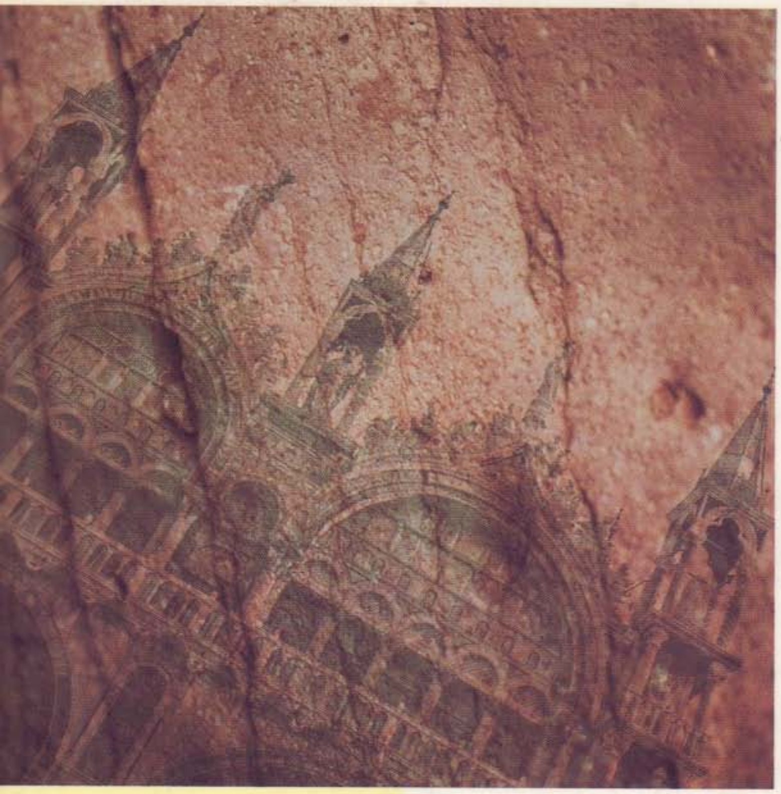
"The 'must-have' PC architecture reference set."

—*PC Magazine's* "Read Only" column

ISA System Architecture

THIRD EDITION



MINDSHARE, INC.

Tom Shanley / Don Anderson

PC SYSTEM ARCHITECTURE SERIES

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

world-class technical training

Are your company's technical training needs being addressed in the most effective manner?

MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students' time away from work and provides cost-effective alternatives. MindShare offers many flexible solutions to meet those needs. Our courses are taught by highly-skilled, enthusiastic, knowledgeable and experienced instructors. We bring life to knowledge through a wide variety of learning methods and delivery options.

training that fits your needs

MindShare recognizes and addresses your company's technical training issues with:

- Scalable cost training
- Just-in-time training
- Training in a classroom, at your cubicle or home office
- Customizable training options
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training

MindShare training courses expand your technical skillset

- PCI Express 2.0®
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals

... and many more.

All courses can be customized to meet your group's needs. Detailed course outlines can be found at **www.mindshare.com**

bringing life to knowledge.

real-world tech training put into practice worldwide

*PCI Express ® is a registered trademark of the PCISIG

MindShare Learning Options

MindShare Classroom

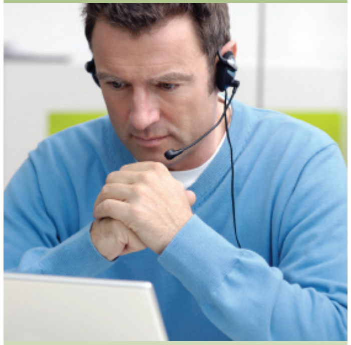
In-House Training

Public Training

Classroom Training

Invite MindShare to train you in-house, or sign-up to attend one of our many public classes held throughout the year and around the world. No more boring classes, the 'MindShare Experience' is sure to keep you engaged.

MindShare Virtual Classroom



Virtual In-House Training

Virtual Public Training

Virtual Classroom Training

The majority of our courses live over the web in an inter-active environment with WebEx and a phone bridge. We deliver training cost-effectively across multiple sites and time zones. Imagine being trained in your cubicle or home office and avoiding the hassle of travel. Contact us to attend one of our public virtual classes.

MindShare eLearning

Intro eLearning Modules

Comprehensive eLearning Modules

eLearning Module Training

MindShare is also an eLearning company. Our growing list of interactive eLearning modules include:

- **Intro to Virtualization Technology**
- **Intro to IO Virtualization**
- **Intro to PCI Express 2.0 Updates**
- **PCI Express 2.0**
- **USB 2.0**
- **AMD Opteron Processor Architecture**
- **Virtualization Technology**

...and more

MindShare Press

Books

eBooks

MindShare Press

Purchase our books and eBooks or publish your own content through us. MindShare has authored over 25 books and the list is growing. Let us help make your book project a successful one.

Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to "Bring Your Knowledge to Life." Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual class-room settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

ADAPTEC • AMD • AGILENT TECHNOLOGIES • APPLE • BROADCOM • CADENCE • CRAY • CISCO • DELL • FREESCALE
GENERAL DYNAMICS • HP • IBM • KODAK • LSI LOGIC • MOTOROLA • MICROSOFT • NASA • NATIONAL SEMICONDUCTOR
NETAPP • NOKIA • NVIDIA • PLX TECHNOLOGY • QLOGIC • SIEMENS • SUN MICROSYSTEMS SYNOPSYS • TI • UNISYS

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

4285 SLASH PINE DRIVE COLORADO SPRINGS, CO 80908 USA
M 1.602.617.1123 O 1.800.633.1440 ravi@mindshare.com

www.mindshare.com

ISA System Architecture

Third Edition

MINDSHARE, INC.

TOM SHANLEY
AND
DON ANDERSON

EDITED AND REVISED BY
JOHN SWINDLE

Addison-Wesley Publishing Company

Reading, Massachusetts • Menlo Park, California • New York

Don Mills, Ontario • Wokingham, England • Amsterdam

Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

Paris • Seoul • Milan • Mexico City • Taipei

Library of Congress Cataloging-in-Publication Data

ISBN: 0-201-40996-8

Sponsoring Editor: Keith Wollman
Project Manager: Eleanor McCarthy
Production Coordinator: Deborah McKenna
Cover design: Barbara T. Atkinson
Set in 10 point Palatino by MindShare, Inc.

1 2 3 4 5 6 7 8 9 -MA- 9998979695
First printing, February 1995

Contents

Chapter 2: Introduction to the Bus Cycle

Chapter 3: Addressing I/O and Memory

Chapter 4: The Address Decode Logic

Chapter 5: The 80286 Microprocessor

Chapter 17: Types of ISA Bus Cycles

Chapter 18: The Interrupt Subsystem

Chapter 19: Direct Memory Access (DMA)

Chapter 1

This Chapter

This chapter defines the microprocessor's role in the system, the usage of memory, and defines the role of the address, control and data buses.

The Next Chapter

The next chapter, “Introduction to the Bus Cycle,” introduces the concept of the bus cycle and defines the basic sequence of events when the microprocessor uses a bus cycle to communicate with memory or an I/O device.

Instruction Fetch and Execution

General

The microprocessor is an engine with only one task in life — to continually read instructions from memory and execute them (perform the operations specified by the instructions).

An instruction tells the microprocessor to perform one of three basic types of operations:

- Read data from an external device.
- Write data to an external device.
- Perform an internal operation that doesn't involve reading from or writing to the outside world (such as math functions).

Note — As used in this book, the term external device refers to a device external to the microprocessor chip itself. Refer to figure 1-1.

The instructions fetched (read) from memory tell the microprocessor what to do. When a microprocessor-based system is initially powered up, the microprocessor knows what address in memory to fetch (read) its first instruction from. This location is known as the power-on restart address. A group of instructions that cause the microprocessor to perform a particular task is referred to as a program. After fetching the first instruction from the restart address, the microprocessor is totally dependent on the program to tell it what to do.

In a properly functioning system, the microprocessor is never idle. At any given moment in time, the microprocessor is reading data from an external device, writing data to an external device, or executing an instruction that doesn't require that a read or write take place (for example, an instruction to add the contents of two of the microprocessor's internal registers together).

The microprocessor communicates with all external devices by reading data from them or writing data to them. The terms read and write are extremely important in any discussion of microprocessors. Always think of reading and writing from the point of view of the microprocessor rather than from that of the device the microprocessor is communicating with. The microprocessor does the reading and writing. Devices are read from and written to by the microprocessor. If you think of the terms read and write from the device's point of view, much of the information in this book will not make sense.

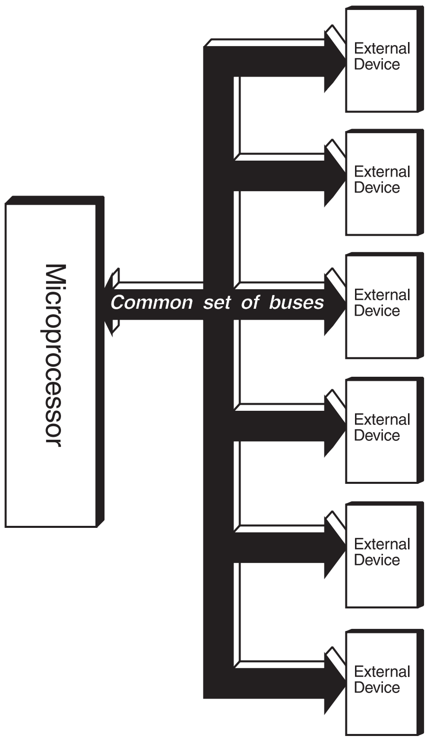


Figure 1-1. Microprocessor's Relationship to External Devices

In-Line Code Fetching

When a microprocessor fetches an instruction from a memory location and executes it, the address of the next instruction may or may not be specified as part of the currently executing instruction.

Case 1: The currently executing instruction doesn't specify the memory address of the next instruction. The microprocessor automatically assumes the next instruction is to be fetched from the next sequential memory location.

or

Case 2: The currently executing instruction specifies the memory address of the next instruction. Execution of the instruction, called a jump instruction, causes the microprocessor to alter its program flow (which is usually sequential). Rather than fetching its next instruction from the next sequential memory location, the microprocessor fetches it from the memory location specified by the instruction.

After executing a JUMP instruction, the microprocessor resumes fetching instructions from sequential memory locations until another JUMP instruction is executed. Most well-written programs do not contain an excessive number of JUMPs. Statistically then, the microprocessor is executing non-jump instructions the majority of the time. This means that the microprocessor is performing what is commonly referred to as in-line code fetches most of the time.

As an example, refer to figure 1-2. Assume that the microprocessor has just fetched the ADD instruction from location 00000h in memory. The flow of instructions fetched and executed proceeds as follows:

1. The microprocessor executes the ADD instruction fetched from location 00000h. Since it's not a JUMP instruction, the microprocessor fetches its next instruction from memory location 00001h.
2. The microprocessor executes the SUBTRACT instruction fetched from location 1. Since it's not a JUMP, the microprocessor fetches its next instruction from memory location 00002h.
3. The microprocessor executes the MOVE instruction fetched from location 2. Since it's not a JUMP, the microprocessor fetches its next instruction from memory location 00003h.
4. The microprocessor executes the JUMP 10000 instruction fetched from location 3. The JUMP instruction alters program flow. Rather than fetch its next instruction from the next sequential memory location (00004h), the microprocessor fetches it from memory location 10000h.
5. The microprocessor executes the MOVE instruction fetched from location 10000h. Since it's not a JUMP, the microprocessor fetches its next instruction from memory location 10001h.
6. And so on.

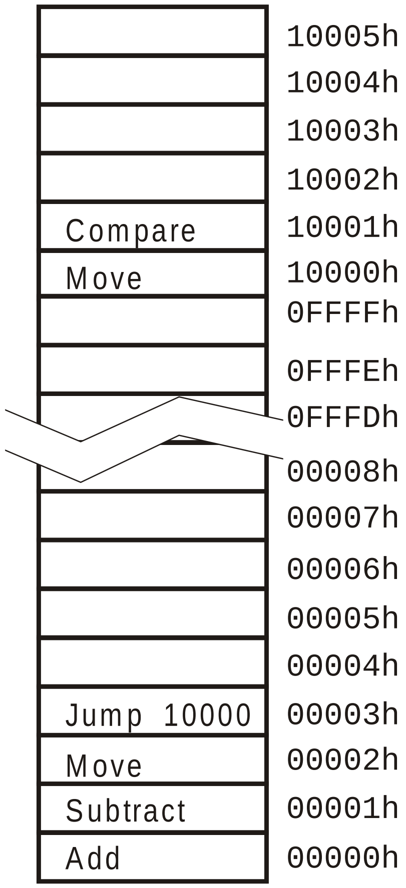


Figure 1-2. Sample Instruction Sequence

Reading and Writing

The microprocessor communicates with external devices under the following circumstances:

- To fetch (read) the next instruction from memory.
- When the currently executing instruction directs the microprocessor to read data from an external device.
- When the currently executing instruction directs the microprocessor to write data to an external device.

Chapter 2

The Previous Chapter

In "Introduction to Microprocessor Communications," the microprocessor's role in the system, usage of memory, and the role of the address, control and data buses were defined.

This Chapter

When the microprocessor must communicate with an external device, it uses its buses. The sequence of events necessary when using the buses to perform a read or write transaction is referred to as a bus cycle. This chapter introduces the microprocessor's bus unit, the concept of a state machine, and defines address time, data time and the wait state.

The Next Chapter

The next chapter, "Addressing I/O and Memory," provides a definition of an I/O device. It defines the method used by the x86 processors to distinguish memory and I/O addresses and the range of memory and I/O addresses available to the 8088, 8086, 80286, 80386, 80486 and Pentium microprocessors.

Introduction

The microprocessor's internal bus unit is a state machine that performs the required bus cycle when the microprocessor must communicate with another device. The concept of the state machine is introduced, as well as the concept of the clock, or timebase, used by the state machine to define the duration of each state.

Automatic Dishwasher – Classic State Machine Example

Any task is easier to perform when divided into logical steps. A state machine is a device (either mechanical, electronic, or software-based) designed to perform a task that can be divided into steps. Prior to starting the task, the state machine is said to be idle. When commanded to perform the task, the state machine leaves the idle state and moves through a series of steps, or states. Predefined portions of the overall task are performed during each state. The duration of each state is defined by a clock, or timebase.

In the case of the automatic dishwasher, the dishwasher is idle until you start it. A timer then begins to run, defining the duration of each state the dishwasher must pass through in order to accomplish the overall task of washing dishes. These states are:

- During the first state, the state machine wets down the dishes.
- During the second state, the dishes are soaped.
- During the third state, the dishes are rinsed.
- During the fourth and final state, the dishes are dried.
- The state machine then returns to the idle state.

Most dishwashers also have one or more switches the user can manipulate to alter the sequence of states or possibly to cause the machine to execute a particular state more than once. A perfect example would be the “Pot Scrubber” button. When pressed, this might cause the state machine to execute the “rinse” state two times instead of one.

The System Clock – a Metronome

Every x86 microprocessor has an input called CLOCK (or a similar name). This signal is produced when a voltage is applied to a crystal oscillator. It then begins to generate an electrical signal of a specific frequency. In essence, the oscillator acts as a highly accurate electronic tuning fork. The signal looks like Figure 2-1.

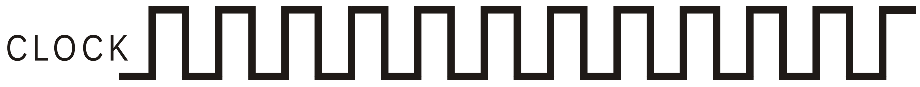


Figure 2-1. Crystal Oscillator Output

All microprocessors perform their operations in a highly organized, predefined fashion. Portions of each task are always performed during predefined time slots. The duration of each time slot is defined by the output of the crystal oscillator.

Indirectly, the 80286 and 80386 microprocessors use the CLOCK input to define the length of a time slot. They divide the frequency of the CLOCK input by two to yield an internal timebase referred to as the Processor Clock (PCLK). The CLOCK input is referred to as a double-frequency clock because its frequency is double that of the required PCLK. The 486DX microprocessor, on the other hand, uses the CLOCK input as the PCLK without dividing it. Clock-doubling or clock-multiplying processors, such as the DX4, use phase-locked loops or similar technology to multiply (instead of divide) the CLOCK signal to produce a PCLK that is faster than CLOCK.

PCLK is the real metronome that defines the duration of the time slots during which the microprocessor performs a task or a pre-defined portion of a task. Refer to Figure 2-2.

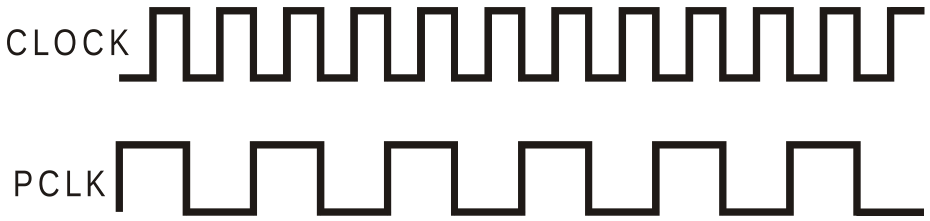


Figure 2-2. Relationship of CLOCK and PCLK

As illustrated in figure 2-2, PCLK is half the frequency of CLOCK. When a PC manufacturer refers to the operating speed of their computer as 8MHz (megahertz, or million cycles-per-second), 10MHz, etc., they are referring to the microprocessor's PCLK frequency, not that of CLOCK.

Microprocessor's Bus Cycle State Machine

When the microprocessor performs a read or a write operation, it initiates a sequence of events called a bus cycle.

During a bus cycle, the microprocessor places the address on the address bus, sets the control bus lines to indicate the type of transaction (such as a memory read or I/O write bus cycle), and transfers the data between the target location and itself. This happens in a very orderly fashion, with each step occurring at the proper point during the appropriate time slot.

x86 microprocessor chips include a subsystem called a bus unit, tasked with the job of running bus cycles when required. The bus unit is a state machine that is stepped through its various states by the PCLK signal. The duration of each state is one cycle of PCLK.

Refer to figure 2-3. As an example, if the CLOCK input frequency is 40MHz (40 million cycles per second), the PCLK frequency is half that frequency, or 20MHz. In order to determine the duration of one cycle of PCLK, just divide 20 million cycles-per-second into one second. In this example, a PCLK cycle is 50ns in duration (50 nanoseconds; 0.000000050 seconds or 50 billionths of a second). Since the bus unit state machine's states are each one PCLK in duration, this means that each bus unit state is 50ns in duration.

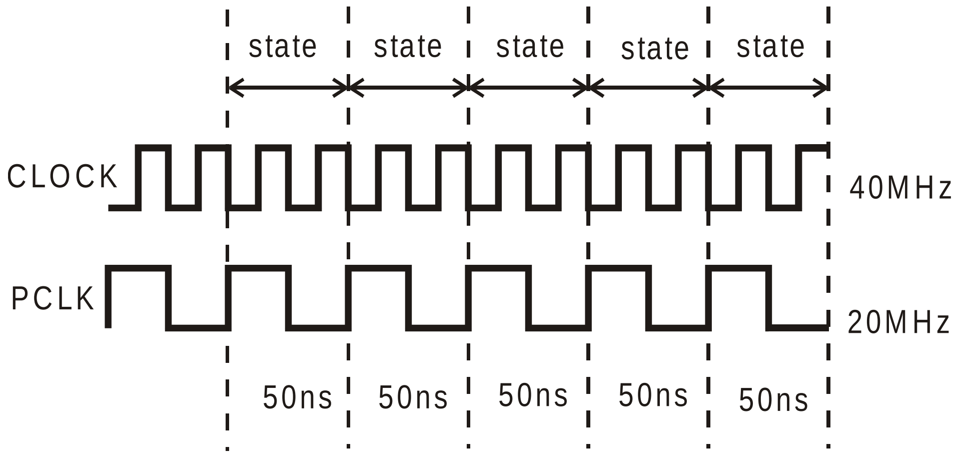


Figure 2-3. Each Cycle of PCLK Defines the Duration of a State

If not currently engaged in a bus cycle (a read or a write operation), the bus unit state machine is said to be in the idle state. It remains in the idle state until the microprocessor must perform a bus cycle.

Address Time

When the microprocessor must perform a read or a write, its bus unit initiates a bus cycle. The bus unit leaves the idle state and enters a state that will be referred to in this text as address time. Although Intel uses a different name for this state, address time is the name that will be used in this text for clarity's sake. During address time, one PCLK cycle in duration, the microprocessor places the address on the address bus and the bus cycle definition (type of bus cycle) on the control bus lines.

Data Time

After performing all actions required during address time, the bus unit state machine immediately enters the state we will refer to as data time. As with address time, Intel uses a different name for this state, but, for clarity's sake, data time is the name that will be used in this text. During data time, one PCLK cycle in duration, the microprocessor expects the data to be transferred between itself and the currently addressed device. At the end, or trailing edge, of data time (refer to reference point 1 in figure 2-4), the microprocessor samples (tests the state of) its READY# input to see if the currently addressed device is ready to complete the bus cycle. If the READY# input is asserted (low, as indicated by the pound sign), the bus unit state machine terminates the bus cycle.

If a read were in progress, the bus unit interprets an asserted level on READY# to mean that the currently addressed device has placed the requested data onto the data bus for the microprocessor. The microprocessor reads the data from the data bus and terminates the bus cycle.

If a write were in progress, the bus unit interprets an asserted level on READY# to mean that the currently addressed device has accepted the data being written to it. The microprocessor terminates the bus cycle. The chapter entitled “The 80286 System Kernel: the Engine” provides a detailed operational description of the Ready logic. Figure 2-4 illustrates a bus cycle consisting of address time and data time.

The 0-wait-state bus cycle is the fastest type of bus cycle the 80286, 80386, 80486 and Pentium microprocessors are capable of performing. In other words, the fastest x86 bus cycle takes two “ticks” (cycles) of PCLK. If this were an 80386 running at 20MHz (PCLK speed), a 0-wait-state bus cycle would take 100ns (50ns each for address time and data time).

Chapter 3

The Previous Chapter

The previous chapter, “Introduction To the Bus Cycle,” provided a basic description of the microprocessor's communications scheme. Using the three buses, the microprocessor performs bus cycles to read information from or write information to memory or I/O devices.

This Chapter

This chapter explores the addressing scheme used by the x86 processors. It defines the addressing range provided by the 8086, 8088, 286, 386, 486 and Pentium microprocessors. A basic definition of the term I/O device is also provided.

The Next Chapter

Having covered the addressing scheme used by the x86 processors in this chapter, the next chapter, “The Address Decode Logic,” introduces the concept of the address decoder. Several examples of address decoders are examined in detail.

Evolution of Memory and I/O Address Space

The Intel 8080 microprocessor is the ancestor of the entire x86 microprocessor family. Many of the family's characteristics stem from this common ancestor.

Intel 8080 Microprocessor Address Space

The 8080 microprocessor's address bus consists of 16 address lines, A[15:0]. With 16 address lines, the microprocessor can place any address pattern on the address bus from all zeros (0000h) to all ones (FFFFh). In other words, the 8080 can place any one of 65536 addresses on the address bus and can therefore address any one of 65536 locations located in external devices. In the computer industry, 65536 is referred to as 64K (K stands for Kilo, Greek for 1,000, but in the computer industry, kilo is 2^{10} or 1,024). Refer to figure 3-1. This means that the designer of an 8080-based system can include memory devices with a total of no more than 64K locations in the system.

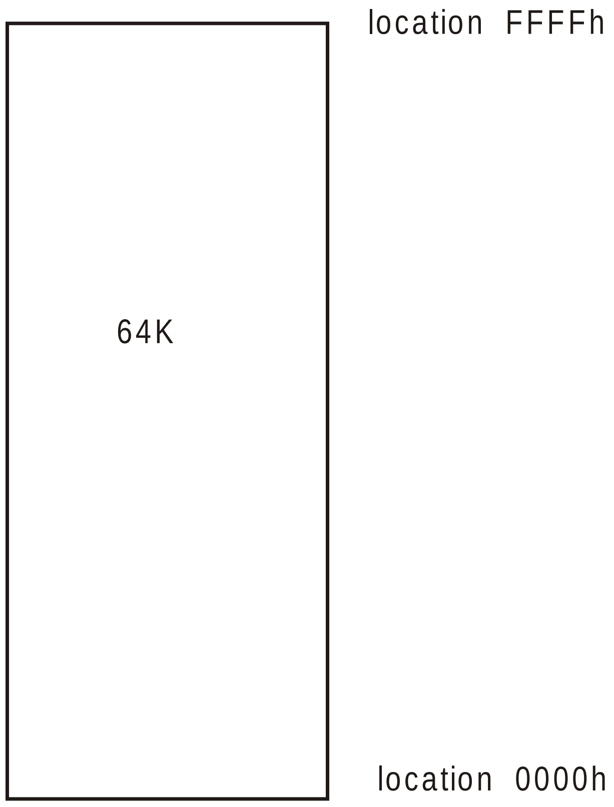


Figure 3-1. Single 64K Address Space

In addition to memory devices used for program and data storage, the designer must also use a number of locations for input/output ports (I/O ports) used to communicate with I/O devices. In simple terms, any device that the microprocessor can read data from or write data to that isn't a memory device (RAM or ROM) is an I/O device. Implementing these ports certainly is necessary, but depletes the number of addressable locations available to be assigned to memory devices.

Rather than use up already scarce memory locations to implement I/O devices, Intel provided two new instructions to the 8080 microprocessor: IN and OUT, and added a way for the 8080 to indicate to the system that the 8080 wishes to communicate with an I/O device instead of a memory device. The system typically included an Intel 8228 controller chip which detected the 8080's intention to communicate with an I/O device.

Some of the functionality of the 8228 was incorporated in the 8085 microprocessor, a successor of the 8080. The 8085 has 16 address lines, as does the 8080, so the 8085 can address 64K locations.

The 8085 has a new output pin called IO/M#. In Intel signal names, the "/" character should always be read as the English word "or." IO/M# therefore stands for "IO or Memory." Since there is no pound sign after the "IO", a high on this signal line means an I/O address is being output by the microprocessor. The pound sign after the "M" portion of the signal name indicates that a memory address is present on the address bus when IO/M# is low.

The addition of the IO/M# pin actually created two separate address spaces (or maps), one for Memory and the other for I/O. In essence, the IO/M# pin "points" to the appropriate address space when the microprocessor places an address on the address bus.

Refer to figure 3-2. As previously mentioned, the 8085 provides 64K of memory address space. However, due to the limited number of I/O devices typically found in systems, Intel only implemented 256 I/O locations.

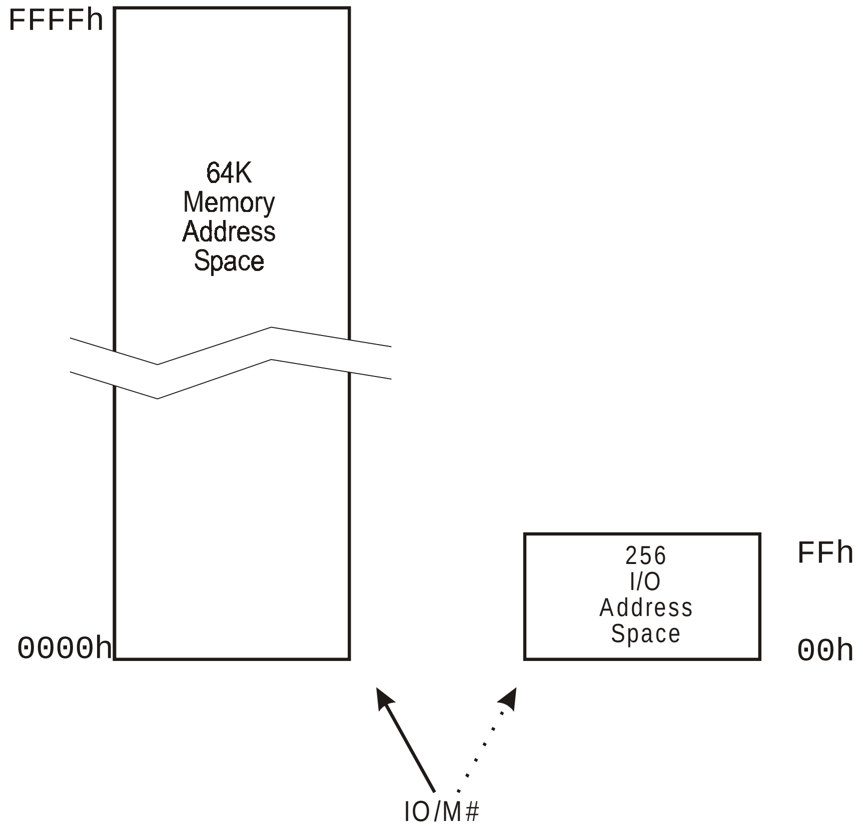


Figure 3-2. The 8085's Memory and I/O Address Spaces

As an example, if the microprocessor places address 0010h on the address bus and places a high on IO/M#, it is addressing I/O location 0010h, not memory location 0010h. On the other hand, if the microprocessor places address 0010h on the address bus and places a low on IO/M#, it is addressing memory location 0010h, not I/O location 0010h.

In addition to the IO/M# pin, Intel also added two I/O instructions, IN and OUT, to the 8085's instruction set. When executed, the IN instruction causes the microprocessor to perform a read from the I/O address specified by the programmer. This address is placed on the address bus and the IO/M# pin is set high, thereby indicating the presence of an I/O address on the address bus to external logic. In the same way, the OUT instruction also causes the microprocessor to place an I/O address on the address bus with IO/M# set high to indicate the I/O address it wishes to write data to.

A separate instruction, MOV, is used by the programmer to specify a memory address to move data to or from. When executed, the MOV instruction causes the specified memory address to be placed on the address bus and the IO/M# pin to be set low, indicating the presence of a memory address.

These changes allowed the designer to implement I/O ports without encroaching on memory space.

8086 and 8088 Microprocessor Address Space

When Intel designed the 8086 and 8088 microprocessors, they increased the number of address lines from 16 to 20. This means that the 8086 and 8088 microprocessors can place any address from 00000h to FFFFFh on the address bus, giving them an address range of 1,048,576 or 1M locations. "M" stands for the Greek prefix Mega, meaning large or great. In physics, M is 1,000,000, but in the computer industry, M is 2^{20} or 1,048,576. Some computer companies use M to mean 1,000,000 when referring to disk storage while still using M to mean 2^{20} when referring to memory size, so beware!

As with the 8085 microprocessor, the 8086 and 8088 microprocessors have a pin to indicate memory or I/O addresses. Note that the pin name and function is slightly different (or completely different, depending on your point of view). It's named M/IO#, meaning that the 8086 and 8088 set the M/IO# pin low when executing an IN or OUT instruction. The 8086 and 8088 set the M/IO# pin high when executing a MOV instruction to access memory. This is just the opposite of the way the 8085 did it.

For the 8086 and 8088 microprocessors, Intel also increased the size of the I/O address space to 64K. The programmer may specify any memory address within the 1M memory address range, but is restricted to I/O addresses in the first 64K locations of address space (0000h to FFFFh). There were two reasons for this decision:

1. The 8086 and 8088 microprocessors remain backward-compatible with the I/O instructions in the 8080 instruction set, although these instructions will not take full advantage of the processors address capability.
2. Virtually no designer requires more than 64K I/O ports (locations) to implement a full complement of I/O devices, no matter how complex they might be.

Chapter 4

The Previous Chapter

The previous chapter, "Addressing I/O and Memory," explored the addressing scheme used by the x86 processors. It defined the addressing range provided by the 8086, 8088, 80286, 80386, 80486 and Pentium microprocessors, and provided a basic definition of the term "I/O device."

This Chapter

This chapter describes the function performed by address decoders, discusses the address decoder design process, defines data bus contention, and provides a detailed analysis of two example address decoders found in ISA systems.

The Next Chapter

The next chapter, "The 80286 Microprocessor," provides a detailed description of the internal hardware architecture of the 80286 microprocessor, describes its register set, and supplies definitions of real and protected modes.

The Address Decoder Concept

Refer to figure 4-1. When the microprocessor must read data from or write data to an external device, it places the address on the address bus. Every device that can be read from or written to has an associated piece of logic called an address decoder. The address decoder selects its associated device when an address within its defined range is detected.

In most cases, the selected device contains multiple internal locations. When selected, the addressed device examines the address to identify the exact internal location with which the microprocessor wishes to perform a data transfer.

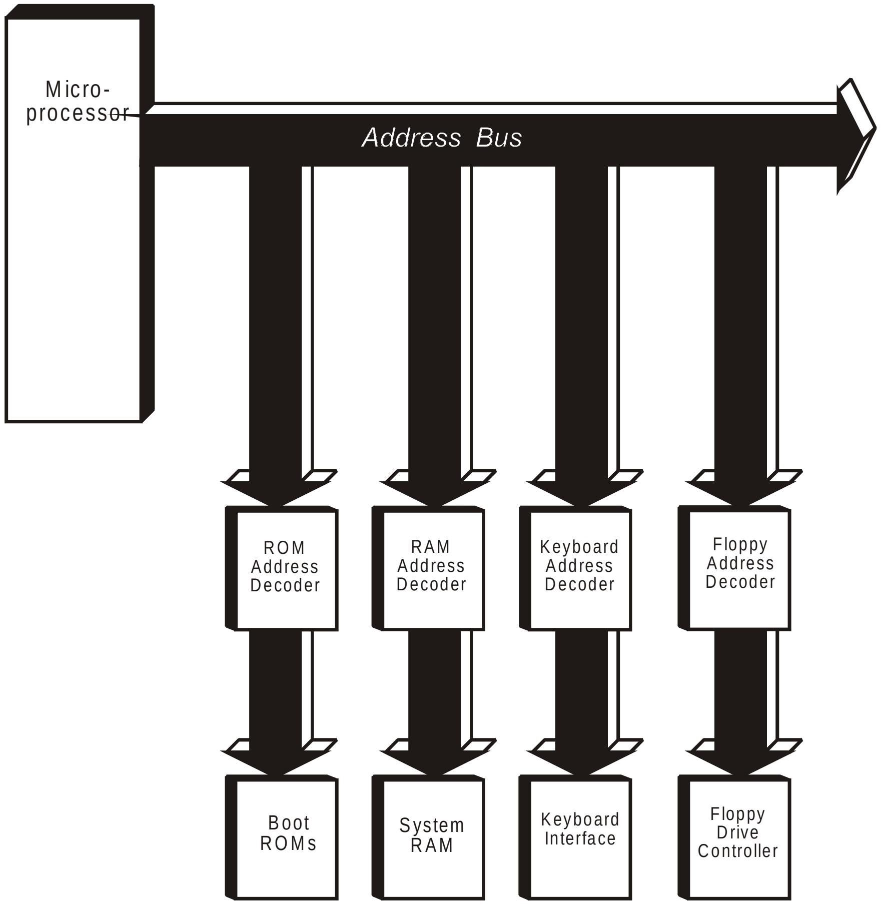


Figure 4-1. Relationship of Address Bus, Address Decoders and Devices

A memory or I/O chip does not respond to any pre-assigned address range. Rather, the device's address decode logic informs it that the address currently on the address bus is assigned to one of its internal locations by asserting the device's chip-select input pin.

The overall range of memory and I/O locations addressable by a particular microprocessor are commonly referred to as address maps. The design of its associated address decoder, not that of the memory or I/O device itself, defines where a device lives within the overall memory or I/O address map of a microprocessor.

Data Bus Contention (Address Conflicts)

Under no circumstances should the address decoders for two devices installed in a system be designed in such a fashion that they both detect the same address ranges. This would cause data bus contention.

Every expansion card installed in an ISA system has its own address decode logic that defines the address range the card responds to. Address conflicts are a common problem in the PC world. Consider the problem that arises if the address decoders on two expansion cards are both designed to detect the same address range.

As an example, assume that two cards respond to the I/O address range 03F0h to 03F7h. If the microprocessor executes an instruction that causes a read from I/O location 03F1h, both devices are chip-selected by their respective address decoders. Each then places the contents of its respective location 03F1h onto the common data bus simultaneously. One card may be driving a particular data bus signal line high while the other card is driving the same data bus signal line low. In this situation, we have two separate current sources trying to drive five volts and zero volts onto the same piece of wire.

At best, this situation will cause garbled data and, at worst, hardware damage. The proper term for this condition is data bus contention, a problem quite common in the PC world. A user populates a PC with many cards purchased from various manufacturers. The prospect of an address conflict is very real.

In order to permit resolution of address conflicts, a card designer should allow the user to easily alter the address range the board responds to. In ISA systems, this is usually accomplished with configuration switches (or jumpers) on the board. By changing the switch setting, the user causes the card's address decoder to detect a different address range, thereby resolving the conflict.

How Address Decoders Work

The address decoder function is similar to that performed by the post office. Because it more or less identifies a neighborhood, the zip code is analogous to the high-order part of the address (upper address bits). The low-order portion of the address identifies the exact location within the neighborhood.

In much the same way, the address decoder associated with a device examines the high-order part of the address being output by the microprocessor. It determines if the address is within the range assigned to its device. If it is, the address decoder chip-selects the device. The chip-selected device then examines the lower part of the address to determine the exact location within it the microprocessor is addressing.

It should be noted that an address decoder doesn't have to look at the entire address to determine that the current address is within the address range assigned to its respective device. If you were traveling in a car and were told to keep an eye out for the two thousand block, you would only need to look at the thousands digit of the address to determine if it was within the desired range. The lower digits would be insignificant to you.

The following two sections describe example address decoders and how they function. The first example is that of the ROM address decoder in an IBM PC/XT. The second example is that of the address decoder that detects addresses assigned to the I/O devices found on a typical ISA system board.

Example 1– PC and PC/XT ROM Address Decoder

Background

The original IBM PC was designed around the 8088 microprocessor. With an address bus consisting of 20 lines, designated A[19:0], the 8088 could address any one of 1,048,576 memory locations. The designers chose to populate this memory map as illustrated in Figure 4-2.

The memory address range from F0000h to FFFFFh was assigned to the system board ROM. This ROM is frequently referred to as the boot or POST/BIOS ROM. The 64K addresses from F0000h to FFFFFh are set aside for the boot ROM. It should therefore be selected whenever the uppermost digit on the address bus is Fh and the M/IO# pin is high (indicating the presence of a memory address on the address bus). Table 4-1 illustrates the state of the address lines when any memory address in this range is accessed.

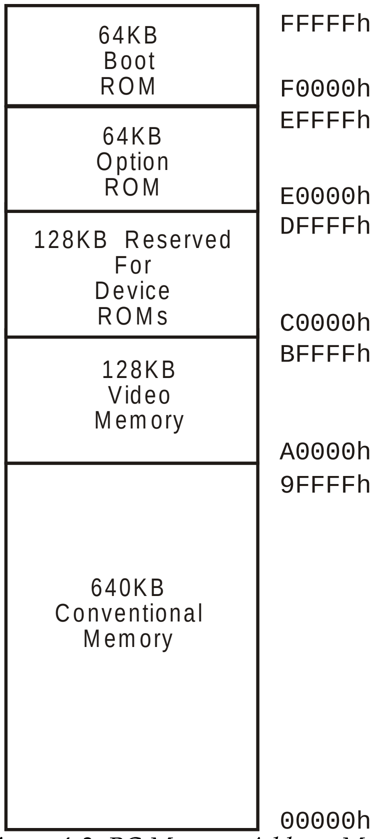


Figure 4-2. PC Memory Address Map

Table 4-1. State of the Address Lines When Any Memory Address in Range F0000h to FFFFFh Is Addressed

<u>19</u>	<u>18</u>	<u>17</u>	<u>16</u>	<u>15</u>	<u>14</u>	<u>13</u>	<u>12</u>	<u>11</u>	<u>10</u>	<u>9</u>	<u>8</u>	<u>7</u>	<u>6</u>	<u>5</u>	<u>4</u>	<u>3</u>	<u>2</u>	<u>1</u>	<u>0</u>
1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
F				X				X				X				X			

In addition, memory address range E0000h to EFFFFh was assigned to the optional ROM that could be installed in the empty ROM socket provided on the system board. This capability was provided to allow the installation of a ROM containing the BIOS routines and interrupt service routines for optional devices that could be added to the machine. This add-in ROM is commonly referred to as the option ROM. The 64K addresses from E0000h to EFFFFh are set aside for the option ROM. It should therefore be selected whenever the uppermost digit on the address bus is Eh and the M/IO# pin is high (indicating the presence of

Chapter 5

The Previous Chapter

The basics of microprocessor communication have been introduced. This includes concept of the bus cycle and the types of devices that the microprocessor communicates with (memory and I/O), along with how these devices are addressed.

This Chapter

This chapter provides a description of the 80286 microprocessor. Emphasis is placed on the hardware interface between the 80286 microprocessor and the remainder of the system. Memory address formation in real mode is covered in detail and an introduction to protected mode is provided.

The Next Chapter

The next chapter provides a description of the six possible sources for various reset signals found in ISA systems and the effect each of them has on the system. It also explains the actions taken when Ctrl/Alt/Delete are depressed.

The 80286 Functional Units

Any microprocessor is really a system consisting of a number of functional units. Each unit has a specific job to do and all of the units working together comprise the microprocessor. Figure 5-1 illustrates the units which make up the 80286 microprocessor.

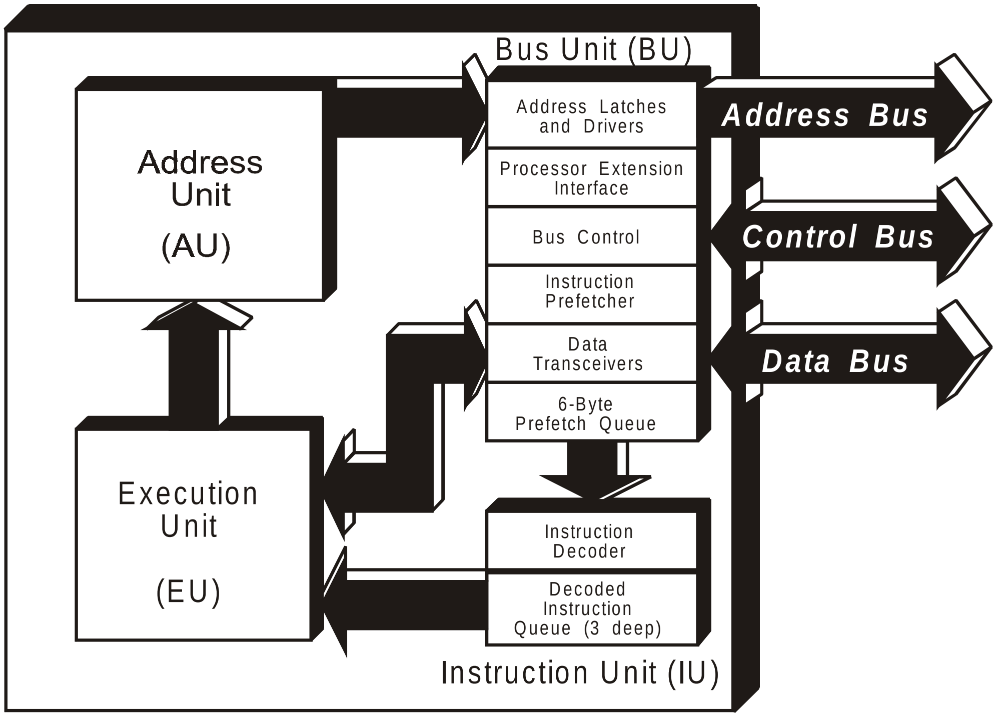


Figure 5-1. 80286 Microprocessor Block Diagram

Refer to 5-1. When the microprocessor starts operation, it will automatically begin fetching (reading) instructions from memory. It is the bus unit's task to perform the bus cycles required to fetch all instructions from memory. Instructions are read from memory over the data bus and placed into the instruction Prefetch Queue. From there they go to the instruction unit where they are decoded and sent to the execution unit one at a time for processing.

Execution of some instructions requires that data be read from or written to memory or I/O devices. If a memory device is specified, the execution unit directs the address unit to form the memory address of the location specified by the instruction. I/O addresses specified by an instruction do not require special address formation as do memory address. Instead they are sent directly from the execution unit to the bus unit to run the bus cycle specified by the instruction.

In summary, two types of information may be read into the microprocessor; instructions (code) and data. Instructions are read from memory over the data bus, go through the data transceivers, and are placed into the prefetch queue. Data to be processed is read from either memory or I/O devices over the data bus and is transferred via the data transceivers to the execution unit.

An instruction may also specify that data be written to a memory or I/O location, in which case data is sent from the execution unit to the data transceivers for transfer over the data bus.

In short, the 80286 microprocessor consists of four functional units:

1. **Instruction unit**. Decodes the instruction prior to passing it to the execution unit for execution.
2. **Execution unit**. Handles the actual execution of the instruction.
3. **Address unit**. When the microprocessor must address a memory location, the address unit forms the memory address that is driven out onto the address bus by the bus unit during the bus cycle.
4. **Bus unit**. Handles communication with the world outside the microprocessor chip (by performing bus cycles).

The following sections provide a more detailed description of each of these units.

The Instruction Unit

One at a time, the 80286 instruction unit pops (reads) instructions from the prefetch queue, decodes them into their component parts and places them into the decoded instruction queue to be forwarded to the execution unit.

The Execution Unit

Instructions are executed by the 80286 execution unit. In general, instructions cause either internal processing of data already stored within the execution unit registers, or reading data from or writing data to devices external to the microprocessor.

When the currently executing instruction requires writing or reading a memory or I/O location, the execution unit issues a bus cycle request to the bus unit. The execution unit then stalls until the bus unit completes the requested data transfer.

Two types of information may be read into the microprocessor: instructions (code) or data. Instructions are always read from memory and are placed into the instruction prefetch queue. From there, they are routed to the instruction unit where they are decoded and sent to the execution unit for processing. Data to be processed goes directly to registers inside the execution unit. These registers have several functions as described in the following paragraphs.

A register can be thought of as a storage location within the microprocessor. Some registers can only be read from, and some can only be written to, while others are both readable and writable. Registers within the execution unit are used by the programmer or the microprocessor itself for the following purposes:

- to temporarily hold values to be used in calculations
- to receive data being read from an external memory or I/O location
- to provide the data to be written to an external memory or I/O location
- to keep track of the microprocessor's current status
- to control certain aspects of the microprocessor's operation

Although they are storage locations, the programmer refers to the microprocessor's registers by names (rather then by the hex addresses used to address external memory and I/O locations). Most of the 80286 registers are 100% backward-compatible with the 8086 and 8088 microprocessor's registers. The registers inside the execution unit can be logically divided into two categories:

- General Registers
- Status and Control Registers

The following sections describe each of the registers in the General and the Status and Control register sets.

General Registers

This section offers a description of each of the general registers and examples of their usage. Figure 5-2 details the registers in the general register set.

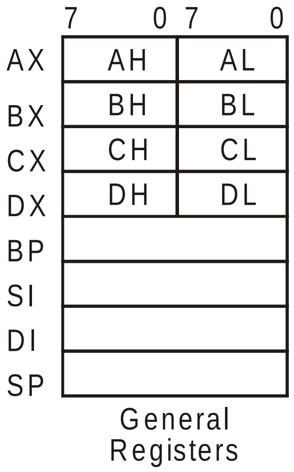


Figure 5-2. The 80286 General Registers

The AX, BX, CX and DX Registers—Each of these four registers can hold 16 bits (two bytes) of information. As an example, the following instruction will move the 16-bit value 1234h into the AX register:

```
MOV AX,1234    ;move the value 1234h into AX
```

In addition, the programmer can individually refer to each of the 8-bit registers that make up the upper and lower half of each of these 16-bit registers. As an example, the AX register actually consists of two 8-bit registers: the AL (Lower half of AX register) and AH (upper half of the AX register) registers. The same is true of the BX, CX and DX registers.

If the programmer wants to move only one byte of information, he or she would specify one of the 8-bit registers rather than a 16-bit register. In the following example, the programmer wants to write the value 02h into I/O location 60h:

```
MOV AL,02 ;move value 02h into AL
OUT 60,AL ;write AL contents to I/O port 60h
```

In the example above, the programmer first moves the value 02h into the 8-bit AL register and then performs an I/O write instruction to write the contents of the AL register to I/O location 60h. This will cause an I/O write bus cycle with address 000060h on the address bus and 02h on the lower part of the data bus

Chapter 6

The Previous Chapter

The first five chapters introduced the concepts of microprocessor communications and provide a detailed understanding of the 80286 microprocessor's interface signals.

This Chapter

ISA products usually have six separate source for reset, two of which are generated only by hardware and four that can be initiated by software commands. This chapter details the reset sources within typical ISA systems.

The Next Chapter

The next chapter explains the sequence of events that occur immediately after an ISA system is powered on.

The Power Supply Reset

The power supply provides the operating voltages necessary for system operation. When the power switch is first placed in the on position, it takes some time for the power supply's output voltages to reach their proper operating levels. If the system components were allowed to begin operating before the voltages stabilized, it would result in erratic operation.

Every PC power supply produces an output signal commonly called POWERGOOD. On the system board, the POWERGOOD signal is used to produce the RESET signal. During the period required for stabilization of the output voltages, the POWERGOOD signal is kept deasserted by the power supply. While POWERGOOD is deasserted, the RESET signal is kept asserted.

While RESET is asserted, it has two effects on the microprocessor and other system components:

1. Keeps any activity from occurring until power has stabilized.
2. Presets the microprocessor and other system devices to a known state prior to letting them begin to do their job. This ensures that the machine will always start up the same way.

When POWERGOOD becomes asserted, the RESET signal is deasserted and the microprocessor can begin to fetch and execute instructions. The first instruction is always fetched from the power-on restart address which is always located 16 locations from the very top of the memory address space. For the 8088 microprocessor, this would be FFFF0h, FFFFF0h for the 80286 and 80386SX, and FFFFFFF0h for the 80386DX, 80486 and Pentium microprocessors. This location contains the first instruction of the POST.

Reset Button

Some systems provide a momentary-contact switch which the user may press to force the RESET signal to be asserted to the microprocessor and to the rest of the system. The effect on the system is the same as deasserting POWERGOOD from the power supply, but the power supply is not actually turned off.

Shutdown Detect

When the microprocessor detects an exception while attempting to execute the handler for a prior exception, the two exceptions are generally handled serially. Certain combinations of exceptions cannot be handled serially, however. These exceptions are caused by protection violations in protected mode. For example, in protected mode, if an application attempts to access data that is outside its data segment, the microprocessor will start running the general protection fault exception handler. If, say, a stack overflow were to occur while the microprocessor ran the general protection fault exception handler, the microprocessor would not be able to handle the two faults serially. The microprocessor would invoke the double-fault exception instead.

If the microprocessor detects another exception while attempting to service the double-fault, it goes into a shutdown condition. This is sometimes referred to as a triple fault. The microprocessor indicates the shutdown condition to external logic by running the halt or shutdown bus cycle. Since the microprocessor has abnormally ceased to fetch and execute instructions, it can’t run a program to inform the user that a severe error has occurred. Therefore, in an ISA machine, the shutdown detect logic outside the microprocessor detects the shutdown and toggles the microprocessor's RESET line, causing the system to reboot.

See the chapters entitled "Detailed View of the 80286 Bus Cycle" and "Detailed View of the 80386 Bus Cycle" for additional information on the halt or shutdown bus cycle.

Hot Reset

MS-DOS was written specifically for the Intel 8088 microprocessor using 8088-specific instructions. Since the 8088 only has 20 address lines, it is incapable of generating a memory address greater than FFFFFh, or 1MB. Furthermore, protected mode wasn't introduced until the advent of the 80286 microprocessor. Consequently, MS-DOS can only address the lower 1MB and doesn't understand protected mode.

When an 80286 is powered up, it operates in real mode. In other words, it operates as if it were an 8088. This means that, although the 80286 has twenty-four address lines and can physically address up to 16MB of memory address space, it is limited to the lower 1MB when in real mode.

Most current applications programs are written to run under MS-DOS. Many of these programs require access to more memory space than allowed under MS-DOS. As an example, many Lotus 1-2-3 spreadsheets require very large amounts of memory space, well in excess of that allowed under MS-DOS and the 8088 microprocessor.

If a system is based on the 80286 microprocessor, the problem can be solved by switching the microprocessor into protected mode. This allows the microprocessor to access up to 16MB of memory space, but MS-DOS presents a problem. It doesn't understand protected mode and is therefore incapable of generating addresses above 1MB.

Special software that understands protected mode operation prepares the segment descriptor tables in memory prior to switching into protected mode. It also must save the address of the next instruction to be executed when the microprocessor returns to real mode so MS-DOS can continue to run at the point where it went to protected mode. The 80286 is then switched into protected mode and the extended memory above 1MB accessed (for example, to store spreadsheet data).
Once the extended memory access has been completed, the microprocessor must be switched back into real mode so the applications program can continue

to run under MS-DOS. Here's where the problem comes in. In order to switch the 80286 microprocessor from protected to real mode, the microprocessor must be reset.

To do this, the following actions must be taken:

1. The programmer stores an address pointer that points to the address of the real mode instructions that are responsible for restoring the system to its previous operating condition. This pointer is stored in locations 0040:0067 to 0040:006A. The contents of these locations will be used later when the processor returns to real mode to find the address in memory where the real mode instructions reside.
2. A special value (05h or 0Ah) is stored in the configuration CMOS RAM location (0Fh) to indicate the reason for the reset. Refer to table 6-1 for definitions of the reset byte in CMOS RAM.
3. The system has now been prepared to return to real mode. The Hot Reset command must now be issued to the keyboard/mouse interface.* This is accomplished by writing a FEh to the keyboard/mouse interface's command port at I/O address 0064h.
4. In response, the keyboard/mouse interface pulses its Hot Reset output one time, causing the hardware to generate a reset to the 80286.
5. When reset becomes deasserted, the microprocessor begins to fetch and execute instructions at the power-on restart address exactly as if a power-up had just occurred.
6. At the beginning of the POST, the programmer reads the value stored in the configuration RAM location to ascertain the reset's cause. In this case, the value (05h or 0Ah) indicates that it was caused by Hot Reset to get back to real mode and continue program execution.
7. POST then retrieves the previously stored real mode address pointer found at location 0040:0067 and jumps to the indicated address and picks up where it left off in real mode.

* Some manufacturers employ a Hot Reset “intercept” where external circuitry recognizes an FEh written to the keyboard controller and immediately generates a <u>Fast</u> Hot Reset. This provides a faster reset signal compared to the slower generation via a command to the keyboard controller. Later versions of Intel's microcontrollers (typically used as keyboard controllers) provide an internal intercept that also results in a fast Hot Reset.

Another method used to produce a fast Hot Reset is to cause the microprocessor to go into the shutdown condition by intentionally causing faults using software. This causes the shutdown detect logic to reset the microprocessor.

Alternate (Fast) Hot Reset

The Alternate or Fast Hot Reset command is found in some ISA systems that implement system control port A at I/O address 0092h, sometimes called the PS/2 compatibility port. Alternate Hot Reset performs the same function as the Hot Reset command. However, the microprocessor is reset more quickly using this method than when using Hot Reset. The Hot Reset command must be interpreted by the ROM-based program inside of the keyboard/mouse interface, while the Alternate Hot Reset command is executed by the hardware much more quickly.

Alternate Hot Reset is generated by setting bit 0 in system control port A to a one. This generates a pulse on the Alternate Hot Reset signal line which, in turn, causes a pulse on the Hot Reset signal. This resets the microprocessor. For a more extensive explanation of Hot Reset, see the previous section.

Ctrl-Alt-Del Soft Reset

When the control and alternate keys are depressed and held, followed by the delete key being depressed, three “make” codes for these keys are stored in the keyboard buffer in memory. When the sequence of “make” codes are detected by the keyboard interrupt service routine, the soft reset routine is invoked. The soft reset routine causes the system to:

1. Place the value 1234h into the reset flag location (0040:0072) in main memory. This value tells the system to skip a portion of the POST when rebooting.
2. Store a special value (00h)in configuration RAM to indicate that a Control/Alternate/Delete is causing the reset (also used for power-on reset).
3. Depending on the computer model and manufacturer one of two methods is typically used to reset the processor.
   - Write a FEh to port 64h, causing a CPU reset. When the reset is removed, the microprocessor begins to fetch and execute code from location FFFF0h, the beginning of POST.
   - Jump to location FFFF0h and begin program execution.
4. Near the beginning of the POST, the programmer reads the value stored in the configuration RAM location to ascertain the reset's cause. In this case, the value indicates that it was caused by Control/Alternate/Delete or power-on reset.

Chapter 7

The Previous Chapter

The previous chapter, “The Reset Logic,” provided a detailed description of the possible sources for RESET.

This Chapter

This chapter provides a description of the ISA machine power-up sequence from the moment power is applied until the microprocessor begins to fetch and execute instructions.

The Next Chapter

The next chapter, “The 80286 System Kernel: the Engine,” describes the support logic necessary to allow the 80286 microprocessor to communicate with 8- and 16-bit devices.

The Power Supply – Primary Reset Source

The power supply provides the operating voltages necessary for system operation. When the power switch is first placed in the on position, a period of time is required for the power supply's output voltages to reach their proper operating levels. If the system components are allowed to begin operation before the supply voltages have stabilized, erratic operation would result.

Figure 7-1 illustrates the power supply-related logic that produces the microprocessor's RESET signal. Every ISA PC power supply produces an output signal commonly called POWERGOOD. On the system board, the RESET signal is derived from the POWERGOOD signal. In order to guarantee proper operation of the microprocessor, the RESET input must change state in synchronism with the CLK2 signal (the microprocessor's double-frequency clock input). On each rising edge of CLK2, the state of the POWERGOOD input is latched by the flip-flop, inverted to its opposite state, and presented on the flip-flop's output, RESET. During the period required for stabilization of the output voltages, the

POWERGOOD signal is held deasserted by the power supply. While POWERGOOD is deasserted, the RESET signal is held asserted. The flip-flop in the figure provides a RESET output that is synchronized with CLK2 and is the invert of the POWERGOOD signal from the power supply.

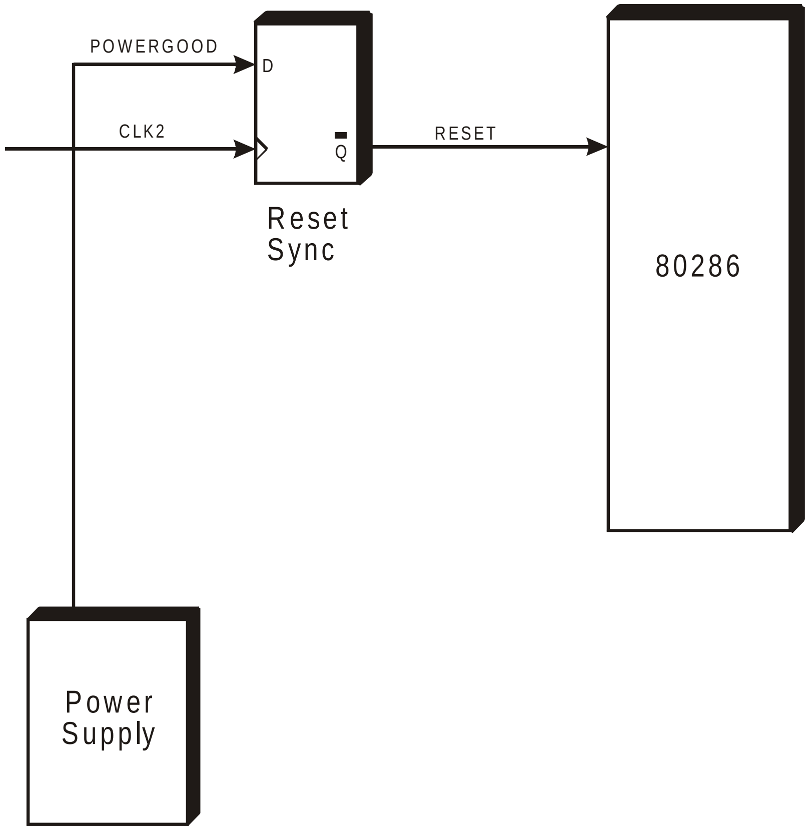


Figure 7-1. RESET Is Derived from POWERGOOD

While RESET is asserted, it has two effects on the microprocessor and other system components:

1. Keeps any activity from occurring until power has stabilized.
2. Presets the microprocessor and other system devices to a known state prior to letting them begin normal operation. This ensures that the machine will always start up the same way.

How RESET Affects the Microprocessor

Table 7-1 defines the values forced into the 8086 and 8088 microprocessor's registers when RESET is asserted.

Table 7-1. Values Preset Into 8086 and 8088 Microprocessor Registers by RESET

Register	Contents
FLAGS	0002h
MSW	FFF0h
IP	FFF0h
CS	F000h
DS	0000h
ES	0000h
SS	0000h

While RESET is asserted, the microprocessor cannot fetch and execute instructions and its outputs are set as illustrated in tables 7-2 and 7-3.

Table 7-2. State of 80386DX Outputs While RESET is Asserted

Pin Name	Pin State
LOCK#, D/C#, ADS#, A[31:2]	high
W/R#, M/IO#, HLDA, BE#[3:0]	low
D[31:0]	tri-state (floating)

Table 7-3. State of 80286 Outputs While RESET Is Asserted

Pin Name	Pin State
A[23:0], S0#, S1#, PEACK#, BHE#, LOCK#	high
M/IO#, COD/INTA#, HLDA	low
D[15:0]	tri-state (floating)

Processor Reaction When Output Voltages Stabilize

When the power supply output voltages have stabilized, the POWERGOOD signal is asserted and the logic on the system board responds by deasserting RESET. The deasserted level on RESET allows the microprocessor to begin functioning. It should be noted that the 386, 486 and Pentium microprocessors can perform a self-test when RESET becomes deasserted. Details of the processor self-test are discussed in the chapter entitled “The 80386DX and SX Microprocessors" and in the MindShare Architecture Series books *80486 System Architecture* and *Pentium Processor System Architecture* published by Addison-Wesley. The microprocessor initiates the fetching and executing of instructions from memory, using the code segment (CS) and instruction pointer (IP) register contents to point to the memory location containing the first instruction.

While RESET was asserted, the machine status word (MSW) register had FFF0h forced into it. Since the protect enable bit, bit 0, is therefore 0, the microprocessor always begins operation in real mode. In real mode, the microprocessor always appends an extra 0h on the end of the CS register contents, F000h, resulting in a code segment start address of F0000h. It then adds the offset portion of the address, FFF0h, contained in the IP register, to the segment start address:

```
CS       F0000h
IP    +   FFF0h
         FFFF0h = physical memory address
```

The resultant memory address, FFFF0h, is referred to as the power-on restart address. Since the address is formed exactly the same way every time the system is powered up, the 8086 or 8088 microprocessor always fetches its first instruction from the power-on restart address.

The First Bus Cycle

The microprocessor then initiates a memory read bus cycle to fetch the first instruction from the power-on restart address in memory. The power-on restart address is always located in the boot ROMs. This is because the first instruction must be located in ***non-volatile memory***. If the first instruction were fetched from RAM memory, the information returned would be junk. This is because RAM memory is ***volatile***.

The first instruction fetched from the power-on restart address is always the first instruction of the power-on self-test, or POST. The POST program is contained in the boot ROMs and is always the first program to be run.

Although they always begin operation in real mode and should therefore function as a fast 8086, the 286, 386, 486 and Pentium microprocessors have more than twenty address lines. When the 286 or higher microprocessors place the power-on restart address on A[19:0] of the address bus, the address lines above A19 are automatically set high. This results in address FFFFF0h being placed on the 286 and 386SX address bus, rather than 0FFFF0h. The 386DX, 486 and Pentium processors place FFFFFFF0h on the address bus.

The processor keeps these upper address lines high until the program performs a jump to another code segment; in other words, until a new value is loaded into the CS register. A jump instruction that reloads both the CS and the IP registers is referred to as an "inter-segment jump" or a "far jump." The upper address lines are then set low and remain low unless the processor is switched into Protected Mode by the program. In practice, the first instruction executed from system ROM is always a jump instruction, thus a new CS value is immediately loaded when the first instruction is executed.

After the upper bits are set low, they will not be set high again while the processor remains in real mode. This means that the processor cannot access extended memory (memory above the lower megabyte) while in real mode. There is one exception to this rule. It is discussed in the chapter entitled "The 80286 Microprocessor" under the heading "Accessing Extended Memory in Real Mode."

Chapter 8

The Previous Chapter

The preceding chapters have explained how the 80286 microprocessor communicates with memory and I/O devices and how the 80286 interfaces with external devices. Sources of system reset and the power-up sequence have also been covered.

This Chapter

This chapter introduces and explains the external logic required by the 80286 microprocessor to communicate with 8- and 16-bit devices. Additional logic needed to synchronize memory and I/O devices of varying speeds is also covered. A thorough understanding of this chapter is crucial to understanding the 80386 kernel.

The Next Chapter

The next chapter provides a detailed look at the bus cycles that an 80286 can run. The chapter includes timing diagrams and explanations for read, write, and the halt or shutdown bus cycles.

The Bus Control Logic

In order to perform all of the actions required to complete a bus cycle, the 80286 microprocessor requires the aid of external logic. One of the major pieces of logic involved in this process is referred to as the bus control logic. Refer to figure 8-1.

Logic external to the microprocessor can detect the beginning of a bus cycle by looking at the 80286's bus cycle definition outputs. When either S0# or S1# is detected going low, this indicates the start of the bus cycle. This triggers the bus control logic's state machine that works with the microprocessor's Bus Unit to accomplish a data transfer during a bus cycle. The bus control logic uses CLK2, the double frequency clock, to define time slots for its state machine. At the proper points during a bus cycle, the bus control logic performs the appropriate actions to accomplish the bus cycle.

The functions performed by the bus control logic are described in this chapter.

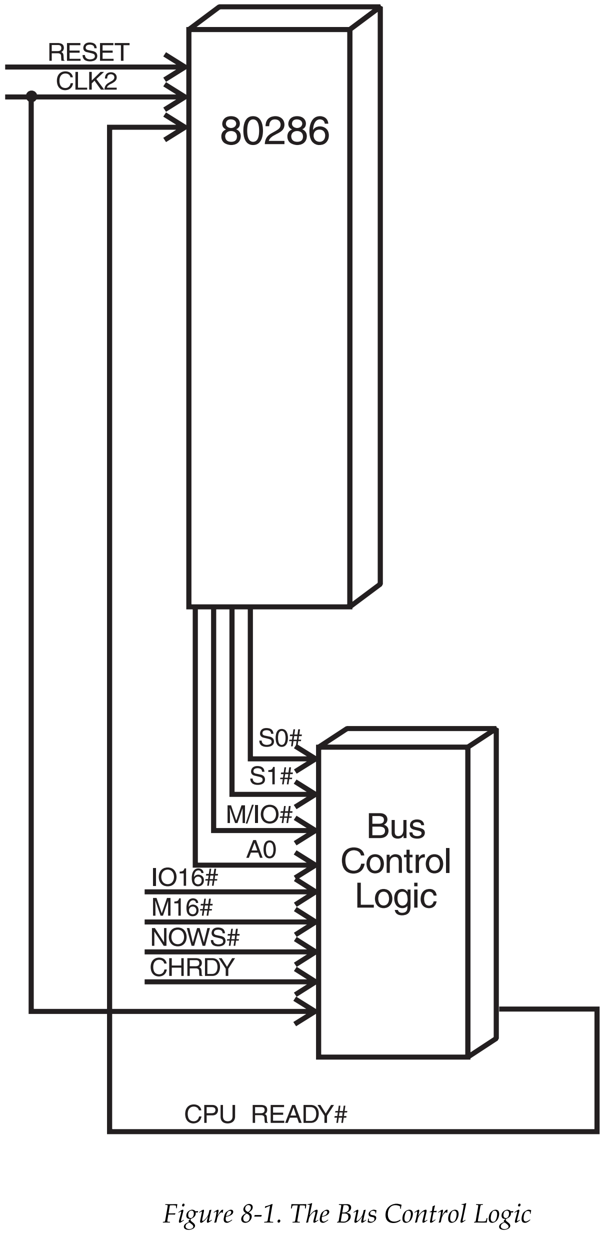


Figure 8-1. The Bus Control Logic

The Address Latch

Refer to figure 8-2. Intel dictates that every 80286-based system must incorporate an external device known as an address latch. When the microprocessor outputs the address onto its local address bus during a bus cycle, the bus control logic signal ALE (address latch enable) commands the address latch to hold (latch) the address and remember it. Once latched, the address latch outputs the address to the system on the system address (SA) bus, SA[19:1]. Address decoders throughout the system can then examine the latched address to see if the microprocessor is attempting to communicate with their respective device.

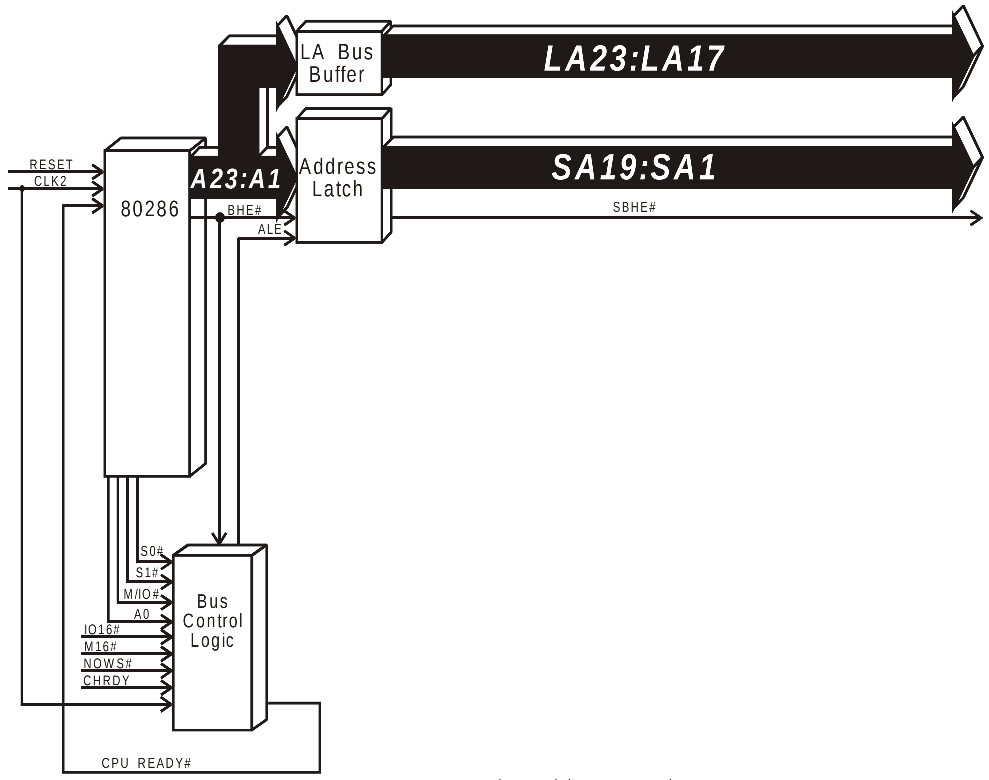


Figure 8-2. The Address Latch

The address latch and LA lines (latchable address lines) must be included to support the 80286's address pipelining capability. Notice that address lines A[23:20] are not latched. These address lines go directly to the LA bus through a buffer, along with address lines A[19:17]. From the buffer, the LA bus is connected directly to the 16-bit slots on the ISA bus. The purpose of the LA bus will be described later.

Notice that the address latch only latches A[19:1], but not A0 (address bit A0). The reason for this is clarified later. Rather than going to the address latch, A0 goes to the bus control logic where it is allowed to pass through onto SA0 at the same time as A[19:1] are latched and presented on SA[19:1].

Address Pipelining

When the 80286 microprocessor must perform back-to-back memory transfers, it uses address pipelining. This involves placing the address for the next cycle on the microprocessor's local address bus during the current bus cycle. This is possible because the current address will have been latched on the system address bus (SA Bus) and held for the duration of the current bus cycle. As a result, the address for the next bus cycle can be output by the microprocessor since the address for the current bus cycle has already been latched.

A portion of the next bus cycle's address, LA[23:17], goes directly to the ISA bus via an address buffer. LA[23:17] (the latchable address bus) are simply a buffered version of the uppermost bits of the microprocessor's address bus. 16-bit memory expansion boards are designed to decode the LA address lines. This allows 16-bit memory expansion boards to chip select their memory devices much earlier than would be possible if they waited for the latched address (SA lines).

At the end of address time of the next bus cycle, the bus control logic again pulses the address latch enable (ALE) line, commanding the address latch to latch address lines SA[19:0] and SBHE#. Since the high order bits (LA[23:17]) have already been used to generate the chip select, address decoding can be completed very quickly. Additional information regarding address pipelining can be found in the chapter entitled “Detailed View of the Bus Cycle.”

Although the 80286 microprocessor pipelines out its entire 24-bit address, the ISA bus only provides seven of those lines, LA[23:17]. It would have been more expensive to include all 24 of the pipelined address lines and system designers are always looking for ways to reduce cost while maintaining or improving performance. If the designers had provided all 24 lines, the cost would have gone up but the performance would not have improved. There are two reasons for this: 1) the vast majority of bus cycles are memory reads since the processor is constantly fetching instructions and 2) the 4164 memory chip was popular when the PC/AT was designed. The 4164 has 64Kbits of information in it, but it’s only one bit wide. Sixteen 4164’s are needed to make a bank of memory for the 80286 microprocessor. Sixteen times 64Kbits yields 128KB, which is the smallest block of addresses that can be produced using LA[23:17]. With LA[23:17], the memory subsystem has enough information to select the proper bank of memory chips. Additional information about memory banks can be found in the chapter entitled “RAM Memory: Theory of Operation.”

Chapter 9

The Previous Chapter

Previous chapters have covered how the microprocessor communicates with memory and I/O devices and have explained the support logic needed to allow the microprocessor to communicate with both 8- and 16-bit devices.

This Chapter

The exact timing of read and write bus cycles for both memory and I/O are covered in this chapter. Detailed explanations of the halt and shutdown bus cycles, along with the causes of each are also covered.

The Next Chapter

The next several chapters detail the 80386 microprocessor and the support circuitry required for the 80386 kernel.

Address and Data Time Revisited

The concept of the bus cycle was introduced in the chapter entitled "Introduction to the Bus Cycle." The microprocessor performs a bus cycle when it has to transfer information between itself and a memory or I/O location. The microprocessor's bus unit uses the address, data and control buses to address a device, indicate the type of transaction in progress and to transfer the data between the microprocessor and the currently addressed location.

The microprocessor uses the bus cycle definition lines to indicate the type of transaction. The types of bus cycles are indicated in table 9-1.

With the exception of the interrupt acknowledge bus cycle, this chapter provides a detailed description of each bus cycle type. The interrupt acknowledge bus cycle is described in the chapter entitled "The Interrupt Subsystem."

Table 9-1. 80286 Bus Cycle Definition

M/IO#	S1#	S0#	Bus Cycle Type
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
1	0	0	Halt or Shutdown
1	0	1	Memory Read
1	1	0	Memory Write

When it must perform a bus cycle, the microprocessor's bus unit leaves the idle state and enters address time. During this “tick” of PCLK, the microprocessor places the address and bus cycle definition on the buses. Address decoders throughout the system start decoding the address during this time slot.

Address time is always followed by data time. During this state, the microprocessor expects the data to be transferred between itself and the currently addressed device.

In reality, these aren't the state names Intel uses for the 80286 and 80386 bus unit states. Table 9-2 lists the state names used by Intel:

Table 9-2. 80286 and 80386 Bus Cycle State Names

Nickname	80286 Name	80386 Name
Address Time	Ts	T1
Data Time	Tc	T2

These states were renamed for clarity's sake because the names Ts and Tc provide little information regarding the actions that occur during these states. Intel calls address time Ts, meaning “Send Status” time, because one of the things that occurs during Ts is the output of the bus cycle definition on M/IO#, S0#, and S1#. Intel refers to S0# and S1# as the microprocessor's status outputs, hence the name “Send Status” time. Data time is called Tc, “Perform Command” time, because it's time to perform the command, or data transfer, that the microprocessor initiated.

The various actions that must take place during a bus cycle have been described in previous chapters. This chapter provides an in-depth look at the sequence and exact timing of these actions in relation to each other.

The Read Bus Cycle

With the exception of the M/IO# output, I/O and memory read bus cycles are identical. This section describes the exact timing of the actions performed during a read bus cycle. Figure 9-1 is a timing diagram illustrating a series of typical 80286 read bus cycles.

PCLK has been placed across the top of the timing diagram as a point of reference. Remember that each cycle of PCLK defines the duration of one bus unit state. Vertical dotted lines have been superimposed on the diagram to define each cycle of PCLK (and, therefore, each state). In addition, the name of each state is written across the top of the diagram.

A general idea of the information presented can be derived from the state names across the top. In this example, the sequence of states is as follows:

Tc	Data Time
Ts	Address Time
Tc	Data Time
Ts	Address Time
Tc	Data Time
Tc	Data Time
Ts	Address Time

Every bus cycle consists of at least an address time and data time pair. Additional data times (wait states) may be inserted in the bus cycle if the microprocessor samples its READY# input deasserted at the end of data time. Based on these rules, the states illustrated on the diagram can be interpreted as follows:

Tc	Data Time	end of previous bus cycle
Ts	Address Time	start of bus cycle A
Tc	Data Time	1st data time of bus cycle A
Ts	Address Time	start of bus cycle B
Tc	Data Time	1st data time of bus cycle B
Tc	Data Time	wait state inserted in bus cycle B
Ts	Address Time	start of next bus cycle

In other words, the diagram illustrates the end of a bus cycle followed by a 0-wait-state bus cycle (bus cycle A), a 1-wait-state bus cycle (bus cycle B) and the start of another bus cycle.

When reading a timing diagram, one scans from left-to-right (from earlier in time to later in time) looking for the signals that change first, second, and so on. Since PCLK should always be pulsing (changing), it's not necessary to note the fact that it changes.

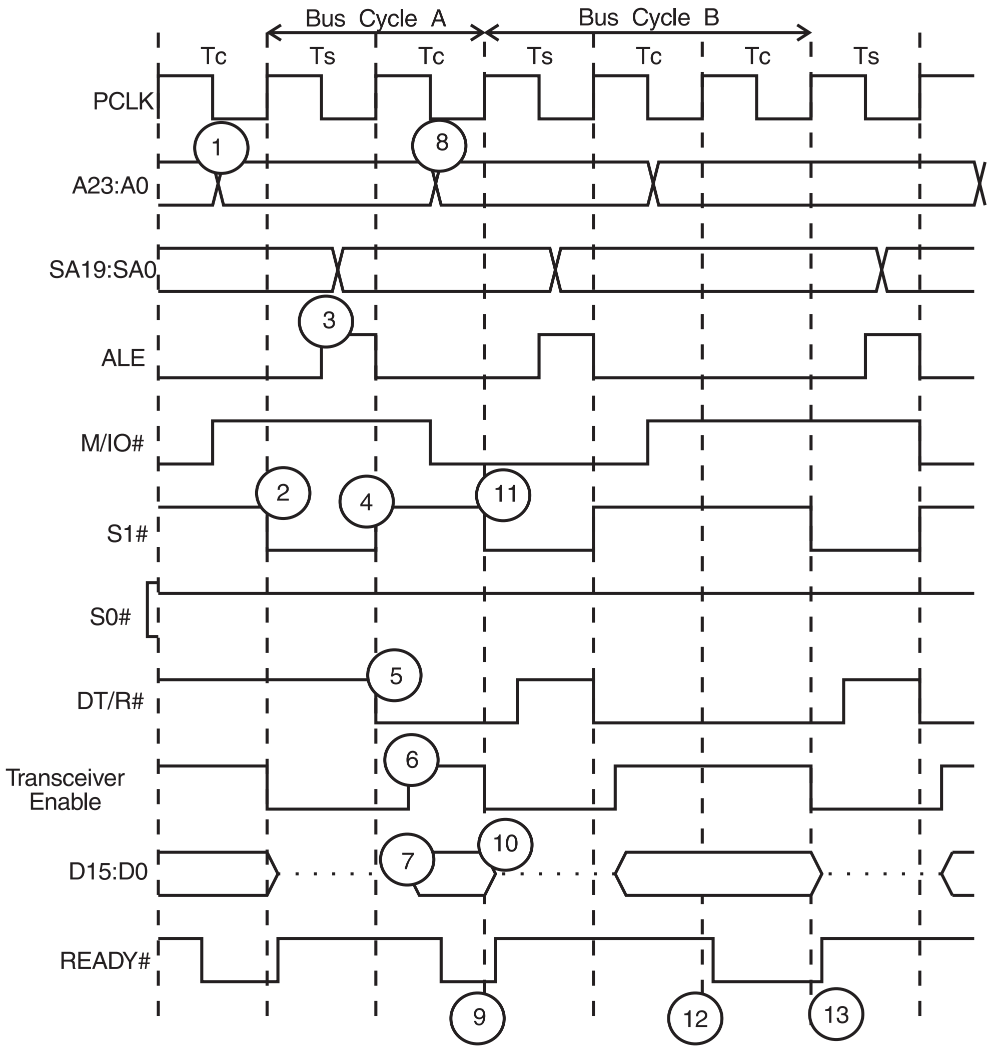


Figure 9-1. The Read Bus Cycle

Figure 9-2 shows the 80286 system engine. Referencing both the timing diagram and the functional block diagram will help tie the timing and functions together for a more complete understanding.

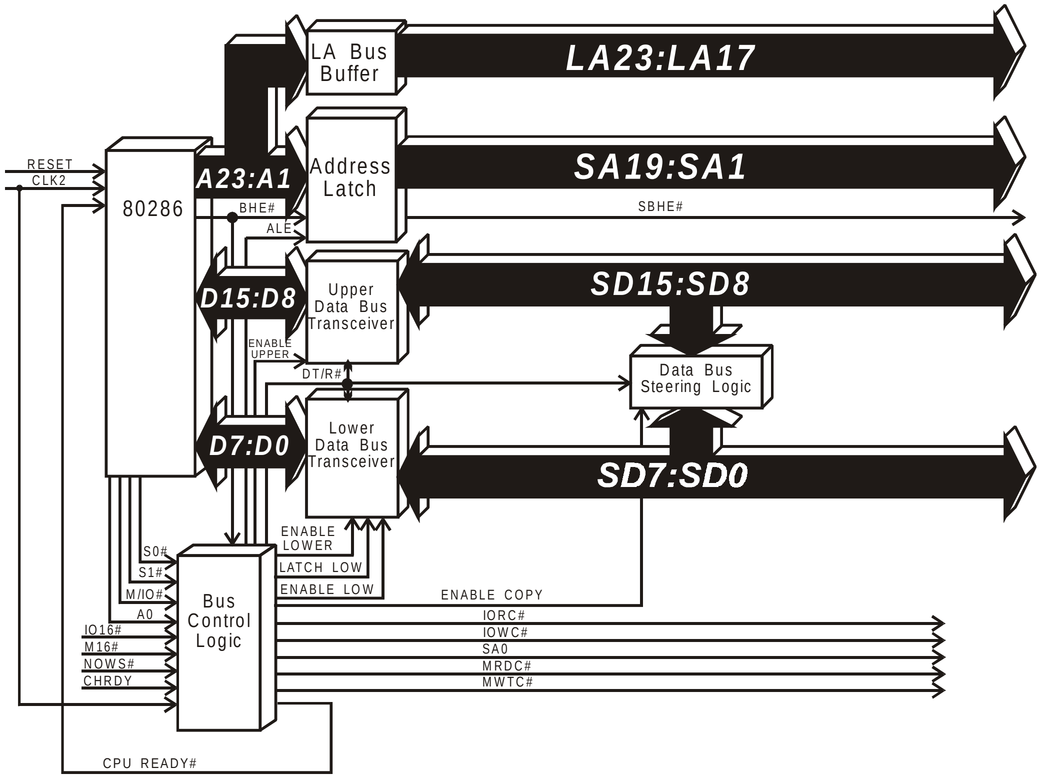


Figure 9-2. The 80286 System Engine

Chapter 10

The Previous Chapter

A detailed analysis of the 80286 microprocessor, the kernel logic necessary to interface it to external devices, and its bus cycle types were covered in prior chapters.

This Chapter

This chapter provides a detailed description of the 80386DX and SX microprocessors. It covers the functional units, processor self-test, enhancements to protected mode, virtual paging, virtual 8086 mode, and the processor's hardware interface to external devices.

The Next Chapter

The next chapter, “The 80386 System Kernel,” defines the support logic necessary to interface the 80386DX and SX microprocessors to external devices.

Introduction

This chapter describes the 80386DX and SX microprocessors. The 80386DX and SX microprocessors are identical internally and only differ in their interface to external devices. This chapter is therefore divided into three parts:

- a discussion of the internal structure and operation of the 80386 microprocessor.
- a discussion of the 80386DX interface to external devices.
- a discussion of the 80386SX interface to external devices.

The 80386 Functional Units

General

Figure 10-1 illustrates the units that make up the 80386 microprocessor.

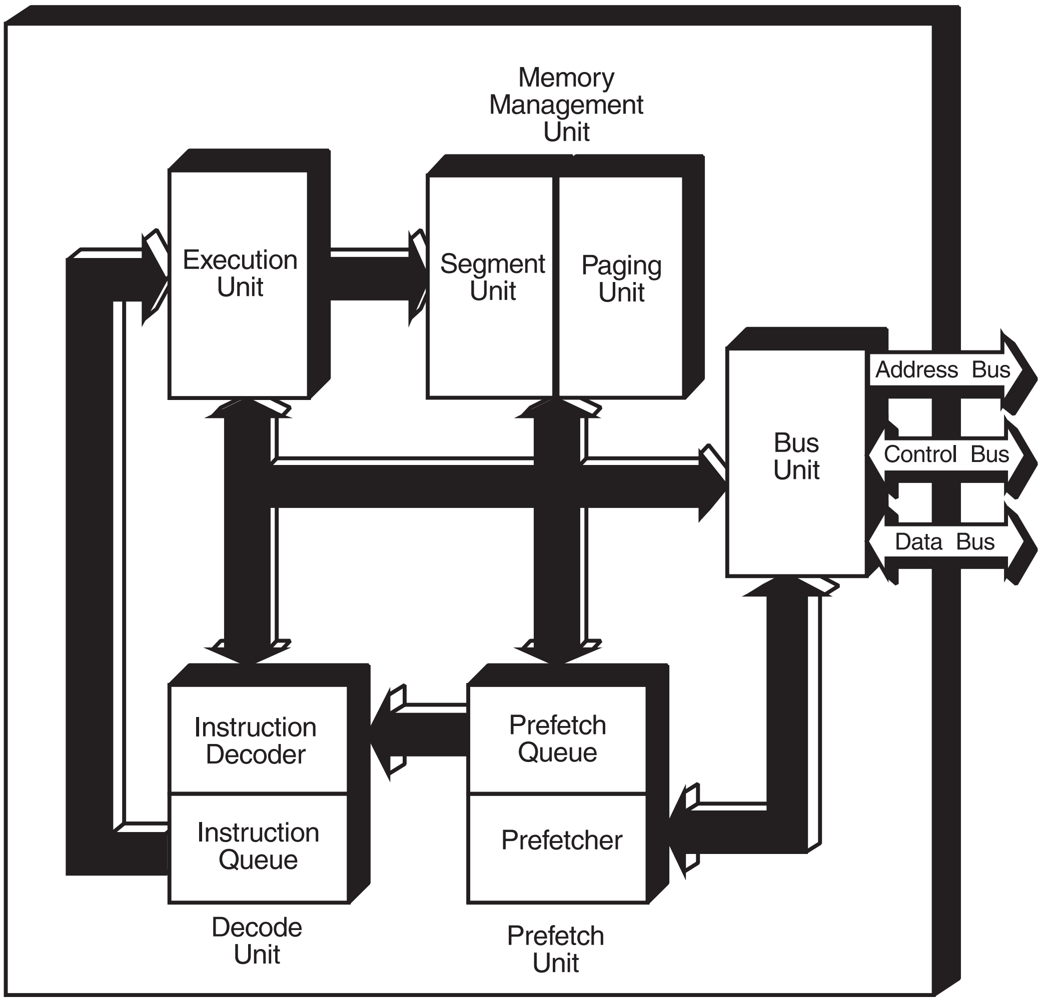


Figure 10-1. 80386 Microprocessor Block Diagram

The 80386 microprocessor consists of five functional units:

- Bus Unit. Handles communication with devices external to the microprocessor chip.
- Code Prefetch Unit. Fetches instructions from memory before the microprocessor actually requests them.
- Instruction Decode Unit. Decodes the instruction prior to passing it to the execution unit for execution.
- Execution Unit. Handles the actual execution of the instruction.
- Memory Management Unit (MMU). When the microprocessor must address a memory location, the MMU forms the physical memory address that is driven out onto the address bus by the bus unit during a bus cycle.

Code Prefetch Unit

The code prefetch unit capitalizes on the predictability of program execution. Statistically, program instructions are fetched from memory sequentially most of the time. Only jump instructions alter program flow, causing it to jump to another area of memory. Once the flow alteration has occurred, however, the program returns to sequential instruction processing until another jump instruction is encountered.

When the bus unit isn't performing bus cycles for the execution unit, the code prefetch unit uses the bus unit to fetch the next sequential instruction from memory and stores it in the 16-byte prefetch queue. The code prefetch unit always attempts to maximize the bus unit's throughput by fetching an entire doubleword (four bytes) at a time. The prefetched instructions are placed in the prefetch queue to await processing by the instruction decode unit.

Code prefetch requests are given a lower priority by the bus unit than requests from the execution unit. This ensures that the execution unit will not be stalled while waiting for a memory or I/O data transfer to complete. For all practical purposes, instruction prefetching reduces the time the execution unit spends waiting for the next instruction to zero.

Instruction Decode Unit

The instruction decode unit takes an instruction from the 16-byte prefetch queue, converts it into microcode and stores it in a three-deep decoded instruction queue for use by the execution unit. Any immediate data and offset associated with the instruction is also taken from the prefetch queue and stored in the decoded instruction queue.

Execution Unit

General

The execution unit executes each instruction received from the decoded instruction queue. It communicates with the other microprocessor functional units on an as needed basis in order to execute each instruction. The execution unit is comprised of three sub-units:

- The control unit's microcode and special parallel hardware speeds multiplies, divides and effective address calculation.
- The data unit contains the arithmetic logic unit, or ALU, eight general-purpose registers and a 64-bit barrel shifter. The data unit performs data operations requested by the control unit.
- The protection test unit checks for segmentation violations.

The Registers

The 80386 execution unit incorporates the following registers:

- The general registers
- The status and control registers
- The debug registers
- The test registers

The following sections describe each of these register groups.

General Registers

The general registers implemented in the 80386 microprocessor are a superset of those found in the earlier implementations of the X86 processor family. Figure 10-2 illustrates the general registers.

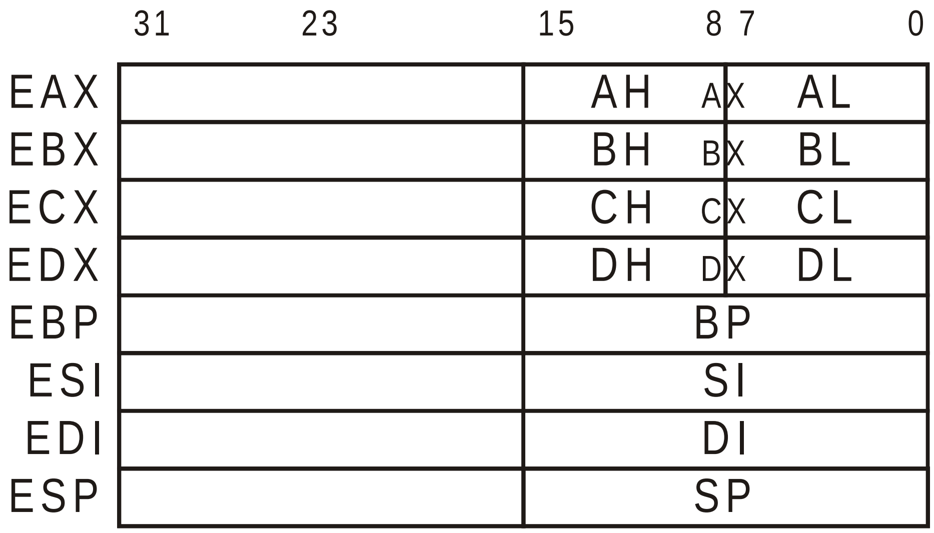


Figure 10-2. The 80386 General Registers

Extended versions of the AX, BX, CX, DX, BP, SI, DI and SP registers are now available to the programmer. The names of each of the extended registers starts with the letter "E". Referring to the EAX, EBX, ECX or EDX registers within a MOV instruction allows the programmer to specify a doubleword, or 32-bit, object to be moved. Extending the width of BP, SI, DI and SP registers to 32 bits permits the programmer to specify any memory address within the 4GB addressing range of the microprocessor without changing the contents of the segment registers.

Status, MSW and Instruction Registers

Figure 10-3 illustrates the 80386's status and instruction registers.

Appendix B

Glossary

access time - The time span between a device being addressed and when it delivers or accepts valid data.

active - see asserted

address bus - The group of signal lines that carry an address from a bus master.

address decoder - Every device that can be read from or written to has an associated piece of logic called an address decoder. The address decoder selects its associated device when an address within its defined range is detected.

address latch - A device connected to the microprocessor's local address bus. This device drives the address to all devices throughout the system and latches (holds) the address for the duration of the bus cycle. Once latched, the Address Latch outputs the address to the system on the system address (SA) bus, SA19:SA1.

address pipelining - A process in which the microprocessor places the address for the next cycle on its local address bus during the current bus cycle. This can reduce access time to system memory if designed to take advantage of address pipelining.

address translation - The process of converting one type of address to another. For example: translating the address from an ISA bus master (SA16:SA0, LA23:LA17 and SBHE#) to a 32-bit memory address (A31:A2 and BE3#:BE0#) required by 32-bit memory.

AEN - An ISA bus signal or a pin on the 8237 DMA controllers. Note that the same name is used for two differently-acting signals. When the CPU asserts the HLDA signal, the ISA bus AEN signal is asserted, disabling all I/O address decoders to prevent I/O devices from mistaking the address as theirs during DMA transfers. When the DMA controller sees its HLDA input asserted, it asserts its AEN pin to enable the outputs of its address latch and page register.

ALE - The Address Latch Enable signal that controls the operation of the Address Latch.

arbiter - An arbiter is a device that evaluates the pending requests for access to the bus and grants the bus to a bus master based on a system-specific algorithm.

asserted - A signal is asserted when it is at its logic true state. A signal such as HOLD is asserted when it is high (positive logic). A signal such as MRDC# is asserted when it is low (negative logic). A signal such as W/R# is always asserted since both its high and low states represent true logic states. The opposite of asserted is deasserted. The term "asserted" is used in this book instead of "active" because "active" is ambiguous. The term "active" can mean either that a high impedance buffer's output is on (driven), or that a signal is at a logic true state. For example, a tri-state buffer may be actively driving CHRDY low (false), in which case the signal is active (by one of the ambiguous definitions) but inactive (by the other definition). In this case, CHRDY is deasserted.

BALE - An ISA bus signal that is used by expansion devices to notify them that a valid address is on the ISA bus. BALE is a buffered version of ALE on 8088- and

80286-based PC's, but it is asserted at different times than ALE on 80386 and higher PC's.

BCLK - An ISA bus signal (bus clock) that provides the timing reference for all bus transactions.

big-endian byte-ordering - A bus master using big-endian byte-ordering stores the MSB from the specified source register into the start memory address and the less-significant bytes from the register into the ascending memory locations immediately following the start address.

BIOS - Basic Input/Output System routines usually contained in system ROM and device ROMs that provide a low level interface to devices.

bit cell - A one bit storage location in a typical DRAM consisting mainly of a capacitor.

block - Same as cache line in X86 processors. The cache stores information in its cache in blocks of a fixed length. The cache block length is processor design-dependent. As an example, an 80486 cache block is 16 bytes long.

block transfer mode - A DMA transfer mode in which an entire block of data is transferred prior to the DMA Controller giving up ownership of the buses.

bridge - The device that provides the interface between two independent buses. An example is the Bus Control Logic and associated transceivers between the 80386 host processor's bus and the ISA bus.

buffered write-through - A variation of the write-through policy where a look-through cache controller stores an entire write operation in a buffer so that it can complete the memory write to main memory later.

bus concurrency - Separate transfers occurring simultaneously on two or more separate buses. An example would be an ISA bus master transferring data to or from another ISA device while the host processor is transferring data to or from system memory. Bus concurrency allows a processor to gain access to memory from its memory cache, while another bus master accesses main memory over the system bus at the same time.

bus control logic - The logic that assists the microprocessor in running bus cycles over the ISA bus.

bus master, ISA - An ISA card that can gain ownership of the ISA buses and run bus cycles.

bus snooping - Bus snooping is the method used by cache subsystems to monitor main memory accesses made by another bus master.

byte enables - 80386 and higher X86 processor address bus signals that indicate which data paths will be used during a transfer. The byte enables indicate which bytes within an addressed doubleword are being accessed.

cache - A relatively small amount of high-speed static RAM (SRAM) that is used to keep copies of information recently read from system DRAM memory. The cache controller maintains a directory that tracks the information currently resident within the cache. If the host processor should request any of the information currently resident in the cache, it will be returned to the processor quickly (due to the fast access time of the SRAM).

cache coherency - If the information resident in a cache accurately reflects the information in DRAM memory, the cache is said to be coherent or consistent.

cache consistency - see **cache coherency**

cache controller - A cache memory controller manages cache memory which stores copies of frequently accessed information read from DRAM memory.

cache directory - Memory inside of a cache controller that keeps track of information stored in cache memory. Sometimes called the tag RAM.

cache FLUSH# - A cache controller input that causes the cache controller to clear the valid bit in every directory entry to 0, thereby invalidating all cache entries.

cache line - A line is the smallest unit of data that a cache can keep track of.

cache line fill - When a processor's internal cache, or its external second level cache has a miss on a read attempt by the processor, it will read a fixed amount (referred to as a line) of information from the external cache or system DRAM memory and record it in the cache. This is referred to as a cache line fill. The size of a line of information is cache controller design dependent.

cache memory - Cache memory is a relatively small amount of high cost, fast access SRAM designed to improve access to system memory.

cache page - The size of the cache data RAM in direct-mapped caches and the size of each way in a set-associative cache. The cache views main memory as divided into pages that are the same size as the cache pages.

cache read hit - The process in which the cache memory controller sees the microprocessor initiate a memory read bus cycle, checks to determine if it has a copy of the requested information in cache memory, and if a copy is present, immediately reads the information from the cache and sends it back to the microprocessor at zero wait-states.

cache read miss - The process in which the cache memory controller sees the microprocessor initiate a memory read bus cycle, checks to determine if it has a copy of the requested information in cache memory, and find no copy is present. The cache memory controller then passes the read bus cycle on to slow main memory.

CAS# - A memory enable called Column Address Strobe used to latch the column portion of the address inside a DRAM chip.

CAS before RAS refresh - Some DRAMs incorporate their own row counters to be used for DRAM refresh. The external DRAM refresh logic has only to assert the DRAM's CAS line and then assert its RAS line. The DRAM will automatically increment its internal row counter and refresh (recharge) the next row of storage.

CBR refresh - see **CAS before RAS refresh**

channel check - This signal is asserted by an ISA expansion board to signal a catastrophic error condition to the microprocessor.

channel ready - By using the CHRDY signal (Channel Ready) on the ISA bus, designers can override the default timer and stretch out the bus cycle by the required number of wait states to match the access time of the device.

CHCHK# - see **channel check**

CHRDY - see **channel ready**

CMOS RAM - see **configuration RAM**

coherency - see **cache coherency**

concurrent bus operation - see **bus concurrency**

configuration RAM - A small amount of low power battery-backed memory used to store configuration information used during the boot-up process.

consistency - see **cache coherency**

control bus - A group of signal lines used by the microprocessor to control a variety of miscellaneous functions.

cycle time - The amount of time needed between accesses to DRAM memory, consisting of access time plus precharge delay.

DAKn# - A group of signals on the ISA bus (named DACKn# or DAKn#) used to notify DMA I/O devices that a DMA cycle is about to be performed to transfer data for them.

data bus - The group of signal lines used to transfer data between devices.

data bus contention - An address decoding problem that arises if the address decoders for two devices are both designed to detect the same address range. In such cases both devices would deliver data to the data bus simultaneously during a read operation, causing contention.

data bus steering logic - Logic used to match transfers between devices with different sized data busses. Consists of transceivers connected between the data paths.

deasserted - A signal is deasserted when it is at its logic false state. See **asserted**.

debug registers - Six programmer-accessible debug registers provide on-chip support for debugging and are present in the 80386. Debug registers DR0 through DR3 specify the four linear breakpoint addresses. The Breakpoint Control Register, DR7, is used to set the breakpoints, define them as data or code breakpoints, and whether to break on an attempted read or write. The Breakpoint Status Register, DR6, reflects the current state of the breakpoints.

demand transfer mode - The transfer mode where the DMAC runs DMA bus cycles back-to-back as long as the I/O device can continue to supply data.

device ROM - ROM on an ISA expansion device. Also known as expansion ROM. This ROM typically contains the initialization code and possibly the BIOS and interrupt service routines for its associated device. Device ROMs should reside in memory address space between C0000h and DFFFFh.

direct-mapped cache - A cache organization in which only one directory entry needs to be checked to determine whether the requested memory location is in the cache. A direct-mapped cache is the same size as a (cache) page in memory. When a line of information is fetched from a position within a page of DRAM

Index