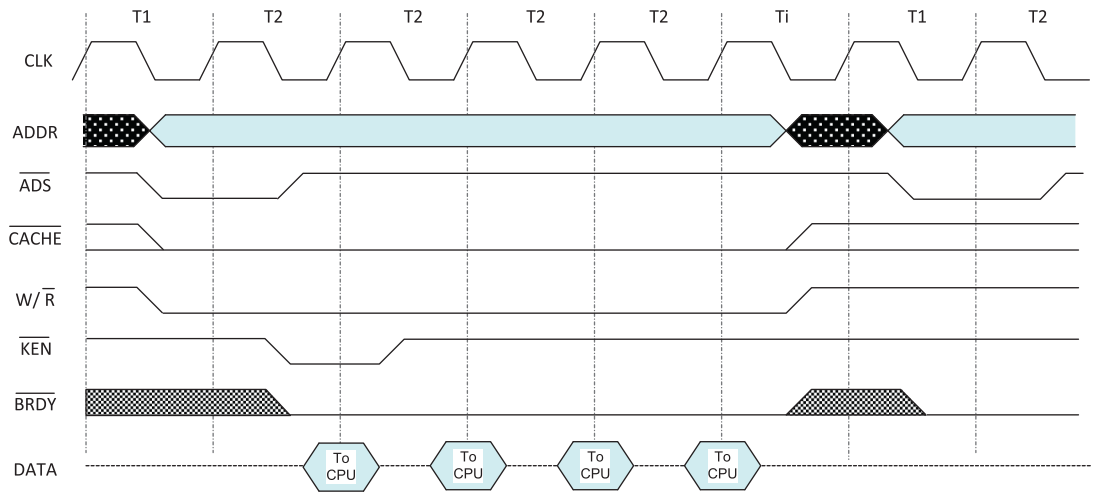

Figure 3.19
Pentium timing diagram: burst read.

Figure 3.20 shows the *pipelining* of a burst read cycle and a write cycle. The first bus cycle starts as a read cycle. $\overline{\text{BRDY}}$ is sampled high in the first T2 clock, indicating that another T2 clock is needed to finish the data transfer. $\overline{\text{BRDY}}$ is sampled active in the second T2 clock, indicating that one data item has been transferred. However, $\overline{\text{KEN}}$ is asserted in the second T2 clock, which transforms the read cycle into a burst read cycle.

Also notice that $\overline{\text{NA}}$ is asserted in the second T2 clock, indicating that the external memory system is ready to accept a new bus cycle although all data transfers for the current read cycle have not yet finished. The processor will issue $\overline{\text{ADS}}$ for a pending cycle two clocks later.

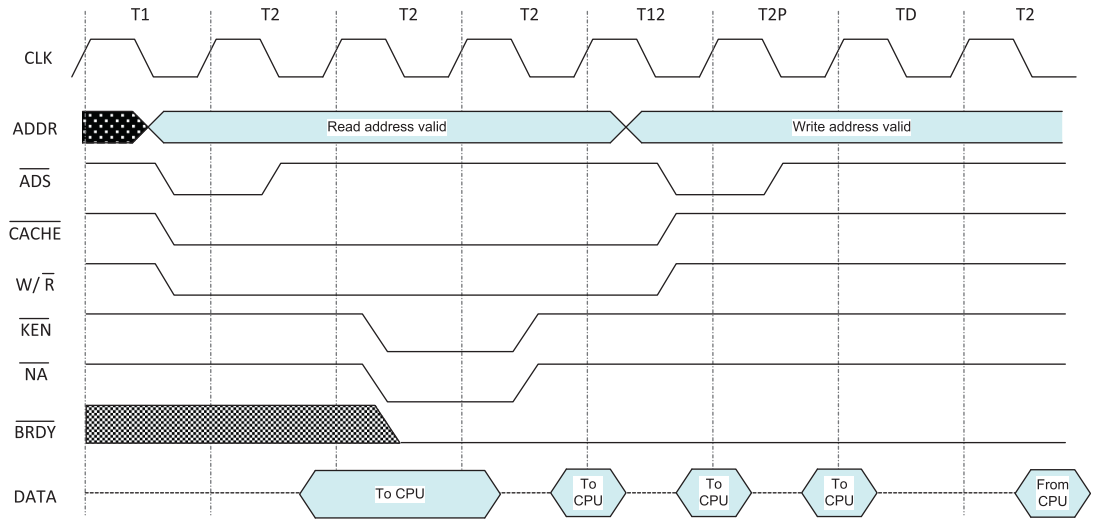

Figure 3.20
Pentium timing diagram: pipelining.

$\overline{\text{BRDY}}$ is sampled low in the third T2 clock, indicating that another data item has been transferred. Also, the assertion of $\overline{\text{NA}}$ in the second T2 clock makes $\neg\overline{\text{NA}}_{-1}$ true in the third T2 clock, which enables the transition from state T2 to state T12.

In the T12 clock, $\overline{\text{BRDY}}$ is sampled low, indicating that another data item has been transferred for the burst read cycle. During the same clock, the processor initiates a new write cycle, driving address, status, and $\overline{\text{ADS}}$ for this second outstanding bus cycle.

In the T2P clock, $\overline{\text{BRDY}}$ is sampled low, indicating that the fourth data item has been transferred for the burst read cycle. Up to this point, the burst read cycle has finished. Upon the next clock, the bus transits from the T2P state to the TD state, because in pipelining a dead clock is needed between read and write cycles. After the TD clock, the bus is ready for transferring data for the write cycle.

3.4.2 Memory Organization

The Pentium processor has a memory space of 4 GB (2^{32} bytes) and a separate I/O space with 64 KB of addressable locations. The memory space is organized as a sequence of 64-bit quantities. Each 64-bit location has eight individually addressable bytes at consecutive memory addresses. The I/O space is organized as a sequence of 32-bit quantities. Each 32-bit quantity has four individually addressable bytes at consecutive memory addresses.

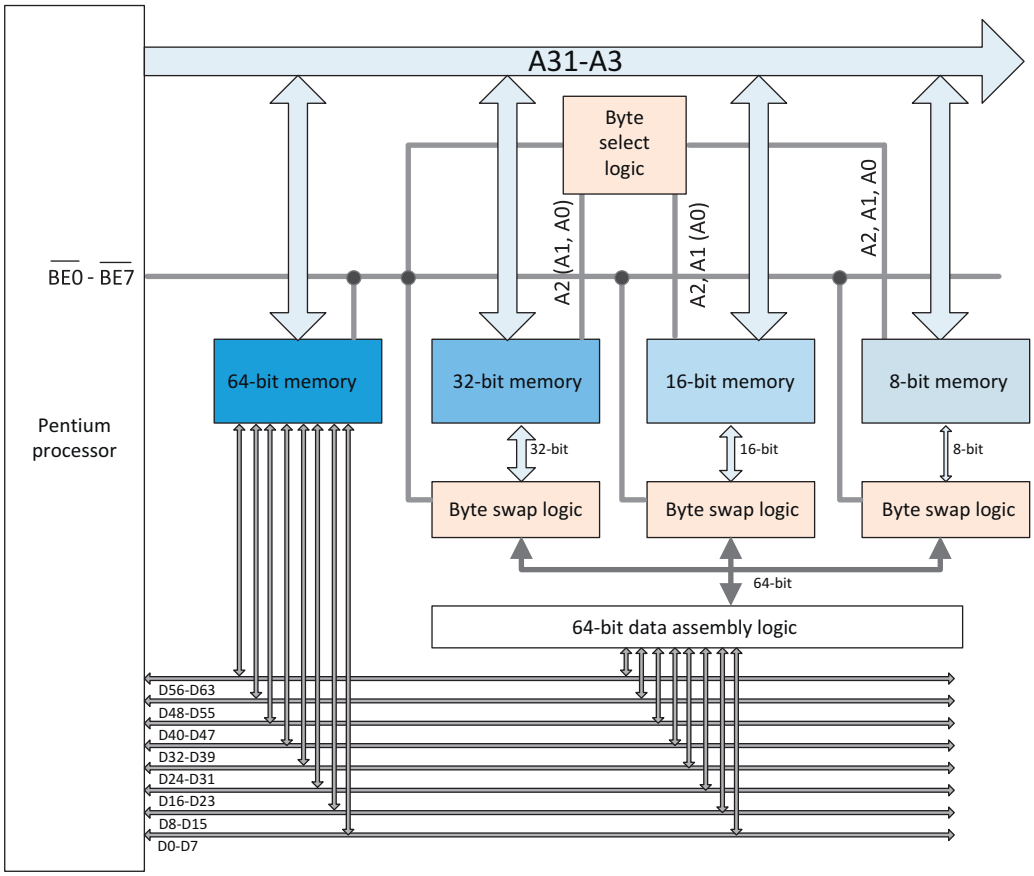

Figure 3.21
Pentium memory mapping.

Figure 3.21 illustrates how the Pentium processor interfaces with memory of various width:

- 64-bit memories are organized as arrays of quadwords (a quadword is a block of 8 bytes). Quadwords begin at addresses evenly divisible by 8. Address lines A31-A3 are used to access quadwords, and $\overline{\text{BE7}}$-$\overline{\text{BE0}}$ are used to access individual bytes within a quadword.
- 32-bit memories are organized as arrays of dwords (a dword is a block of 4 bytes). Dwords begin at addresses evenly divisible by 4. A dword can be accessed by address lines A31-A3 together with A2. A2 is not a physical address line, but is decoded from $\overline{\text{BE7}}$-$\overline{\text{BE0}}$ according to Table 3.4:

$$A2 = \overline{\text{BE0}} \wedge \overline{\text{BE1}} \wedge \overline{\text{BE2}} \wedge \overline{\text{BE3}} \wedge (\neg\overline{\text{BE4}} \vee \neg\overline{\text{BE5}} \vee \neg\overline{\text{BE6}} \vee \neg\overline{\text{BE7}}).$$

Individual bytes within a dword can be accessed by A1 and A0, which are also decoded from $\overline{\text{BE7}}$-$\overline{\text{BE0}}$ according to Table 3.4.

Table 3.4 Byte access enabled by BE7-BE0

A2	A1	A0	$\overline{BE7}$	$\overline{BE6}$	$\overline{BE5}$	$\overline{BE4}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
0	0	0	X	X	X	X	X	X	X	L
0	0	1	X	X	X	X	X	X	L	H
0	1	0	X	X	X	X	X	L	H	H
0	1	1	X	X	X	X	L	H	H	H
1	0	0	X	X	X	L	H	H	H	H
1	0	1	X	X	L	H	H	H	H	H
1	1	0	X	L	H	H	H	H	H	H
1	1	1	L	H	H	H	H	H	H	H

- 16-bit memories are organized as arrays of words (a word is a block of 2 bytes). Words begin at addresses evenly divisible by 2. A word can be accessed by address lines `A31-A3` together with `A2` and `A1`. `A0` is used to access individual bytes within a word.
- Eight-bit memories are organized as array bytes. A byte can be accessed by address lines `A31-A3` together with `A2`, `A1`, and `A0`.

Note that in Figure 3.21 external byte swapping logic and data assembly logic are needed for memory widths smaller than 64 bits so that data can be supplied to and received from the Pentium processor on the correct data pins.

3.5 ARM926EJ-S

The ARM926EJ-S processor [12], a member of the ARM9 family, is a 32-bit RISC microprocessor capable of supporting a full operating system such as Linux, Windows CE, and QNX. The ARM926EJ-S processor has the following key features:

- It supports the 32-bit ARM instruction set and 16-bit Thumb instruction set, enabling a designer to trade off between high performance and high code density.
- It has hardware support for Java bytecode execution, providing Java performance similar to that of a just-in-time compiler.
- It has logic to assist in both hardware and software debugging.
- It has memory-mapped I/O (all peripherals are memory mapped).
- It has a Harvard cached architecture with
  - a memory management unit to support virtual addressing and memory protection;
  - separate instruction and data interfaces to its level 1 memory system, which is known as tightly coupled memory (TCM).

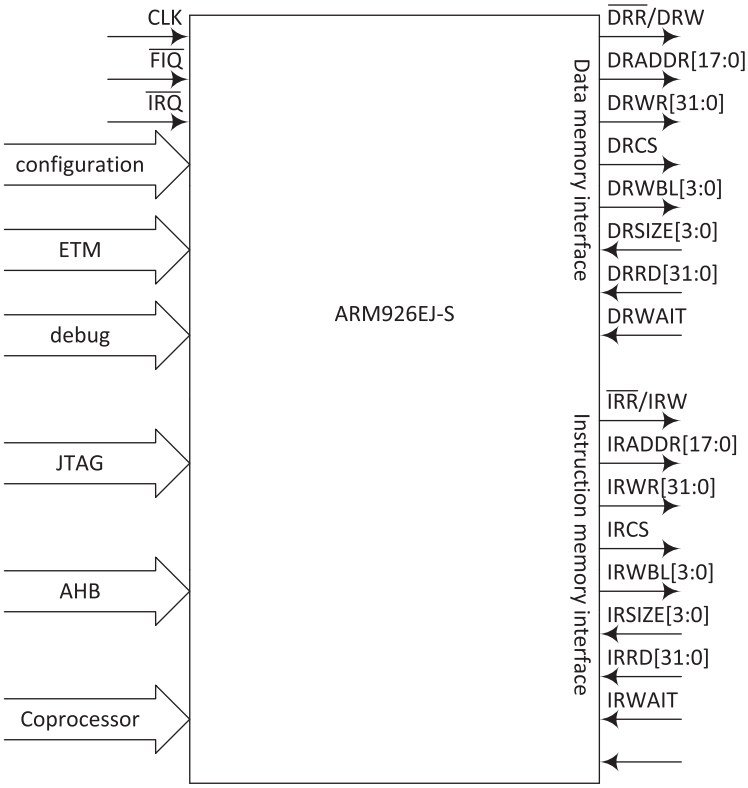

Figure 3.22
ARM ARM926EJ-S processor: pin diagram.

- an advanced microcontroller bus architecture with separate instruction and data interfaces to its level 2 memory system, where the bus is called advanced high performance bus (AHB).

Figure 3.22 gives the pin diagram of the ARM926EJ-S processor, where the pins are grouped according to their functions. It is worth noting that the ARM926EJ-S processor provides two separate interfaces to data memory and instruction memory.

3.5.1 TCM Interface

For real-time systems it is paramount that code execution is deterministic—the time taken for loading and storing instruction or data must be predictable. This is achieved in the ARM926EJ-S processor through the so-called tightly coupled memory (TCM) interfaces.

The ARM926EJ-S processor supports two TCM regions, one for instructions (ITCM) and one for data (DTCM). The physical size of the TCM regions is defined by external inputs (`IRSIZE`, `DRSIZE`), and ranges from 4 KB to 1 MB.

Some DTCM signals are described below:

- `DRADDR[17:0]`: DTCM address lines, addressing by words (a word in the ARM926EJ-S processor is 4 bytes).
- `DRCS`: chip select, indicating if an access will take place in the following cycle.
- $\overline{\texttt{DRR}}$/`DRW`: indicates if the access is a read or write.
- `DRRD[31:0]`: input data lines for TCM read.
- `DRSIZE[3:0]`: static configuration input that specifies the physical size of DTCM memory.
- `DRWBL[3:0]` DTCM byte lane indicator, indicating which bytes are to be written. Bits of DRWBL are set only when a write is taking place. In little-endian mode, `DRWBL[0]` indicates the LSB of the word and `DRWBL[3]` indicates the MSB. In big-endian mode, `DRWBL[3]` indicates the LSB of the word and `DRWBL[0]` indicates the MSB.
- `DRWD[31:0]` output data lines for TCM write.

The instruction-side TCM signals are almost identical to the DTCM signals. All the signals on the DTCM have an equivalent on the instruction side.

Figure 3.23 shows a configuration where four banks of byte-wide SRAM are used to form the DTCM region. Note that (a) the 4-byte-wide SRAMs have the same address block (by address

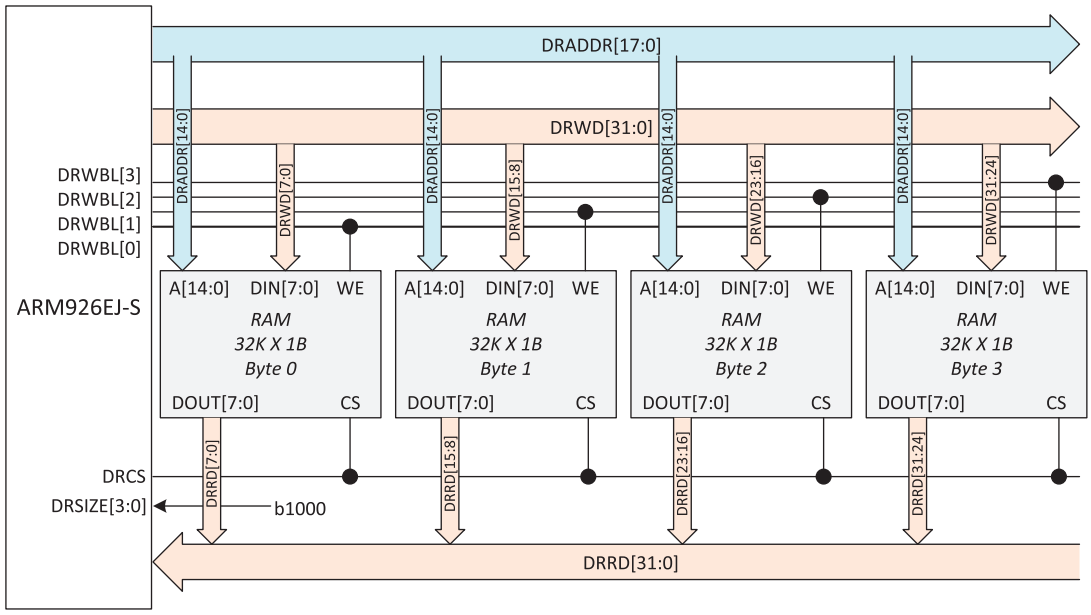

Figure 3.23
ARM926EJ-S processor: 8-bit memory.

lines DRADDR[14:0] and are selected simultaneously by the same chip-select signal;[12] (b) the chip labeled "byte 0" is connected to data input lines DRWD[7:0] and data output lines DRRD[7:0] (The other three chips are connected similarly); (c) the write-enable pin WE of each chip is controlled by a distinct line from DRWBL[3:0], so that each chip can receive only 1 byte of a word on the data bus.

Figure 3.24 shows a configuration where two-word-wide SRAMs are used to form the DTCM region. Note here that the two chips are write enabled by both DRW and address line DRADDR[14]: chip A is accessed when DRADDR[14]=0, and chip B is accessed when DRADDR[14]=1. As an exercise, figure out the address ranges of chip A and chip B.

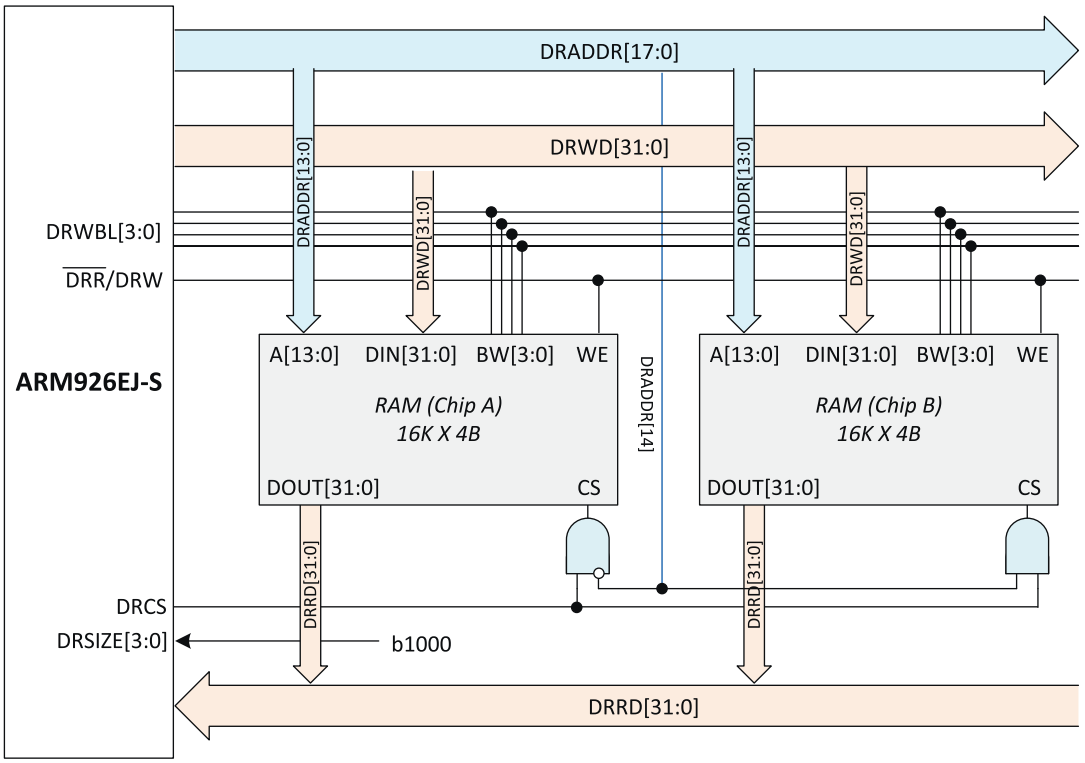

Figure 3.24
ARM926EJ-S processor: 32-bit memory.

[12] The term "memory rank" is also used to refer to a set of memory banks which are connected to the same chip select, and are therefore accessed simultaneously. In practice, the banks also share all of the other command and control signals, and only the data pins for each bank are separate. In the case here, the ARM926EJ-S processor has a 32-bit-wide data bus. If 8-bit memory chips are used, four chips are needed to form four banks.

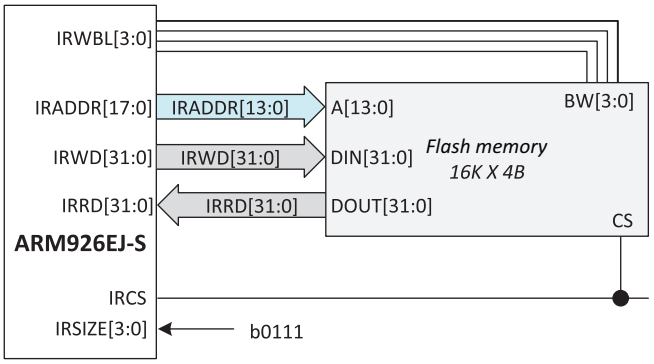

Figure 3.25
ARM926EJ-S processor: instruction.

Figure 3.25 shows a configuration where one-word-wide flash memory is used to form the instruction TCM region. Note here how the instruction-side TCM signals are used.

Problems

3.1 Refer to Figure 3.26, and list the memory address ranges for each of the four chips.
3.2 Refer to Figure 3.27, and list the memory address ranges for each of the four chips. Which chips are memory devices? Which chips are I/O devices?
3.3 Refer to Figure 3.7; which chip would be written to if in the first cycle the instruction is "TBLWT 12h to address 1A6B4Ch" and in the second cycle the instruction is "TBLWT 8Ah to address 1A6B4Dh"? Would the data 12h and 8Ah be stored on the memory chip correctly? Explain why.
3.4 Refer to Figure 3.9; which chip would the data 12h be written to? Which chip would the data 8Ah be written to? Explain why.
3.5 Refer to Figure 3.9, and follow the format of Table 3.2 to work out the memory mappings of chips A, B, and C.

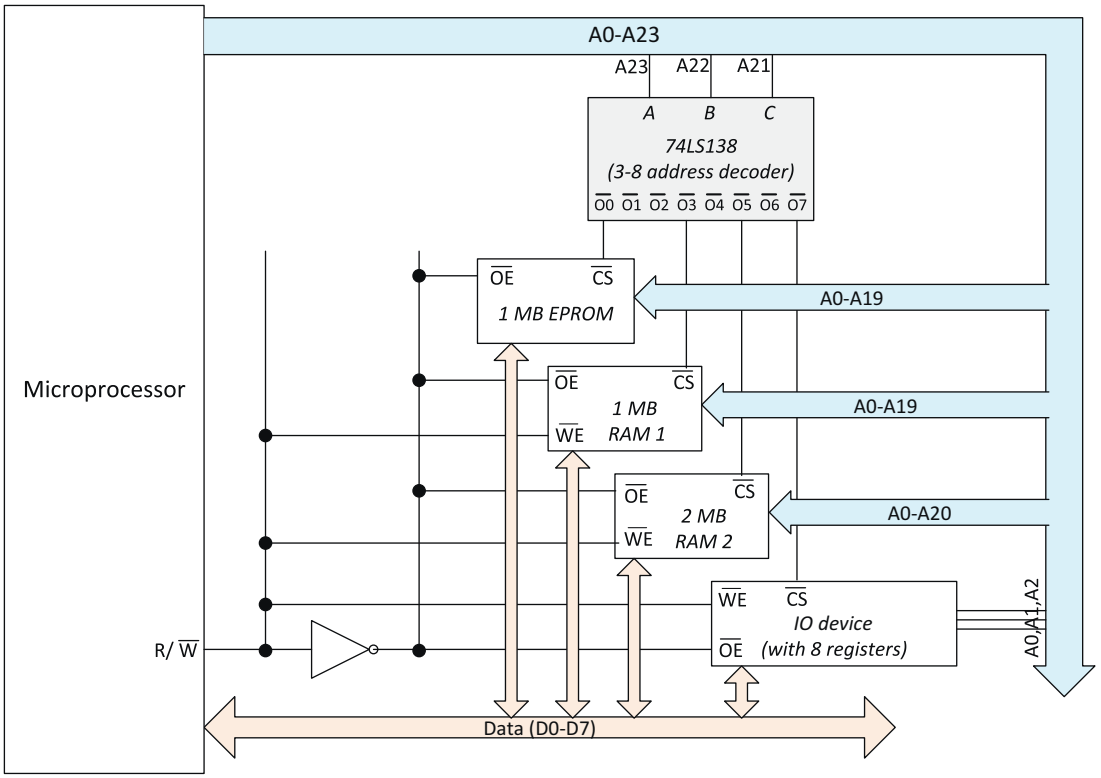

Figure 3.26
Memory-mapped I/O.

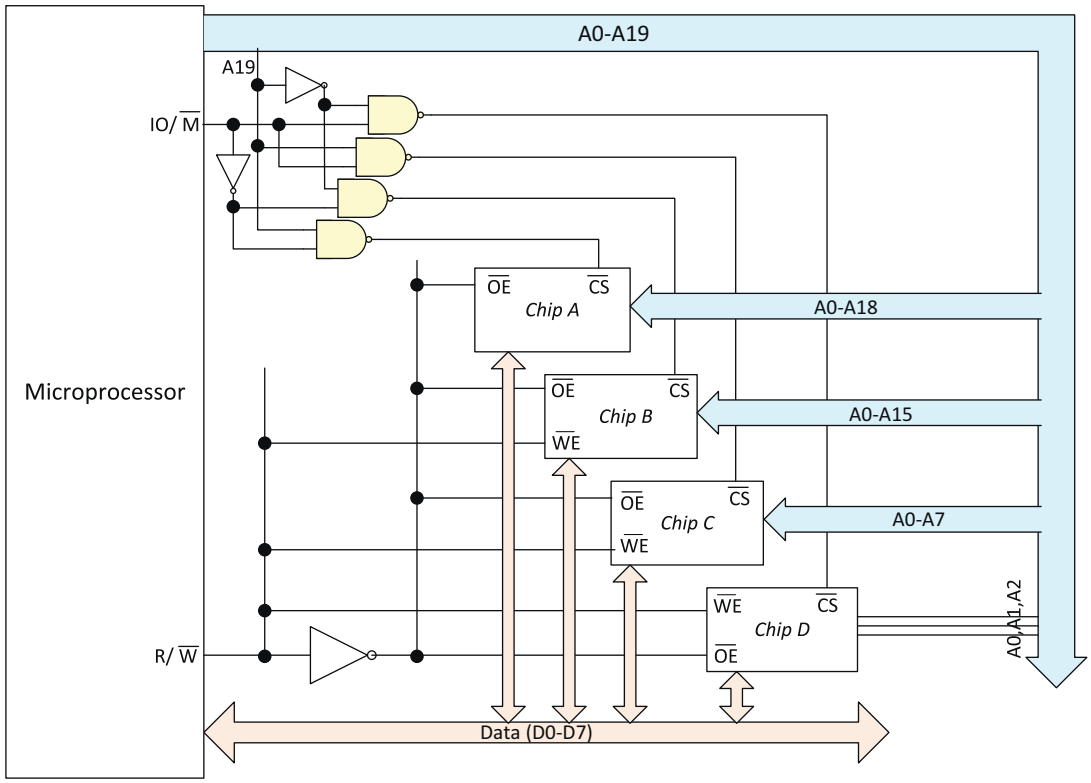

Figure 3.27
Memory mapping configuration.